33rd Symposium on Theoretical Aspects of Computer Science

STACS'16, February 17–20, 2016, Orléans, France

Edited by

Nicolas Ollinger

Heribert Vollmer

LIPICS

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

LIPIcs – Vol. 47 – STACS'16 www.dagstuhl.de/lipics

Editors

Nicolas Ollinger
LIFO
Université d'Orléans
45067 Orléans Cedex 2
France
nicolas.ollinger@univ-orleans.fr

Heribert Vollmer
Institut für Theoretische Informatik
Leibniz Universität Hannover
30167 Hannover
Germany
vollmer@thi.uni-hannover.de

ACM Classification 1998
F.1.1 Models of Computation, F.2.2 Nonnumerical Algorithms and Problems, F.4.1 Mathematical Logic, F.4.3 Formal Languages, G.2.1 Combinatorics, G.2.2 Graph Theory

ISBN 978-3-95977-001-9

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/978-3-95977-001-9.

Publication date
February, 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.

License
This work is licensed under a Creative Commons Attribution 3.0 Unported license (CC-BY 3.0): http://creativecommons.org/licenses/by/3.0/legalcode.

In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.STACS.2016.0

ISBN 978-3-95977-001-9 **ISSN 1868-8969** **http://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

Editorial Board

- Susanne Albers (TU München)
- Chris Hankin (Imperial College London)
- Deepak Kapur (University of New Mexico)
- Michael Mitzenmacher (Harvard University)
- Madhavan Mukund (Chennai Mathematical Institute)
- Catuscia Palamidessi (INRIA)
- Wolfgang Thomas (*Chair*, RWTH Aachen)
- Pascal Weil (CNRS and University Bordeaux)
- Reinhard Wilhelm (Saarland University)

ISSN 1868-8969

http://www.dagstuhl.de/lipics

Foreword

The Symposium on Theoretical Aspects of Computer Science (STACS) conference series is an international forum for original research on theoretical aspects of computer science. Typical areas are (cited from the call for papers for this year's conference): algorithms and data structures, including: parallel, distributed, approximation, and randomized algorithms, computational geometry, cryptography, algorithmic learning theory, analysis of algorithms; automata and formal languages; computational complexity, parameterized complexity, randomness in computation; logic in computer science, including: semantics, specification and verification, rewriting and deduction; current challenges, for example: natural computing, quantum computing, mobile and net computing.

STACS is held alternately in France and in Germany. This year's conference (taking place February 17–20 in Orléans) is the 33rd in the series. Previous meetings took place in Paris (1984), Saarbrücken (1985), Orsay (1986), Passau (1987), Bordeaux (1988), Paderborn (1989), Rouen (1990), Hamburg (1991), Cachan (1992), Würzburg (1993), Caen (1994), München (1995), Grenoble (1996), Lübeck (1997), Paris (1998), Trier (1999), Lille (2000), Dresden (2001), Antibes (2002), Berlin (2003), Montpellier (2004), Stuttgart (2005), Marseille (2006), Aachen (2007), Bordeaux (2008), Freiburg (2009), Nancy (2010), Dortmund (2011), Paris (2012), Kiel (2013), Lyon (2014), and München (2015).

The interest in STACS has remained at a high level over the past years. The STACS 2016 call for papers led to 205 submissions with authors from 44 countries. Each paper was assigned to three program committee members who, at their discretion, asked external reviewers for reports. The committee selected 54 papers during a three-week electronic meeting held in November/December. For the second time within the STACS conference series, there was also a rebuttal period during which authors could submit remarks to the PC concerning the reviews of their papers. As co-chairs of the program committee, we would like to sincerely thank all its members and the many external referees for their valuable work. In particular, there were intense and interesting discussions. The overall very high quality of the submissions made the selection a difficult task.

This year, the conference includes a tutorial. We would like to express our thanks to the speaker Jarkko Kari for this tutorial, as well as to the invited speakers, Jérôme Leroux, Carsten Lutz, and Virginia Vassilevska Williams. Special thanks also go to Andrei Voronkov for his EasyChair software (`http://www.easychair.org`). Moreover, we would like to warmly thank Isabelle Renard and Fabienne Le Bihan for continuous help throughout the conference organization.

We would also like to thank Marc Herbstritt from the Dagstuhl/LIPIcs team for assisting us in the publication process and the final production of the proceedings. These proceedings contain extended abstracts of the accepted contributions and abstracts of the invited talks and the tutorials. The authors retain their rights and make their work available under a Creative Commons license. The proceedings are published electronically by Schloss Dagstuhl – Leibniz-Center for Informatics within their LIPIcs series.

STACS 2016 has received funds and help from the University of Orléans, the Région Centre-Val de Loire, the Département du Loiret, the Mairie d'Orléans, the CNRS, the lab LIP at the École Normale Supérieure de Lyon and the lab LIFO at the University of Orléans. We thank them for their support!

Orléans and Hannover, February 2016 — Nicolas Ollinger and Heribert Vollmer

Conference Organization

Program Committee

Isolde Adler	Goethe University Frankfurt
Nathalie Bertrand	Inria Rennes
Nader Bshouty	Technion, Haifa
Arkadev Chattopadhyay	Tata Institute of Fundamental Research – Mumbai
Philippe Duchon	Université de Bordeaux
Henning Fernau	Universität Trier
Samuel Fiorini	Université libre de Bruxelles
Danny Hermelin	Ben-Gurion University of the Negev
Rahul Jain	National University of Singapore
Artur Jeż	University of Wrocław
Stefan Kiefer	University of Oxford
Andreas Krebs	Eberhard Karls University, Tübingen
Gregory Kucherov	CNRS
Sławomir Lasota	University of Warsaw
Guillaume Malod	Université Paris Diderot
Conrado Martínez	Universitat Politècnica de Catalunya
Nicole Megow	Technische Universität München
Nicolas Ollinger	Université d'Orléans *(co-chair)*
Giovanni Pighizzini	Università degli Studi di Milano
Dror Rawitz	Bar Ilan University
Christian Sohler	Technische Universität Dortmund
Stefan Szeider	TU Wien
Suresh Venkatasubramanian	University of Utah
Heribert Vollmer	Leibniz Universität Hannover *(co-chair)*
James Worrell	University of Oxford
Marc Zeitoun	Université de Bordeaux

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

Local Organization Committee

Tom Besson
Jérôme Durand-Lose
Valentin Garnero
Diego Maldonado
Pedro Montealegre
Viven Pelletier
Anthony Perez
Isabelle Renard
Ioan Todinca *(chair)*

External Reviewers

Amir Abboud
Faisal Abu-Khzam
Anna Adamaszek
Dan Alistarh
Aris Anagnostopoulos
Alexandr Andoni
Patrizio Angelini
Spyros Angelopoulos
Antonios Antoniadis
Vikraman Arvind
Per Austrin
Yossi Azar
Arturs Backurs
Max Bannach
Luis Barba
Nicolas Basset
Eli Ben-Sasson
Michael Benedikt
Olaf Beyersdorff
Maria Paola Bianchi
Marcin Bieńkowski
Laurent Bienvenu
Achim Blumensath
Manuel Bodirsky
Hans L. Bodlaender
Andrej Bogdanov
Mikolaj Bojanczyk
Benedikt Bollig
Ilario Bonacina
Marthe Bonamy
Nicolas Bonichon
Flavia Bonomo
Vivek Borkar
Yacine Boufkhad
Marin Bougeret
Nicolas Bousquet
Simone Bova
Tomas Brazdil
Karl Bringmann
Gerth Stølting Brodal
Laurent Bulteau
Marc Bury
Jaroslaw Byrka
Michaël Cadilhac
Leizhen Cai
Shaowei Cai
Clément Canonne
Yixin Cao
Jean Cardinal
Arturo Carpi
Olivier Carton
Katarina Cechlarova
Keren Censor-Hillel
Douglas Cenzer
Marco Cerami
Deeparnab Chakrabarty
Parinya Chalermsook
Maurice Chandoo
Witold Charatonik
Lin Chen
Shiteng Chen
Dehua Cheng
Dmitry Chistikov
Christian Choffrut
Ventsislav Chonev
Lorenzo Clemente
Maxime Crochemore
Marek Cygan
Silke Czarnetzki
Wojciech Czerwiński
Stefan Dantchev
Adnan Darwiche
Samir Datta
Alberto Dennunzio
Yann Disser
Paul Dorbec
Rod Downey
Laurent Doyen
Manfred Droste
Ran Duan
Vida Dujmovic
Arnaud Durand
Stephane Durocher
Charilaos Efthymiou
Kord Eickmeyer
Michael Elberfeld
Matthias Englert
Yuri Faenza
Angelo Fanelli
Nazim Fatès
John Fearnley
Cristina Fernandes
Hendrik Fichtenberger
Gabriele Fici
Santiago Figueira
Nathanaël Fijalkow
Eldar Fischer
Vissarion Fisikopoulos
Dimitris Fotakis
Hervé Fournier
Tobias Friedrich
Hanna Furmańczyk
Eric Fusy
Shmuel Gal
Andreas Galanis
Alex Galicki
Ziyuan Gao
Ankit Garg
Naveen Garg
Leszek Gasieniec
Serge Gaspers
Paul Gastin
Tomáš Gavenčiak
Paweł Gawrychowski
Blaise Genest
Konstantinos Georgiou
George Giakkoupis
Archontia Giannopoulou
Christian Glasser
Emmanuel Godard
Wayne Goddard
Leslie Ann Goldberg
Massimiliano Goldwurm
Petr Golovach
Fabrizio Grandoni
Katarzyna Grygiel
Jiong Guo
Stefan Göller
Mika Göös
Demen Güler
Christoph Haase
Torben Hagerup
Matthew Hague
Michael Hahn
Mohammadtaghi Hajiaghayi
Vesa Halava
Jie Han
Yo-Sub Han
Kristoffer Arnsfelt Hansen
Nicolas Hanusse
Tobias Harks

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

External Reviewers

Prahladh Harsha
Jason Hartline
Loic Helouet
Miki Hermann
Petr Hlineny
Martin Hoefer
Piotr Hofman
Markus Holzer
Florian Horn
Mathieu Hoyrup
Paul Hunter
Tony Huynh
Falk Hüffner
Leo van Iersel
Takehiro Ito
Dmitry Itsykson
Navendu Jain
Sanjay Jain
Bart M. P. Jansen
Klaus Jansen
Emmanuel Jeandel
Benson Joeris
Matthew Johnson
Timo Jolivet
Peter Jonsson
Brendan Juba
Valentine Kabanets
Gautam Kamath
Iyad Kanj
Mamadou Moustapha Kanté
Jarkko Kari
Telikepalli Kavitha
Alexandr Kazda
Eun Jung Kim
Philipp Kindermann
Marek Klonowski
Tomasz Kociumaka
Kirill Kogan
Sven Köhler
Mikko Koivisto
Roman Kolpakov
Christian Komusiewicz
Alexander Kononov
Eryk Kopczynski
Guy Kortsarz
Dmitry Kosolobov
Łukasz Kowalik
Andreas Krall
Dieter Kratsch
Stefan Kratsch
Jan Krcal
Jan Kretinsky
Andrei Krokhin
Ralf Küsters
Manfred Kufleitner
Sebastian Kuhnert
Adam Kunysz
Denis Kuperberg
Gilad Kutiel
Guillaume Lagarde
Giovanna Lavado
Troy Lee
Yin Tat Lee
Erik Jan van Leeuwen
Christoph Lenzen
Jerome Leroux
Peter Leupold
Ming Li
Minming Li
Nutan Limaye
Anthony Widjaja Lin
Markus Lohrey
Daniel Lokshtanov
Sylvain Lombardy
Michael Ludwig
Christof Löding
Meena Mahajan
Mohammad Mahdian
Cécile Mailler
Yury Makarychev
Andreas Malcher
Nikhil Mande
Florin Manea
Sebastian Maneth
Nicolas Markey
Barnaby Martin
Russell Martin
Dániel Marx
Maarten Marx
Monaldo Mastrolilli
Luke Mathieson
Pierre Mckenzie
Moti Medina
Kitty Meeks
Ruta Mehta
Arne Meier
Stefan Mengel
Carlo Mereghetti
Stephan Mertz
Ron Van Der Meyden
Pierre-Étienne Meunier
Henryk Michalewski
Pierre Michaud
Peter Bro Miltersen
Shuichi Miyazaki
Matthias Mnich
Benjamin Monmege
Nelma Moreira
Dana Moshkovitz
Amer Mouawad
Priyanka Mukhopadhyay
Sagnik Mukhopadhyay
Alexander Munteanu
Filip Murlak
Hadrien Mélot
Norbert Th. Müller
Satyadev Nandakumar
Jesper Nederlof
Ofer Neiman
Dang Phuong Nguyen
André Nies
Naomi Nishimura
Stefan Näher
Jan Obdrzalek
Joanna Ochremiak
Sebastian Ordyniak
Sang-Il Oum
Umut Oztok
Dominik Pajak
Katarzyna Paluch
Fahad Panolan
Charles Paperman
Merav Parter
Paweł Parys
Kanstantsin Pashkovich
Matthew Patitz
Ami Paz
Pan Peng
Vianney Perchet
Vitaly Perevoshchikov
Sylvain Perifel
Reinhard Pichler
Marcin Pilipczuk
Michał Pilipczuk
Sophie Pinchinat
Michael Pinsker
Joao Sousa Pinto
Thomas Place
Alexandru Popa
Anupam Prakash

Kirk Pruhs
Gabriele Puppis
Manish Purohit
Svetlana Puzynina
Karin Quaas
Arash Rafiey
Mukund Raghothaman
Narad Rampersad
Igor Razgon
Klaus Reinhardt
Marc Renault
Selim Rexhep
Leonid Reyzin
Gaétan Richard
Jérémie Roland
Adi Rosen
Günter Rote
Nicolas de Rugy-Altherre
Ramanujan M. S.
Akshay S
Mehrnoosh Sadrzadeh
Rishi Saket
Michael Saks
Rahul Saladi
Felix Salfelder
Sylvain Salvati
Arnaud Sangnier
Ocan Sankur
Rahul Santhanam
Nitin Saurabh
Saket Saurabh
Sven Schewe
Melanie Schmidt
Tina Janne Schmidt
Henning Schnoor
Roy Schwartz
Francois Schwarzentruber
Pascal Schweitzer
Chris Schwiegelshohn
Marinella Sciortino
Shinnosuke Seki
Pranab Sen
Geraud Senizergues
Daniel Severin
Hadas Shachnai
Mahsa Shirmohammadi
Aaron Sidford
Matthew Skala
Michał Skrzypczak
Martin Skutella
Eric Sopena
Joachim Spoerhase
A V Sreejith
Karteek Sreenivasaiah
Srikanth Srinivasan
Venkatesh Srinivasan
Rob van Stee
Clifford Stein
Frank Stephan
Christoph Stockhusen
Howard Straubing
Yann Strozecki
Andrew Suk
Marco Di Summa
Scott Summers
Xiaoming Sun
Ola Svensson
Tami Tamir
Christino Tamon
Till Tantau
Jan Arne Telle
Véronique Terrier
Pascal Tesson
Chris Thachuk
Nguyen Kim Thang
Johan Thapper
Dirk Oliver Theis
Guillaume Theyssier
Ioan Todinca
Szymon Toruńczyk
Jacobo Torán
Denis Trystram
Max Tschaikowski
Ilkka Törmä
Oleg Verbitsky
José Verschae
Antoine Vigneron
Eric Vigoda
Fernando Sanchez Villaamil
Jonni Virtema
Paul M.B. Vitanyi
Imrich Vrťo
Magnus Wahlström
Bartosz Walczak
Daria Walukiewicz-Chrząszcz
Kunihiro Wasa
Thomas Watson
Pascal Weil
Oren Weimann
Mathias Weller
Matthias Westermann
Virginia Vassilevska Williams
John Wilmes
Dominik Wojtczak
Prudence W.H. Wong
Jinhui Xu
Abuzer Yakaryilmaz
Tomoyuki Yamakami
Jonathan Yaniv
Haifeng Yu
Bruno Zanuttini
Meirav Zehavi
Akka Zemmari
Rico Zenklusen
Louxin Zhang
Hang Zhou
Martin Zimmermann
Stanislav Živný

Contents

Invited talks

Tutorial

Regular contributions

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Ideal Decompositions for Vector Addition Systems

Jérôme Leroux[1] and Sylvain Schmitz[2]

1 **LaBRI, Univ. Bordeaux & CNRS, France**
leroux@labri.fr
2 **LSV, ENS Cachan & CNRS & INRIA, Université Paris-Saclay, France**
schmitz@lsv.ens-cachan.fr

Abstract

Vector addition systems, or equivalently Petri nets, are one of the most popular formal models for the representation and the analysis of parallel processes. Many problems for vector addition systems are known to be decidable thanks to the theory of well-structured transition systems. Indeed, vector addition systems with configurations equipped with the classical point-wise ordering are well-structured transition systems. Based on this observation, problems like coverability or termination can be proven decidable.

However, the theory of well-structured transition systems does not explain the decidability of the reachability problem. In this presentation, we show that runs of vector addition systems can also be equipped with a well quasi-order. This observation provides a unified understanding of the data structures involved in solving many problems for vector addition systems, including the central reachability problem.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Petri net, ideal, well-quasi-order, reachability, verification

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.1

Category Invited Talk

1 Introduction

Vector Addition Systems and Well-Structured Transition Systems. *Vector addition systems with states* (VASS), or equivalently Petri nets, find a wide range of applications in the modelling of concurrent, chemical, biological, or business processes. They are defined as tuples $\mathcal{V} = \langle Q, d, T\rangle$ where Q is a finite set of states, d is a dimension in $\mathbb{N}$, and T is a finite set of transitions in $Q \times \mathbb{Z}^d \times Q$ (Figure 1 displays an example). A VASS gives rise to an infinite transition system over the set of *configurations* $Confs \stackrel{\text{def}}{=} Q \times \mathbb{N}^d$ by allowing a *step* $(q, \boldsymbol{u}) \xrightarrow{t} (q', \boldsymbol{u}+\boldsymbol{a})$ for all $\boldsymbol{u} \in \mathbb{N}^d$ and $t = (q, \boldsymbol{a}, q') \in T$ such that $\boldsymbol{u}+\boldsymbol{a} \geq \mathbf{0}$. Many problems are decidable for VASS, notably

reachability: given $\mathcal{V}$ and two configurations c and c' in *Confs*, can c reach c' in a finite number of steps, noted $c \to^* c'$?

coverability: given the same inputs, does there exist $c'' \sqsupseteq c'$ such that $c \to^* c''$? Here we use the *product ordering*, i.e. we require $c' = (q, \boldsymbol{u}')$ and $c'' = (q, \boldsymbol{u}'')$ where $\boldsymbol{u}'(i) \geq \boldsymbol{u}''(i)$ for all $1 \leq i \leq d$.

These two decision problems form the algorithmic core of many decidability results – spanning from the verification of asynchronous programs [20] to the decidability of data logics [4, 12, 8] (see the references in [48] for more applications).

Vector addition systems are an instance of a more general class of systems with good algorithmic properties called (strict) *well-structured transition systems* (WSTS), and as


© Jérôme Leroux and Sylvain Schmitz;
licensed under Creative Commons License CC-BY

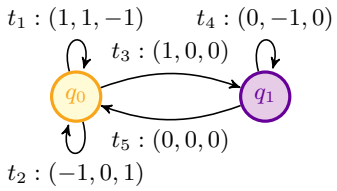


Figure 1 A 3-dimensional VASS (from [48]).

a result several problems are decidable using generic algorithms, including coverability, termination, and boundedness [1, 19]. These algorithms all rely on the existence of a *well-quasi-order* (wqo) on the set of configurations – here the product ordering $\sqsubseteq$ over *Confs* –, which is 'compatible' with the transition relation defined by the system at hand.

Ideals and Complete WSTS. The theory of WSTS alone does however not account for the decidability of several other problems on VASS, like *place boundedness*, which asks whether the reachable valuations of an input subset $K \subseteq \{1, \ldots, d\}$ of the components are bounded. The classical algorithm for this last problem relies instead on the fact that the set of configurations that might be covered starting from some initial configuration c, i.e. the *cover* (also called the *coverability set*)

$$Cover(c) \stackrel{\text{def}}{=} \{(q, \boldsymbol{u}) \in Confs \mid \exists \boldsymbol{u}' \geq \boldsymbol{u} \,.\, c \to^* (q, \boldsymbol{u}')\} = {\downarrow}\{c' \in Confs \mid c \to^* c'\} \tag{1}$$

is downwards-closed and computable thanks to a *coverability tree* construction first defined by Karp and Miller [28]. The construction proceeds forwards from c and explores the tree of reachable configurations, but employs *acceleration* to ensure finiteness (see Section 3 for details). Due to acceleration, the nodes of this tree are labelled by 'extended configurations' in $Q \times (\mathbb{N} \cup \{\omega\})^d$, where an ω value reflects a component that might become arbitrarily large in reachable configurations; the cover is then exactly the union of the downward closures ${\downarrow}c$ when c ranges over the labels in the tree.

The ingredients required to carry out such a construction in general have been identified by Finkel and Goubault-Larrecq [17, 18] with *complete* WSTS. This framework relies on the existence of

1. an acceleration procedure along finite traces of the system, and of
2. some means to finitely represent downwards-closed sets of configurations. Finkel and Goubault-Larrecq advocate for this the use of *ideals*, which provide canonical finite decompositions for downwards-closed subsets of a wqo (see Section 2). Finkel and Goubault-Larrecq also provide a range of effective representations for ideals; for instance, the ideals of $(Confs, \sqsubseteq)$ are exactly the sets ${\downarrow}c$ for $c \in Q \times (\mathbb{N} \cup \{\omega\})^d$ employed in Karp and Miller's construction.

Based on these two ingredients, the framework of Finkel and Goubault-Larrecq provides a generic procedure to compute a finite representation of the cover – without any general guarantee of termination: the cover is not always computable, e.g. for VASS extended with *transfer* operations (which are also strict WSTS) the place boundedness problem is undecidable [15].

The Reachability Problem. The decidability of the reachability problem for VASS is a famous result, first proven by Mayr [40] in 1981 after years of attempts and partial solutions, notably by Sacerdote and Tenney [46]. Mayr's algorithm and proof have since been clarified

and refined by Kosaraju [29] and Lambert [30]; we call the resulting algorithm the *KLMST algorithm* after its inventors. Put succinctly, this algorithm performs successive refinements on a finite set of structures (called respectively 'regular constraint graphs' by Mayr, 'generalised VASS' by Kosaraju, and 'marked graph-transition sequences' by Lambert), until a condition is fulfilled (called respectively 'consistent marking', 'θ condition', and 'perfectness'); at this point the algorithm terminates and can answer whether reachability holds depending on whether the resulting set is empty.

These results have at first sight little to do with WSTS, for which reachability is often undecidable (the case of transfer VASS is again an example). Nevertheless, a recent insight into the algorithm of Mayr, Kosaraju, and Lambert is that they compute an *ideal decomposition* for the set of runs from source to target configuration. More precisely, we show in [37] that the data structures manipulated in the KLMST algorithm are representations for run ideals, and that the result of the computation is exactly the ideal decomposition of the downward-closure of the set of runs (see Section 4).

Overview of the Talk. To sum up, ideals provide the data structures involved in both

- Karp and Miller's coverability tree algorithm, which computes the ideal decomposition of the cover using configuration ideals (Section 3), and
- the KLMST algorithm, which computes the ideal decomposition of the downward-closure of the set of runs using run ideals (Section 4).

The purpose of this talk is to present wqo ideals (Section 2) and overview their algorithmic applications through the coverability and KLMST procedures. We believe that the ideal point of view on those two classical algorithms could guide the principled development of algorithms for VASS extensions and other WSTS – in particular when the decidability status of the reachability problem is open, as for *unordered data Petri nets* [32], *branching VASS* (e.g. [47]), and *pushdown VASS* [31, 38]. We shall only provide the basic definitions and main statements here, but we provide pointers to the relevant literature for the interested reader.

2 Ideals for Well-Quasi-Orders

Quasi-Orders. A *quasi-order* (qo) $(X, \leq_X)$ combines a support set X with a transitive reflexive relation $\leq_X \subseteq X \times X$. Given a set S, its *downward-closure* is $\downarrow S \stackrel{\text{def}}{=} \{x \in X \mid \exists s \in S \,.\, x \leq_X s\}$; when S is a singleton $\{s\}$ we write more simply $\downarrow s$. A set $D \subseteq X$ is *downwards-closed* (also called *initial*) if $\downarrow D = D$.

Well-Quasi-Orders. A *well-quasi-order* (wqo) [23] is a qo with the *descending chain property*: all the chains $D_0 \supsetneq D_1 \supsetneq \cdots$ of downwards-closed subsets $D_j \subseteq X$ are finite. Equivalently, it has the *finite basis property*: any subset $S \subseteq X$ has a finite number of minimal elements. For instance,

finite sets: any finite set Σ equipped with equality forms a wqo $(\Sigma, =)$: its downwards-closed subsets are singletons $\{x\}$ for $x \in \Sigma$, and its chains are of length one;

natural numbers: the set of natural numbers $(\mathbb{N}, \leq)$ is a wqo: its downwards-closed subsets are either $\mathbb{N}$ itself or of the form $\downarrow n$ for $n \in \mathbb{N}$, and any chain $(\mathbb{N} \supsetneq) \downarrow n_0 \supsetneq \downarrow n_1 \supsetneq \cdots$ corresponds to a decreasing sequence $n_0 > n_1 > \cdots$ and is therefore finite;

Cartesian products: if $(X, \leq_X)$ and $(Y, \leq_Y)$ are wqos, then their Cartesian product $X \times Y$ equipped with the *product ordering* $\leq_{X \times Y}$ is also a wqo $(X, \leq_X) \times (Y, \leq_Y) \stackrel{\text{def}}{=} (X \times$

$Y, \leq_{X\times Y})$, where $(x, y) \leq_{X\times Y} (x', y')$ if and only if $x \leq_X x'$ and $y \leq_Y y'$ – this allows to prove *Dickson's Lemma* [14], which states that $(\mathbb{N}^d, \leq)$ is a wqo when ordered pointwise –;

finite sequences: if $(X, \leq_X)$ is a wqo, then the set X^* of finite sequences over X (sometimes also noted $X^{<\omega}$) equipped with the *embedding ordering* $\leq_{X^*}$ is also a wqo $(X, \leq_X)^* \stackrel{\text{def}}{=} (X^*, \leq_{X^*})$, where $x_0, \ldots, x_{m-1} \leq_{X^*} x'_0, \ldots, x'_{n-1}$ if and only if there exists a monotone injective function f from $\{0, \ldots, m-1\}$ to $\{0, \ldots, n-1\}$ such that $x_j \leq_X x'_{f(j)}$ for all $j \in \{0, \ldots, m-1\}$ – this allows to prove *Higman's Lemma* [23], which states that $(\Sigma^*, \leq_{\Sigma^*})$ for a finite alphabet $(\Sigma, =)$ is a wqo when ordered by subword embedding.

In the following, we will use these basic examples to construct wqos of VASS configurations (in Section 3) and of VASS runs (in Section 4).

Ideals. Let $(X, \leq_X)$ be a wqo. An *ideal* I of X is a non-empty, downwards-closed, and (up-)*directed* subset of X; this last condition enforces that, if x, x' are in I, then there exists $y \in I$ that dominates both: $x \leq_X y$ and $x' \leq_X y$.

The key property of wqo ideals we are going to use is that they provide finite decompositions for downwards-closed sets. This was first shown by Bonnet [5], and rediscovered in the context of complete WSTS (and generalised for Noetherian topologies) by Finkel and Goubault-Larrecq [17]:

▶ **Fact 1** (Canonical Ideal Decompositions). *Every downward-closed set over a wqo is the union of a unique finite family of incomparable (for the inclusion) ideals.*

Ideal Representations. Combined with the descending chain property, Fact 1 provides an abstract template for algorithms computing descending chains $D_0 \supsetneq D_1 \supsetneq \cdots$ of downwards-closed sets: this must terminate when working over a wqo, and furthermore each D_j can be represented as a finite set of ideals. The missing element here is how to effectively represent those ideals.

Depending on the wqo at hand, suitable finite representations have been devised in the literature [26, 27, 2]; see [18] for a rather inclusive algebra of such representations. For the basic wqos introduced earlier, this yields:

finite sets: an ideal of $(\Sigma, =)$ is a singleton $\{x\}$ for $x \in \Sigma$; it can be represented by the element x itself with $[\![x]\!]_\Sigma \stackrel{\text{def}}{=} \{x\}$ as associated ideal.

natural numbers: an ideal of $(\mathbb{N}, \leq)$ is either $\mathbb{N}$ itself or a downwards-closed set $\downarrow n$ for $n \in \mathbb{N}$. They can be represented as elements x of $\mathbb{N} \uplus \{\omega\}$ with $[\![x]\!]_\mathbb{N} \stackrel{\text{def}}{=} \downarrow x$ as associated ideal, where we let $\downarrow\omega = \mathbb{N}$.

Cartesian products: an ideal of $X \times Y$ is simply the product of an ideal from X with an ideal from Y; hence we can use pairs of representations with $[\![x, y]\!]_{X\times Y} \stackrel{\text{def}}{=} [\![x]\!]_X \times [\![y]\!]_Y$.

finite sequences: an ideal of X^* is a *product* $P \subseteq X^*$, i.e. a finite concatenation $A_1 \cdot A_2 \cdots A_n$ of *atoms* $A_j \subseteq X^*$, where the latter are either equal to $I \cup \{\varepsilon\}$ for some ideal I of X (where ε denotes the empty sequence), or to D^* for a downwards-closed subset D of X [27]. Products can therefore be represented as simple regular expressions with abstract syntax

$$p ::= a_1 \cdot a_2 \cdots a_n \;, \qquad\qquad a ::= z + \varepsilon \mid (z_1 + \cdots + z_m)^* \tag{2}$$

where $z, z_1, \ldots, z_m$ range over ideal representations for X. The associated ideal is defined through the usual semantics for regular expressions:

$$\begin{aligned} [\![z + \varepsilon]\!]_{X^*} &\stackrel{\text{def}}{=} [\![z]\!]_X \cup \{\varepsilon\} \;, \\ [\![(z_1 + \cdots + z_m)^*]\!]_{X^*} &\stackrel{\text{def}}{=} ([\![z_1]\!]_X \cup \cdots \cup [\![z_m]\!]_X)^* \;, \\ [\![a_1 \cdot a_2 \cdots a_n]\!]_{X^*} &\stackrel{\text{def}}{=} [\![a_1]\!]_{X^*} \cdot [\![a_2]\!]_{X^*} \cdots [\![a_n]\!]_{X^*} \;. \end{aligned}$$

Those representations come with algorithms to perform the typically required operations [21], e.g. to check whether $[\![z]\!]_X \subseteq [\![z']\!]_X$, or to compute the canonical ideal decomposition of $[\![z]\!]_X \cap [\![z']\!]_X$ or $X \setminus {\uparrow}x$ for any $x \in X$ and representations z, z'.

3 Configuration-Based WQO

Observe that the *cover* defined in Equation (1) is downwards-closed for $\sqsubseteq$; it follows that it can be decomposed as a finite union of ideals. In particular, covers can be finitely represented by finite sets of *extended configurations*, each of them denoting an ideal included in the coverability set. The algorithm of Karp and Miller [28] computes such a representation. We present here in more detail the reasoning leading to this result.

Ordering Configurations. The configurations of a VASS are equipped with the product ordering $\sqsubseteq$:

$$(\mathit{Confs}, \sqsubseteq) \stackrel{\text{def}}{=} (Q, =) \times (\mathbb{N}, \leq)^d \,. \tag{3}$$

Rephrased in a more explicit way, $(q, \boldsymbol{v}) \sqsubseteq (q', \boldsymbol{v}')$ if, and only if, $q = q'$ and $\boldsymbol{v}(i) \leq \boldsymbol{v}'(i)$ for every $1 \leq i \leq d$.

Representing Configuration Ideals. Notice that $(\mathit{Confs}, \sqsubseteq)$ is a wqo as a Cartesian product of wqos, and ideals have the following form where $\boldsymbol{x} \in (\mathbb{N} \cup \{\omega\})^d$:

$$[\![(q, \boldsymbol{x})]\!]_{\mathit{Confs}} = \{q\} \times \{\boldsymbol{v} \in \mathbb{N}^d \mid \boldsymbol{v} \leq \boldsymbol{x}\} \,. \tag{4}$$

Such a pair $(q, \boldsymbol{x})$ is called an *extended configuration* and is used as a representation for configuration ideals.

Extended Steps. The Karp and Miller algorithm is based on an extension of the step relation $\xrightarrow{t}$ over extended configurations, defined by $(p, \boldsymbol{x}) \xrightarrow{t} (q, \boldsymbol{y})$ if, and only if, $t = (p, \boldsymbol{a}, q)$ is a transition in T for some action $\boldsymbol{a}$, and for every $1 \leq i \leq d$:

$$\boldsymbol{y}(i) = \begin{cases} \boldsymbol{x}(i) + \boldsymbol{a}(i) & \text{if } \boldsymbol{x}(i) \in \mathbb{N} \,, \\ \omega & \text{otherwise.} \end{cases} \tag{5}$$

Coverability Tree Construction. The Karp and Miller algorithm is computing a tree as follows. Nodes are labelled by extended configurations. Initially, the tree is reduced to a root node labelled by the initial configuration.

A leaf labelled by c is said to be *covered* if there exists an ancestor labelled by c' such that $c \sqsubseteq c'$. Otherwise the node is said to be *uncovered*. A leaf labelled by c is said to be *live* if $c \xrightarrow{t} c'$ for some transition t in T and some extended configuration c'.

The tree is updated as follows. While there exists a live uncovered leaf, we pick one such leaf n. Assume that $c = (p, \boldsymbol{x})$ is the label of n. If there exists an ancestor labelled by $(p, \boldsymbol{y})$ such that $\boldsymbol{y} \leq \boldsymbol{x}$ and $\boldsymbol{y}(i) < \boldsymbol{x}(i) < \omega$ for some i, we pick such an ancestor and we add a child to n labelled by $(p, \boldsymbol{z})$ where $\boldsymbol{z}$ is defined as follows for $i \in \{1, \ldots, d\}$:

$$\boldsymbol{z}(i) \stackrel{\text{def}}{=} \begin{cases} \boldsymbol{x}(i) & \text{if } \boldsymbol{y}(i) = \boldsymbol{x}(i) \,, \\ \omega & \text{if } \boldsymbol{y}(i) < \boldsymbol{x}(i) \,. \end{cases} \tag{6}$$

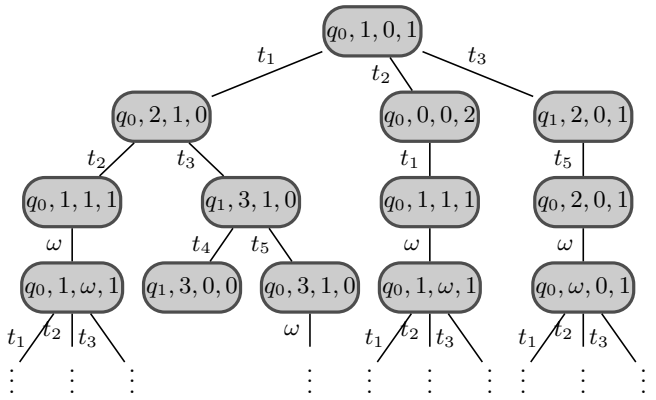


Figure 2 A prefix of the tree computed by the Karp and Miller algorithm on the VASS of Figure 1 from the initial configuration $(q_0, 1, 0, 1)$.

This operation, called *acceleration*, introduces new ω's on components which intuitively can be increased to arbitrary large values. Otherwise, if there does not exist such an ancestor, for each transition t such that $c \xrightarrow{t} c'$ for some extended configuration c', we add a child to n labelled by c'.

Ideal Decomposition using the Coverability Tree. The termination of the previous construction relies on the fact that $(\mathit{Confs}, \sqsubseteq)$ is a wqo. As shown by Karp and Miller [28],

▶ **Theorem 2.** *The Karp and Miller algorithm terminates and it produces a tree satisfying:*

$$\mathit{Cover}(c) = \bigcup_{c' \text{ label of a node}} [\![c']\!]_{\mathit{Confs}} \,.$$

In particular, by keeping only the maximal labels of the tree, we obtain the unique decomposition of the coverability set into maximal ideals.

▶ **Corollary 3.** *The canonical ideal decomposition of the coverability set is effectively computable.*

▶ **Example 4.** A prefix of the tree computed by the algorithm of Karp and Miller on the 3-dimensional VASS of Figure 1 from the initial configuration $(q_0, 1, 0, 1)$ is depicted in Figure 2. Edges of the tree that introduce ωs using (6) are labelled by 'ω.' The other ones are labelled by transitions satisfying the extended step relation. Note that the coverability set for this example is quite simple, as it is equal to:

$$[\![q_0, \omega, \omega, \omega]\!]_{\mathit{Confs}} \cup [\![q_1, \omega, \omega, \omega]\!]_{\mathit{Confs}} \,. \tag{7}$$

In other words, any configuration can be covered in this VASS. This is not so immediate however, as ωs are introduced very progressively in the coverability tree started in Figure 2.

Applications. The decomposition of the coverability set into maximal ideals provides a simple algorithm for deciding the coverability problem, since the latter reduces to finding an ideal of the decomposition that contains a given configuration. The decomposition also provides a way to decide many other problems like the place boundedness problem, that takes as input a set $K \subseteq \{1, \ldots, d\}$ and asks whether there exists a bound $m \in \mathbb{N}$ such that every configuration $(q, \boldsymbol{v})$ reachable from the initial configurations satisfies $\boldsymbol{v}(k) \leq m$ for every $k \in K$. This problem reduces to checking that the extended configurations $(q, \boldsymbol{x})$ denoting the ideals of the coverability set satisfy $\boldsymbol{x}(k) \in \mathbb{N}$ for every $k \in K$.

Notes on Complexity. From the unique decomposition of the coverability set into maximal ideals, we define the size of the coverability set as the sum of the size of the extended configurations denoting these ideals (with numbers encoded in binary). Since there exists a family of initialised VASS with finite but Ackermannian-sized reachability sets [7], the size of the cover is at least Ackermannian in the worst case. This lower bound is tight, because the Karp and Miller algorithm terminates in at most an Ackermannian number of steps [16].

The algorithm of Karp and Miller is therefore optimal for computing the ideal decomposition of the coverability set. This does not entail that it is optimal for all the problems it can help solving. For instance, on the one hand, the *place boundedness* problem mentioned earlier can be solved in exponential space [3, 11]. On the other hand, the *finite containment* problem, which asks given two VASS with finite reachability sets whether the reachability set of the first is included into that of the second, is complete for Ackermannian time [41, 42].

4 Run-Based WQO

Let us denote the *set of runs* from a source configuration c to a target configuration c' in an input VASS by $\mathit{Runs}(c, c')$. We denote by $\downarrow\mathit{Runs}(c, c')$ the downward closure of the set of runs inside a wqo $(\mathit{PreRuns}, \trianglelefteq)$ defined next in Equation (9), the VASS reachability problem can then be recast as asking whether the downward closed set $\downarrow\mathit{Runs}(c, c')$ is empty. Since this set is downwards-closed, it can be decomposed into a finite union of ideals, which is computed by the KLMST algorithm. Let us proceed again through the main steps of this result.

Ordering Runs. The set of runs can be partially ordered by introducing the weaker notion of preruns. A *prerun* is a triple $\rho = (c, w, c')$ where c and c' are two configurations and w is a word over the alphabet $\mathit{PreSteps} = \mathit{Confs} \times T \times \mathit{Confs}$. The configurations c and c' are called respectively the *source* and *target* of ρ. The set of preruns is denoted by $\mathit{PreRuns}$. Presteps and preruns are well-quasi-ordered as follows:

$$(\mathit{PreSteps}, \preceq) \stackrel{\text{def}}{=} (\mathit{Confs}, \sqsubseteq) \times (T, =) \times (\mathit{Confs}, \sqsubseteq) \tag{8}$$

$$(\mathit{PreRuns}, \trianglelefteq) \stackrel{\text{def}}{=} (\mathit{Confs}, \sqsubseteq) \times (\mathit{PreSteps}, \preceq)^* \times (\mathit{Confs}, \sqsubseteq) \tag{9}$$

A prestep $e = (c, t, c')$ is called a *step* if it satisfies the step relation $c \xrightarrow{t} c'$. A prerun (c, w, c') is called a *run* if w satisfies:

- either $w = \varepsilon$ is the empty sequence and then $c = c'$,
- or $w = (c_1, t_1, c'_1) \cdots (c_k, t_k, c'_k)$ is a sequence of steps such that $c = c_1$, $c' = c'_k$, and $c_{j+1} = c'_j$ for all $1 \leq j < k$.

▶ **Example 5.** Consider again the 3-dimensional VASS of Figure 1. It has a sequence of steps from $c = (q_0, 1, 0, 1)$ to $c' = (q_1, 2, 2, 1)$

$$(q_0, 1, 0, 1) \xrightarrow{t_1} (q_0, 2, 1, 0) \xrightarrow{t_2} (q_0, 1, 1, 1) \xrightarrow{t_1} (q_0, 2, 2, 0) \xrightarrow{t_2} (q_0, 1, 2, 1) \xrightarrow{t_3} (q_1, 2, 2, 1)\ ,$$

which we see as a run (c, w, c') in $\mathit{Runs}(c, c')$ with

$$w = ((q_0, 1, 0, 1), t_1, (q_0, 2, 1, 0))\ ((q_0, 2, 1, 0), t_2, (q_0, 1, 1, 1))\ ((q_0, 1, 1, 1), t_1, (q_0, 2, 2, 0))$$
$$((q_0, 2, 2, 0), t_2, (q_0, 1, 2, 1))\ ((q_0, 1, 2, 1), t_3, (q_1, 2, 2, 1))\ .$$

This is just one example of a run witnessing reachability; observe that any sequence of transitions in

$$\{t_1 t_2, t_2 t_1\}^{n+2} t_3 t_4^n \tag{10}$$

for $n \geq 0$ would similarly do.

Representing Prerun Ideals. Notice that ideals of $(PreSteps, \preceq)$ have the following form, where $e = (c, t, c')$ is an *extended prestep*, i.e. c, c' are extended configurations, and $t \in T$:

$$[\![e]\!]_{PreSteps} = [\![c]\!]_{Confs} \times \{t\} \times [\![c']\!]_{Confs} \ . \tag{11}$$

It follows that ideals of $(PreRuns, \trianglelefteq)$ have the following form, where p is a regular expression denoting a product over extended steps as defined in Equation (2) and c, c' are extended configurations:

$$[\![c, p, c']\!]_{PreRuns} = [\![c]\!]_{Confs} \times [\![p]\!]_{PreSteps^*} \times [\![c']\!]_{Confs} \ . \tag{12}$$

Let us instantiate (2) in this case:

$$p ::= a_1 \cdots a_n \ , \qquad\qquad a ::= e + \varepsilon \mid E^*$$

where e ranges over extended presteps and E over finite sets of extended presteps, with semantics $[\![E^*]\!]_{PreStep^*} \stackrel{\text{def}}{=} \big(\bigcup_{e \in E} [\![e]\!]_{PreSteps}\big)^*$. An observation we will use next is that such a set E can be seen as a finite directed graph with extended configurations c as vertices, connected by edges labelled by transitions t in T.

Run Ideals. In [37] we show that the maximal ideals of the decomposition of $\downarrow Runs(c, c')$ satisfy some additional properties. More precisely, thanks to the finite basis property of $(PreRuns, \trianglelefteq)$, $Runs(c, c')$ has a finite number of minimal elements B and we can write

$$\downarrow Runs(c, c') = \downarrow\Big(\bigcup_{\rho \in B} \{\rho' \in Runs(c, c') \mid \rho \trianglelefteq \rho'\}\Big) = \bigcup_{\rho \in B} \downarrow\{\rho' \in Runs(c, c') \mid \rho \trianglelefteq \rho'\} \ . \tag{13}$$

This means we can focus on ideals of the form

$$\downarrow\{\rho' \in Runs(c, c') \mid \rho \trianglelefteq \rho'\} \tag{14}$$

for some run ρ in $Runs(c, c')$. Using the fact that $(Runs(c, c'), \trianglelefteq)$ has the *amalgamation property* – i.e. if $\rho \trianglelefteq \rho_1$ and $\rho \trianglelefteq \rho_2$ for some runs $\rho_1, \rho_2 \in Runs(c, c')$, then there exists a run $\rho_3 \in Runs(c, c')$ with $\rho_1 \trianglelefteq \rho_3$ and $\rho_2 \trianglelefteq \rho_3$ – we see that the set in (14) is directed and therefore an ideal.

When considering the representation (c, p, c') for an ideal defined by (14), we then observed in [37] that it has a specific form, which is essentially a syntactic variant of the *generalised VASS* of Kosaraju [29] and *marked graph-transition sequences* of Lambert [30]: its product expression p is of the form

$$p ::= E_0^* \cdot (e_1 + \varepsilon) \cdot E_1^* \cdots (e_k + \varepsilon) \cdot E_k^* \ , \tag{15}$$

i.e. it intersperses extended presteps e_j and finite sets of extended presteps E_j. Additionally,

- all these extended presteps $e = (c_1, t, c_2)$ satisfy the extended step relation $c_1 \xrightarrow{t} c_2$, and
- the graphs defined by the sets E_j are strongly connected.

Finally, the representation of an ideal like (14) satisfies furthermore a (decidable) *adherence* condition corresponding to the θ *condition* introduced by Kosaraju [29], or the *perfectness condition* introduced by Lambert [30]. The point here is that we have now a semantics, in terms of ideals, associated with these syntactic representations and conditions.

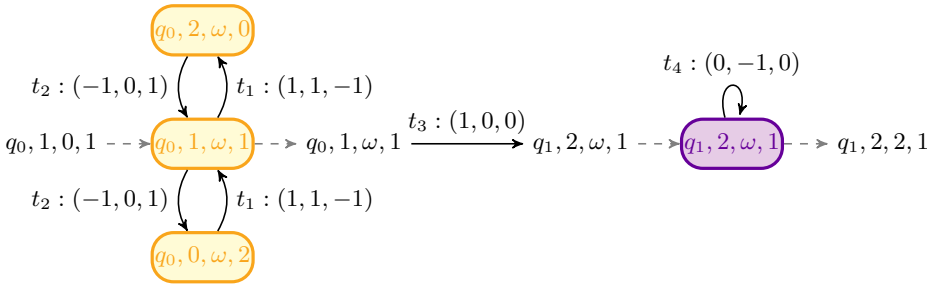


Figure 3 The unique maximal ideal of $\downarrow Runs(c, c')$ for the VASS of Figure 1.

Ideal Decomposition using the KLMST Algorithm. Entering the details of the KLMST algorithm would be too long for the purposes of this presentation. We refer to the nice expositions of Müller [43] and Reutenauer [45] for details and examples. The main point here is that the unique decomposition of $\downarrow Runs(c, c')$ into maximal ideals is precisely what the KLMST algorithm is computing.

▶ **Theorem 6** (Decomposition Theorem [37]). *The KLMST algorithm computes an ideal decomposition of* $\downarrow Runs(c, c')$.

Again, by keeping only the maximal ideals in this decomposition, we obtain the unique decomposition of $\downarrow Runs(c, c')$ into maximal ideals.

▶ **Corollary 7.** *The canonical ideal decomposition of* $\downarrow Runs(c, c')$ *is effectively computable.*

▶ **Example 8.** Let us come back to the 3-dimensional VASS of Figure 1 and let $c = (q_0, 1, 0, 1)$ and $c' = (q_1, 2, 2, 1)$. The ideal decomposition of $\downarrow Runs(c, c')$ contains a unique ideal depicted in Figure 3 (see [48] for more details on how this ideal is computed by the KLMST algorithm). This ideal has the following form:

$$[\![c, E_0^* \cdot (e_1 + \varepsilon) \cdot E_1^*, c']\!]_{PreRuns} \tag{16}$$

where E_0 contains four edges denoting the yellow strongly connected graph, E_1 contains one edge denoting the violet strongly connected graph, and $e_1 = ((q_0, 1, \omega, 1), t_3, (q_1, 2, \omega, 1))$ links these two graphs. This matches the set of runs found earlier in Equation (10); one should contrast this with the very simple ideal decomposition of *Cover* found in (7).

Applications. Theorem 6 entails the decidability of the reachability problem: $Runs(c, c')$ is empty if and only if its downward-closure is. It also allows to prove the completeness of acceleration techniques for computing Presburger definable reachability sets [35].

In the case of *labelled VASS* where we additionally label transitions in T by finite sequences over a finite alphabet Σ, this also provides a way of constructing the downward-closure of the language between c and c', which was already shown to be computable by Habermehl et al. [22] and Zetsche [50].

Notes on Complexity. The decomposition of $Runs(c, c')$ into maximal ideals provides a way to associate to this set a size (with numbers encoded in binary). From a complexity point of view, the already mentioned construction of Cardoza et al. [7] shows that, in the worst case, the size of $Runs(c, c')$ can be Ackermannian, i.e. is in $F_\omega(\Omega(n))$ for an input of size n (using the fast-growing functions $(F_\alpha)_\alpha$ of Löb and Wainer [39]). We exhibit in

[37] the first upper bound for that size by proving a worst case complexity in $F_{\omega^3}(p(n))$ for an Ackermannian function p. This gap between $F_\omega(\Omega(n))$ and $F_{\omega^3}(p(n))$ seems difficult to tighten, and the exact complexity is still open. Again, the Ackermannian lower bound on the size of the ideal decomposition does not entail such a gigantic lower bound on the problems it helps solving; in particular, the reachability problem could very well be much simpler, as the best known lower bound is in exponential space [7].

5 Conclusion

As we have seen in this short presentation, wqo ideals provide abstract foundations for both the coverability tree construction of Karp and Miller [28] and the KLMST algorithm of Mayr [40], Kosaraju [29], and Lambert [30]. On both accounts, these algorithms compute the canonical ideal decomposition of a downwards-closed set, namely the cover for the coverability tree and the downward-closure of the set of runs for the KLMST algorithm.

This abstract viewpoint on those two algorithms makes them easier to extend to more general classes of systems. In fact, the coverability tree construction has already been extended to unordered data Petri nets [24], branching VASS [49, 25], and pushdown VASS [36]. In each of these cases however, the decidability of the reachability problem is currently open.

Those are not the only algorithmic applications of wqo ideals. For instance, Lazić and Schmitz [33] revisit the usual *backward coverability* algorithm for WSTS [1, 19] using ideals, and employ it to derive in a uniform manner several known complexity upper bounds on the coverability problem, which were initially based on an approach due to Rackoff [44]: for VASS [6], alternating VASS [9], and branching VASS [13, 34]. Another example is the use of ideal decompositions of formal languages by Zetsche [50], employed for instance by Czerwiński et al. [10] to prove the decidability of separation by piecewise testable languages.

References

1 P. A. Abdulla, K. Čerāns, B. Jonsson, and Yih-Kuen Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Inform. and Comput.*, 160(1–2):109–127, 2000.

2 Parosh Aziz Abdulla, Aurore Collomb-Annichini, Ahmed Bouajjani, and Bengt Jonsson. Using forward reachability analysis for verification of lossy channel systems. *Form. Methods in Syst. Des.*, 25(1):39–65, 2004. doi:10.1023/B:FORM.0000033962.51898.1a.

3 Michel Blockelet and Sylvain Schmitz. Model-checking coverability graphs of vector addition systems. In *MFCS 2011*, volume 6907 of *LNCS*, pages 108–119. Springer, 2011. doi:10.1007/978-3-642-22993-0_13.

4 Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Logic*, 12(4):1–26, 2011. doi:10.1145/1970398.1970403.

5 Robert Bonnet. On the cardinality of the set of initial intervals of a partially ordered set. In *Infinite and finite sets: to Paul Erdős on his 60th birthday, Vol. 1*, Coll. Math. Soc. János Bolyai, pages 189–198. North-Holland, 1975.

6 Laura Bozzelli and Pierre Ganty. Complexity analysis of the backward coverability algorithm for VASS. In Giorgio Delzanno and Igor Potapov, editors, *Proc. RP 2011*, volume 6945 of *LNCS*, pages 96–109. Springer, 2011. doi:10.1007/978-3-642-24288-5_10.

7 E. Cardoza, Richard J. Lipton, and Albert R. Meyer. Exponential space complete problems for Petri nets and commutative semigroups: Preliminary report. In *Proc. STOC'76*, pages 50–54. ACM, 1976. doi:10.1145/800113.803630.

8 Thomas Colcombet and Amaldev Manuel. Generalized data automata and fixpoint logic. In *Proc. FSTTCS 2014*, volume 29 of *LIPIcs*, pages 267–278. LZI, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.267.

9 Jean-Baptiste Courtois and Sylvain Schmitz. Alternating vector addition systems with states. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Proc. MFCS 2014*, volume 8634 of *LNCS*, pages 220–231. Springer, 2014. doi:10.1007/978-3-662-44522-8_19.

10 Wojciech Czerwiński, Wim Martens, Lorijn van Rooijen, Marc Zeitoun, and Georg Zetzsche. A characterization for decidable separability by piecewise testable languages. Preprint, 2015. Extended abstract published in *Proc. FCT 2015*. URL: http://arxiv.org/abs/1410.1042.

11 Stéphane Demri. On selective unboundedness of VASS. *Journal of Computer and System Sciences*, 79(5):689–713, 2013. doi:10.1016/j.jcss.2013.01.014.

12 Stéphane Demri, Diego Figueira, and M. Praveen. Reasoning about data repetitions with counter systems. In *Proc. LICS 2013*, pages 33–42. IEEE Press, 2013. doi:10.1109/LICS.2013.8.

13 StéPhane Demri, Marcin Jurdziński, Oded Lachish, and Ranko Lazić. The covering and boundedness problems for branching vector addition systems. *Journal of Computer and System Sciences*, 79(1):23–38, 2013.

14 Leonard Eugene Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *Amer. J. of Math.*, 35(4):413–422, 1913. doi:10.2307/2370405.

15 Catherine Dufourd, Philippe Schnoebelen, and Petr Jančar. Boundedness of reset P/T nets. In *Proc. ICALP'99*, volume 1644 of *LNCS*, pages 301–310, 1999. doi:10.1007/3-540-48523-6_27.

16 Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson's Lemma. In *Proc. LICS 2011*, pages 269–278. IEEE Press, 2011. doi:10.1109/LICS.2011.39.

17 Alain Finkel and Jean Goubault-Larrecq. Forward analysis for WSTS, part I: Completions. In *Proc. STACS 2009*, volume 3 of *LIPIcs*, pages 433–444. LZI, 2009. doi:10.4230/LIPIcs.STACS.2009.1844.

18 Alain Finkel and Jean Goubault-Larrecq. Forward analysis for WSTS, part II: Complete WSTS. *Logic. Meth. in Comput. Sci.*, 8(3:28):1–35, 2012. doi:10.2168/LMCS-8(3:28)2012.

19 Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1–2):63–92, 2001. doi:10.1016/S0304-3975(00)00102-X.

20 Pierre Ganty and Rupak Majumdar. Algorithmic verification of asynchronous programs. *ACM Trans. Prog. Lang. Sys.*, 34(1):1–48, 2012. doi:10.1145/2160910.2160915.

21 Jean Goubault-Larrecq, Prateek Karandikar, K. Narayan Kumar, and Philippe Schnoebelen. The ideal approach to computing closed subsets in well-quasi-orderings. In preparation, 2016.

22 Peter Habermehl, Roland Meyer, and Harro Wimmel. The downward-closure of Petri net languages. In *Proc. ICALP 2010*, volume 6199 of *LNCS*, pages 466–477. Springer, 2010. doi:10.1007/978-3-642-14162-1_39.

23 Graham Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc.*, 3(2):326–336, 1952. doi:10.1112/plms/s3-2.1.326.

24 Piotr Hofman, Sławomir Lasota, Ranko Lazić, Jérôme Leroux, Sylvain Schmitz, and Patrick Totzke. Coverability trees for Petri nets with unordered data. In *Proc. FoSSaCS 2016*, LNCS. Springer, 2016. To appear.

25 Paulin Jacobé de Naurois. Coverability in a non functional extension of BVASS. Preprint, 2014. Presented at CiE 2013. URL: https://hal.archives-ouvertes.fr/hal-00947136.

26 Pierre Jullien. *Contribution à l'étude des types d'ordres dispersés*. Thèse de doctorat, Université de Marseille, 1969.

27 M. Kabil and M. Pouzet. Une extension d'un théorème de P. Jullien sur les âges de mots. *Theor. Inform. Appl.*, 26(5):449–482, 1992.

28 Richard M. Karp and Raymond E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969. doi:10.1016/S0022-0000(69)80011-5.

29 S. Rao Kosaraju. Decidability of reachability in vector addition systems. In *Proc. STOC'82*, pages 267–281. ACM, 1982. doi:10.1145/800070.802201.

30 Jean-Luc Lambert. A structure to decide reachability in Petri nets. *Theor. Comput. Sci.*, 99(1):79–104, 1992. doi:10.1016/0304-3975(92)90173-D.

31 Ranko Lazić. The reachability problem for vector addition systems with a stack is not elementary. Preprint, 2013. Presented at RP 2012. URL: http://arxiv.org/abs/1310.1767.

32 Ranko Lazić, Tom Newcomb, Joël Ouaknine, A.W. Roscoe, and James Worrell. Nets with tokens which carry data. *Fund. Inform.*, 88(3):251–274, 2008.

33 Ranko Lazić and Sylvain Schmitz. The ideal view on Rackoff's coverability technique. In *Proc. RP 2015*, volume 9328 of *LNCS*, pages 1–13. Springer, 2015. doi:10.1007/978-3-319-24537-9_8.

34 Ranko Lazić and Sylvain Schmitz. Non-elementary complexities for branching VASS, MELL, and extensions. *ACM Trans. Comput. Logic*, 16(3), 2015. doi:10.1145/2733375.

35 Jérôme Leroux. Presburger vector addition systems. In *Proc. LICS 2013*, pages 23–32. IEEE Press, 2013. doi:10.1109/LICS.2013.7.

36 Jérôme Leroux, M. Praveen, and Grégoire Sutre. Hyper-Ackermannian bounds for pushdown vector addition systems. In *Proc. CSL-LICS 2014*. ACM, 2014. doi:10.1145/2603088.2603146.

37 Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *Proc. LICS 2015*, pages 56–67. IEEE Press, 2015. doi:10.1109/LICS.2015.16.

38 Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. On the coverability problem for pushdown vector addition systems in one dimension. In *Proc. ICALP 2015*, volume 9135 of *LNCS*, pages 324–336. Springer, 2015.

39 M.H. Löb and S.S. Wainer. Hierarchies of number-theoretic functions. I. *Arch. Math. Logic*, 13(1–2):39–51, 1970. doi:10.1007/BF01967649.

40 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *Proc. STOC'81*, pages 238–246. ACM, 1981. doi:10.1145/800076.802477.

41 E.W. Mayr and A.R. Meyer. The complexity of the finite containment problem for Petri nets. *Journal of the ACM*, 28(3):561–576, 1981. doi:10.1145/322261.322271.

42 Kenneth McAloon. Petri nets and large finite sets. *Theor. Comput. Sci.*, 32(1–2):173–183, 1984.

43 Horst Müller. The reachability problem for VAS. In *Advances in Petri Nets 1984*, volume 188 of *LNCS*, pages 376–391. Springer, 1985. doi:10.1007/3-540-15204-0_21.

44 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6(2):223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

45 Christophe Reutenauer. *The mathematics of Petri nets*. Masson and Prentice, 1990.

46 George S. Sacerdote and Richard L. Tenney. The decidability of the reachability problem for vector addition systems. In *Proc. STOC'77*, pages 61–76. ACM, 1977. doi:10.1145/800105.803396.

47 Sylvain Schmitz. On the computational complexity of dominance links in grammatical formalisms. In *Proc. ACL 2010*, pages 514–524. ACL Press, 2010.

48 Sylvain Schmitz. Automata column: The complexity of reachability in vector addition systems. *ACM SigLog News*, 3(1), 2016. To appear.

49 Kumar Neeraj Verma and Jean Goubault-Larrecq. Karp-Miller trees for a branching extension of VASS. *Disc. Math. and Theor. Comput. Sci.*, 7(1):217–230, 2005. URL: `http://www.dmtcs.org/volumes/abstracts/dm070113.abs.html`.

50 Georg Zetzsche. An approach to computing downward closures. In *Proc. ICALP 2015*, volume 9135 of *LNCS*, pages 440–451. Springer, 2015. `doi:10.1007/978-3-662-47666-6_35`.

Complexity and Expressive Power of Ontology-Mediated Queries*

Carsten Lutz

Fachbereich Informatik
Universität Bremen, Germany
clu@uni-bremen.de

Abstract

Data sets that have been collected from multiple sources or extracted from the web or often highly incomplete and heterogeneous, which makes them hard to process and query. One way to address this challenge is to use ontologies, which provide a way to assign a semantics to the data, to enrich it with domain knowledge, and to provide an enriched and uniform vocabulary for querying. The combination of a traditional database query with an ontology is called an ontology-mediated query (OMQ). The aim of this talk is to survey fundamental properties of OMQs such as their complexity, expressive power, descriptive strength, and rewritability into traditional query languages such as SQL and Datalog. A central observation is that there is a close and fruitful connection between OMQs and constraint satisfaction problems (CSPs) as well as related fragments of monadic NP, which puts OMQs into a more general perspective and gives raise to a number of interesting results.

1998 ACM Subject Classification H.2.3 Database Management, Query Languages

Keywords and phrases Ontology-Mediated Queries, Description Logic, Constraint Satisfaction

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.2

Category Invited Talk

1 Description Logic and Ontologies

Description logics (DLs) are a widely known family of knowledge representation formalisms that originated in the 1980s and are now popular as ontology languages; a slightly outdated though still useful overview is given in the DL handbook [6]. From a logic perspective, DLs are best viewed as decidable fragments of first-order logic (FO), some of which are variants of modal logic while others are more closely related to Datalog-like rule languages. DLs come with their own syntax, which involves using logical quantifier symbols in a non-standard way (actually in the same way in which diamonds and boxes are used in modal logic). As a basic example, we introduce the description logic $\mathcal{ALC}$ originating from [37].[1] Its syntax is defined in two steps. In the first step, one inductively defines a set of logical formulas called *concepts* and in the second step, concepts are put into relation with each other in a logical theory called a *TBox*. We fix two sets of symbols beforehand. A set of *concept names* which are denoted with A and B and correspond to monadic predicates in FO; and a set of *role names* which are denoted with r and correspond to dyadic predicates. Predicates of

* This work was partially supported by the ERC Consolidator Grant 647289 (CODA). The material surveyed in this text is mostly taken from [9], which is joint work with Meghyn Bienvenu, Balder ten Cate, and Frank Wolter.

[1] $\mathcal{ALC}$ stands for *Attributive concept Language with Complementation*, a largely historic name.

© Carsten Lutz;
licensed under Creative Commons License CC-BY

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 2; pp. 2:1–2:11

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

higher (or lower) arity are not included in most DLs. The following table lists the concept constructors of $\mathcal{ALC}$ along with their FO translation:[2]

$$\begin{array}{rclcrclcrcl} A & \equiv & A(x) & & C \sqcap D & \equiv & C(x) \wedge D(x) & & \exists r.C & \equiv & \exists y\,(r(x,y) \wedge C(y)) \\ \neg C & \equiv & \neg C(x) & & C \sqcup D & \equiv & C(x) \vee D(x) & & \forall r.C & \equiv & \forall y\,(r(x,y) \rightarrow C(y)) \end{array}$$

where C and D range over (potentially compound) concepts. Note that every concept translates into an equivalent FO formula with one free variable. As in modal logic (and unlike in FO), monadic and dyadic predicates are used in a non-interchangeable way in the syntax. In a sense, DLs have a strong focus on monadic predicates and on formulas with one free variable while dyadic predicates only help to define such formulas. In fact, the connection to CSPs discussed in Section 4 rests on that setup in an essential way. The focus on monadic predicates came from application demands, where it is most important to describe *classes* of objects with similar properties. As an example, let us assume that we want to capture knowledge about the domain of traveling. We can use an $\mathcal{ALC}$ concept to describe the class of hotels in Orléans that got a high rating or are close to the STACS2016 venue:

$$\mathsf{Hotel} \sqcap \exists \mathsf{location.Orleans} \sqcap (\exists \mathsf{ranking.High} \sqcup \exists \mathsf{closeTo.UniversityCampus}).$$

While concepts are used to describe classes such as the one above, TBoxes interrelate these classes and in this way formalize the knowledge of an application domain and give a semantics to the predicate symbols used. In $\mathcal{ALC}$, a TBox is simply a finite set of *concept inclusions* $C \sqsubseteq D$, where both C and D are $\mathcal{ALC}$ concepts. While concepts translate into FO formulas with one free variable, a concept inclusion $C \sqsubseteq D$ translates into the FO sentence $\forall x\,(C(x) \rightarrow D(x))$ and a TBox translates into the conjunction of (the FO translations) of the concept inclusions contained in it. For example, the following $\mathcal{ALC}$ TBox describes some basic knowledge about the traveling domain:

$$\begin{array}{rl} \mathsf{Airlline} & \sqsubseteq \mathsf{BudgetAirline} \sqcup \mathsf{RegularAirline} \\ \mathsf{BudgetAirline} & \sqsubseteq \neg \mathsf{RegularAirline} \\ \mathsf{AirTicket} & \sqsubseteq \exists \mathsf{issuedBy.Airline} \\ \mathsf{AirTicket} \sqcap \exists \mathsf{issuedBy.BudgetAirline} & \sqsubseteq \neg \exists \mathsf{class.Business} \end{array}$$

The above TBox is typical for a DL TBox in that it mainly concentrates on describing properties of classes important for the application domain such as Airline and AirTicket. This is again the focus on monadic predicates mentioned above.

The name *TBox* stems from the area of knowledge representation and DLs where, historically, it stands for "terminological box". In more modern terms, TBoxes are often called *ontologies* [25]. Ontologies have applications in data access as described in more detail in Section 2 and are also used to produce standardized vocabularies. The latter is particularly important in genetics and biology where the terminology is very extensive and in medicine where there is a need to establish a standardized vocabuary for data exchange and accounting. Several hundred ontologies from the biomedical domain have been collected in the BioPortal maintained by the National Centre for Biomedical Ontology, see `http://www.bioontology.org/`. In these areas, ontologies often aim to be of general purpose and can contain up to several hundred thousand classes, as in the case of the SNOMED CT healthcare ontology which is developed and maintained by the International Health Terminology Standards Development Organisation [38]. In contrast, ontologies used

[2] The translation is actually into the two-variable guarded fragment of FO.

for data access are often custom tailored towards a particular application and tend to be smaller in size. Ontologies also play an important rôle in the semantic web, which has led to their standardization as the OWL family of web ontology languages by the World Wide Web committee (W3C). The first version of the OWL recommendation has been released in 2004, followed by the OWL 2 recommendation in 2012 [34]. OWL 2 is a collection of five languages, four of which are DLs. It comes with a variety of more "web friendly" syntaxes based, e.g., on XML and RDF. A large collection of tools for OWL ontologies is available including ontology editors, APIs, logical reasoners, and so on.

From a theoretical perspective, bisimulation is an important tool to understand the expressive power of the description logic $\mathcal{ALC}$. In fact, $\mathcal{ALC}$ concepts are a notational variant of formulas in the modal logic K: simply replace "$\sqcap$" with "$\wedge$", "$\sqcup$" with "$\vee$", "$\exists r.$" with $\Diamond_r$, and "$\forall r.$" with $\Box_r$, and read concept names as propositional letters. $\mathcal{ALC}$ concepts thus inherit the well-known relation between K and bisimulation, manifested e.g. in van Benthem's theorem [22]. TBoxes add a "global" flavour in the sense that the FO translation of concept inclusions universally quantifies over all elements of the universe. This gives raise to the following variation of van Benthem's theorem, which precisely characterizes the expressive power of $\mathcal{ALC}$ TBoxes within FO.

We refer to FO without function symbols and constants and with only monadic and dyadic predicates, which we identify with concept and role names, respectively. We say that a relational structure $\mathfrak{A}_1$ is *globally bisimilar* to a relational structure $\mathcal{A}_2$ if for every d_1 in the universe of $\mathfrak{A}_1$, there is a d_2 in the universe of $\mathfrak{A}_2$ such that there is a bisimulation between $\mathfrak{A}_1$ and $\mathfrak{A}_2$ that related d_1 with d_2, and vice versa. Further, we say that an FO sentence φ is *invariant under global bisimulation* if for all relational structures $\mathfrak{A}_1, \mathfrak{A}_2$ such that $\mathfrak{A}_1 \models \varphi$ and $\mathfrak{A}_1$ is globally bisimilar to $\mathfrak{A}_2$, we have $\mathfrak{A}_2 \models \varphi$.

▶ **Theorem 1** ([29]). *An FO sentence φ is equivalent to an $\mathcal{ALC}$ TBox iff φ is preserved under global bisimulation and under disjoint union.*

The description logic $\mathcal{ALC}$ presented here is a bit too simple and inexpressive to be useful in many applications. In fact, the languages of the OWL 2 family include a wealth of additional expressive means, selected such that satisfiability (and several other relevant reasoning problems) are still decidable, but the expressive power is more satisfactory. In the literature, a large number of decription logics have been considered that balance expressive power and computational complexity in different ways. Although many of them do not admit characterizations as clean as Theorem 1, there are often intimiate relationships with suitably modified notions of bisimulation [29]. It should also be mentioned that several widely-used DLs such as $\mathcal{EL}$, DL-Lite, and their extensions do not include negation, disjunction, and universal quantification [5, 17]. Rather than to modal logic, these languages are more closely related to Datalog and its extension with existential quantification in the rule heads, known as tuple-generating dependencies, existential rules, and Datalog$^{\pm}$ [14, 33].

2 Ontology-Mediated Queries

Using DL ontologies for data access is a very active branch of research, see for example [8, 16, 26] for some recent surveys. In DLs, data is commonly stored in a so-called *ABox* (another historical name, standing for "assertional Box"), which is simply a finite set of ground facts of the form $A(a)$ and $r(a, b)$ called *assertions*, where a, b are FO constants. Note that also ABoxes use only unary and dyadic predicates. This is appropriate for example for web data represented in RDF [35]. It is less appropriate for data that stems from traditional

database systems, whose schemas often use relations of high arity; schema mappings have been proposed as a workaround, see [16]. The data stored in an ABox is considered (potentially) incomplete, that is, additional assertions except those explicitly stated in the ABox might be true. This is reflected in the semantics of query answering. Before giving details, we first introduce some relevant query languages.

A *conjunctive query (CQ)* takes the form $q = \exists \mathbf{x}\, \varphi(\mathbf{x}, \mathbf{y})$ with $\mathbf{x}, \mathbf{y}$ tuples of variables and φ a conjunction of atoms of the form $A(x)$ and $r(x, y)$ that uses only variables from $\mathbf{x} \cup \mathbf{y}$. The variables in $\mathbf{x}$ are called *answer variables*, the *arity* of q is the length of $\mathbf{x}$, and q is *Boolean* if it has arity zero. A *union of conjunctive queries (UCQ)* takes the form $q_1 \vee \cdots \vee q_k$ where $q_1, \ldots, q_k$ are CQs with the same answer variables. An *atomic query (AQ)* is a conjunctive query of the form $A(x)$ and a *Boolean atomic query (BAQ)* is a conjunctive query of the form $\exists x\, A(x)$.

As an example, consider the ABox

$$\mathsf{AirTicket}(\mathsf{offer12}) \quad \mathsf{class}(\mathsf{offer12}, \mathsf{j6}) \quad \mathsf{BusinessClass}(\mathsf{j6}),$$

the TBox given above, and the query $q = \exists y\, \mathsf{Ticket}(x) \wedge \mathsf{issuedBy}(x, y) \wedge \mathsf{RegularAirline}(y)$ asking to return all tickets issued by a regular airline. The domain knowledge in the TBox adds additional facts to the data such as that the ticket named offer12 is issued by some airline, and that this airline cannot be a budget airline, thus must be a regular airline, which allows to return offer12 as an answer to the query.

A (finite or infinite) set of assertions can be viewed as a relational structure in an obvious way. A relational structure $\mathfrak{A}$ is a *model* of an ABox $\mathcal{A}$ if it can be obtained from $\mathcal{A}$ by extending it with additional assertions, possibly involving additional constants. $\mathfrak{A}$ is a model of a TBox $\mathcal{T}$ is it satisfies (the FO translations of) all concept inclusions in $\mathcal{T}$. A tuple of constants $\mathbf{a}$ is an *answer* to a query q in $\mathfrak{A}$ if $\mathfrak{A} \models q[\mathbf{a}]$ in the standard sense of FO logic. Moreover, $\mathbf{a}$ is a *certain answer* to q in an ABox $\mathcal{A}$ given a TBox $\mathcal{T}$ if $\mathbf{a}$ is an answer to q in all models of $\mathcal{A}$ and $\mathcal{T}$. In the above example, offer12 is a certain answer.

An *ontology mediated query (OMQ)* is a triple $(\mathcal{T}, \Sigma, q)$ where $\mathcal{T}$ is a TBox, Σ a data signature (that is, a set of concept and role names that can occur in the ABox), and q an actual query such as a CQ or an AQ. Note that $\mathcal{T}$ might introduce symbols that do not occur in the data, in this way enriching the vocabulary available for formulating the query q. An OMQ $(\mathcal{T}, \Sigma, q)$ is Boolean if q is. We use $(\mathcal{L}, \mathcal{Q})$ to denote the OMQ language which consists of all OMQs $(\mathcal{T}, \Sigma, q)$ with $\mathcal{T}$ formulated in the DL $\mathcal{L}$ and q formulated in the query language $\mathcal{Q}$. The most common choices for $\mathcal{Q}$ are AQs, CQs, and UCQs, giving raise for example to the OMQ languages $(\mathcal{ALC}, \mathrm{AQ})$ and $(\mathcal{ALC}, \mathrm{CQ})$. BAQs are somewhat less common, but, as we will see, constitute very natural cases when establishing the connection between OMQs and CSPs.

Given the exposition above, it is natural to ask in which way adding an ontology to a database query affects the complexity of query answering, and how the expressive power of OMQs relates to the expressive power of more standard database query languages. For simplicity, we will mostly concentrate on Boolean OMQs. As in the case of conventional databases, there are several complexity measures that one might study. In particular, combined complexity considers both the ABox and the OMQ as an input while data complexity assumes the OMQ to be fixed and considers only the ABox to be the input. Since OMQs tend to be small compared to the actual data,[3] data complexity is often viewed as the

[3] Very large ontologies such as SNOMED CT clearly constitute an exception and suggest other complexity measures such as treating both the ABox and the TBox as an input, but not the actual query.

more natural measure. It is not hard to see that there are OMQs that are coNP-complete in data complexity. For example, the following OMQ encodes non-3-colorability:

$$\mathcal{T} = \{\top \sqsubseteq R \sqcup G \sqcup B, \quad R \sqcup \exists r.R \sqsubseteq D, \quad G \sqcup \exists r.G \sqsubseteq D, \quad B \sqcup \exists r.B \sqsubseteq D\}$$
$$\Sigma = \{r\}$$
$$q = \exists x\, D(x).$$

Here, $\top$ is an abbreviation for a tautology such as $A \sqcup \neg A$ and the concept name D signals a defect. Recall that, without ontologies, the data complexity of FO queries such as CQs and UCQs is extremely low, namely in AC_0. By adding ontologies, we have thus transitioned from highly efficient to intractable. This seems unacceptable, but rarely causes unsolvable problems in practice because real-world ontologies do not encode combinatorial problems such as 3-colorability. But how to separate the tractable cases from the intractable ones in a theoretically clean way? A brute-force way is to replace $\mathcal{ALC}$ with a very restricted ontology language such as the DLs $\mathcal{EL}$ and DL-Lite mentioned above, resulting in PTime-completeness in data complexity. While this is acceptable for some applications, it is unacceptable for others where negation and disjunction are needed for proper modeling, though typically in a harmless way. It is tempting but seems impossible to charaterize what "harmless" means in a syntactic way. To achieve maximum flexibility and to circumvent syntactic characterizations, a non-uniform approach was advocated in [30] whose aim is to classify the exact complexity of *every* OMQ within a given OMQ language.

3 Constraint Satisfaction Problems and MMSNP

There is an interesting connection between OMQs and constraint satisfaction problems (CSP) which puts OMQs into a more general perspective and allows to obtain a number of interesting results regarding their expressive power and (non-uniform) complexity. This connection also extends in a very natural way to the logical generalization MMSNP of CSPs introduced in a seminal paper of Feder and Vardi [19], and it provides new motivation for studying this logic. In this section, we briefly introduce CSPs and MMSNP.

There are various equivalent definitions of CSPs [36]. We choose here to define them in terms of homomorphisms between relational structures. A *template* is a finite relational structure in some signature Σ; in contrast to the previous sections, the arity of predicates is unrestricted. Each template T defines the class of finite relational Σ-structures $\mathrm{CSP}(T) = \{S \mid S \to T\}$ where $S \to T$ means that there is a homomorphism from S to T. The constraint satisfaction problem for template T is to decide, given a finite relational Σ-structure S, whether $S \in \mathrm{CSP}(T)$. It is easy to see that every CSP is in NP and that there are CSPs that are NP-complete. An example is $\mathrm{CSP}(K_3)$ where K_3 is the 3-clique and Σ contains only a single dyadic predicate r which represents edges in graphs; note that an undirected graph S is in $\mathrm{CSP}(K_3)$ if and only if S is 3-colorable. Additional NP-complete problems that can be presented as CSPs include 3-satisfiability and integer programming on finite domains. Other NP-complete problems such as Hamilton cycle cannot be presented as CSPs because the class of their "yes"-instances is not closed under homomorphic pre-images. Of course, there are also CSPs of lower complexity such as $\mathrm{CSP}(K_2)$, which is 2-colorability and thus PTime-complete.

Feder and Vardi have asked for a complete classification of the complexity of all CSPs, in particular for a precise delineation of the CSPs that can be solved in PTime from those that are NP-complete [19]. They have conjectured that a transparent such delineation is possible and that there is a dichotomy between PTime and NP for CSPs, that is, that every CSP is

in PTime or NP-hard, unlike the NP-intermediate problems whose existence is established by Ladner's theorem [27]. While the general case is still open, a lot of progress has been made and the dichotomy conjecture has been confirmed for special cases such as undirected graphs [24] and for oriented cycles [18]. To connect the dichotomy question with the field of descriptive complexity, Feder and Vardi identified a fragment of monadic NP called MMSNP (for "monotone monadic strict NP") that corresponds rather closely to CSPs. A sentence of MMSNP takes the form

$$\exists X_1 \cdots \exists X_n \forall x_1 \cdots \forall x_m \, \varphi$$

where φ is a quantifier- and equality-free FO formula in which every atom $R(\mathbf{x})$ with $R \notin \{X_1, \ldots, X_n\}$ occurs only with negative polarity. For example, $\mathrm{CSP}(K_3)$ is equivalent to the MMSNP formula

$$\begin{aligned} \exists R \, \exists G \, \exists B \, \forall x_1 \forall x_2 \; & (R(x_1) \vee G(x_1) \vee B(x_1)) \wedge \\ & \neg(R(x_1) \wedge r(x_1, x_2) \wedge R(x_2)) \wedge \\ & \neg(G(x_1) \wedge r(x_1, x_2) \wedge G(x_2)) \wedge \\ & \neg(B(x_1) \wedge r(x_1, x_2) \wedge B(x_2)) \end{aligned}$$

where r is again the dyadic edge relation in input graphs. While MMSNP is more expressive than CSP (with non-monochromatic triangle[4] being a witnessing problem), a main result of Feder and Vardi shows that there is a dichotomy between PTime and NP for CSPs if and only if there is such a dichotomy for MMSNP. Thus, MMSNP can be viewed as a well-behaved extension of CSPs. There are several seemingly minor generalizations of MMSNP which destroy this property and result in non-dichotomy.

4 Ontologies, CSP, and MMSNP

The examples given above might already suggest to the reader that there is a connection between OMQs and CSPs. In fact, CSPs can be viewed as generalized coloring problems and it is not hard to adapt the OMQ from Section 2 expressing non-3-colorability to any CSP over a signature with only unary and dyadic predicates. Let T be a template over such a signature Σ. The OMQ $Q_T = (\mathcal{T}, \Sigma, q)$ is defined by setting $q = \exists x \, D(x)$ and including the following concept inclusions in $\mathcal{T}$:

- $\top \sqsubseteq A_{d_1} \sqcup \cdots \sqcup A_{d_k}$ when the universe of T is $U = \{d_1, \ldots, d_k\}$;
- $A_d \sqcap B \sqsubseteq D$ when $d \notin B^T$ for all $d \in U$ and concept names $A \in \Sigma$;
- $A_{d_1} \sqcap \exists r.A_{d_2}$ when $(d_1, d_2) \notin r^T$ for all $d_1, d_2 \in U$ and role names $r \in \Sigma$.

Note that Q_T is formulated in the OMQ language $(\mathcal{ALC}, BAQ)$. It is equivalent to the complement of T in the sense that for any finite Σ-structure (equivalently: Σ-ABox) $\mathcal{A}$, we have $\mathcal{A} \not\to T$ if and only if Q_T is true on $\mathcal{A}$. Conversely, it is possible to convert any OMQ Q from $(\mathcal{ALC}, \mathrm{BAQ})$ into a CSP whose complement is equivalent to Q. As the template, one uses a structure that is similar to the structures emerging from filtration and type elimination constructions for modal and description logics [10]. In particular, 1-types are used as elements of the template, and this is sufficient only because of the essentially monadic nature of DLs.

We use coCSP to denote the class of complements of CSPs and likewise for coMMSNP. The next result yields a strong connection between OMQs and the CSP world.

[4] The class of all undirected graphs whose nodes can be colored black and white such that neither the all-white triangle nor the all-black triangle admits a homormophism into the colored graph.

▶ **Theorem 2** ([9]). *The following have the same expressive power:*
1. $(\mathcal{ALC}, \mathit{BAQ})$ *and coCSP;*
2. $(\mathcal{ALC}, \mathit{UCQ})$ *and coMMSNP.*

Theorem 2 can be seen as clarifying the descriptive complexity of the fundamental OMQ languages $(\mathcal{ALC}, \text{BAQ})$ and $(\mathcal{ALC}, \text{UCQ})$. It also has immediate consequences for non-uniform data complexity: classifying the complexity of all OMQs from these OMQ languages is equivalent to classifying the complexity of CSPs with only unary and dyadic predicates. It is known that there is a dichotomy between PTime and NP for such CSPs if and only if there is such a dichotomy for unrestricted CSPs [19]. Consequently, the mentioned OMQ languages have a dichotomy between PTime and coNP if and only if the Feder-Vardi conjecture holds. Other results from CSP research carry over as well. Before we proceed with harvesting the fruits of Theorem 2, we give some remarks on extensions and variations of the theorem, all substantiated in [9]:

- the theorem also provides insight on the expressive power of OMQs from the perspective of more traditional database query languages; in fact, coMMSNP can be seen as a notational variant of monadic disjunctive Datalog, which thus has the same expressive power as $(\mathcal{ALC}, \text{UCQ})$;
- $\mathcal{ALC}$ can be replaced with several other standard description logics such as $\mathcal{ELU}$, $\mathcal{ALCI}$ and $\mathcal{SHI}$, without invalidating the theorem;
- there are some standard features of DLs whose addition to $\mathcal{ALC}$ breaks Theorem 2, for example: (1) if $\mathcal{ALC}$ is extended with forms of counting such as funtional roles or number restrictions, then the resulting class of OMQs provably has no dichotomy between PTime and coNP; (2) the extension of $\mathcal{ALC}$ with transitive roles increases the expressive power beyond coCSP/coMMSNP, but it is not known whether the dichotomy holds or fails;
- while the equivalences given in Theorem 2 are effective, there are substantial differences in succinctness; whereas the translation from coCSP/coMMSNP to OMQs is polynomial, the converse translation is exponential and must be superpolynomial unless ExpTime $\subseteq$ coNP/poly; for OMQs based on the mild extension $\mathcal{ALCI}$ of $\mathcal{ALC}$, only a double exponential translation to coCSP/coMMSNP is known;
- argueably, the practically most important query languages are AQs and CQs; OMQs based on AQs correspond to a slightly generalized (and well-understood) form of CSPs with multiple templates and a single constant symbol. There is indeed no known natural counterpart to OMQs based on conjunctive queries on the CSP/MMSNP side.

An important and very active topic of OMQ research is to rewrite OMQs into equivalent queries that are formulated in traditional database query languages, in this way enabling the use of conventional database systems for efficient OMQ answering. Particularly important target query languages include FO (aka SQL) and Datalog. Rewriting into these languages is not always possible, witnessed for example by the OMQ from Section 2 that expresses non-3-colorability, which can be expressed neither in FO nor in Datalog [1]. However, the simple structure of real-world ontologies gives hope that rewriting will be possible in many practically relevant cases, and there is experimental evidence supporting this hope [23, 39, 40]. Ideally, one would like to have an approach for rewritings OMQs that is complete in the sense that it finds a (preferably small and simple) rewriting if there is one and otherwise reports non-existance. For OMQ languages such as $(\mathcal{ALC}, \text{AQ})$ and $(\mathcal{ALC}, \text{UCQ})$, this is non-trivial to attain. Interesting initial results can be carried over from the CSP world.

It is known that FO-definability and Datalog-definability of coCSPs are decidable [28, 7, 21]. These results can be lifted to multi-template CSPs with a single constant symbol [9].

Theorem 2 and its adaptation to $(\mathcal{ALC}, \text{AQ})$ thus yields the upper bounds in the following theorem; the lower bounds are established by reductions from a tiling problem.

▶ **Theorem 3** ([9]). *In* $(\mathcal{ALC}, \text{BAQ})$ *and* $(\mathcal{ALC}, \text{AQ})$*, FO-rewritability and Datalog-rewritability are decidable and* NExpTime*-complete.*

In principle, the techniques in [28] also allow the construction of concrete FO-rewritings, and the result from [7] that Datalog-definability of coCSPs implies definability by Datalog programs of width three together with the canonical Datalog programs constructed by Feder and Vardi in [19] give a way to construct concrete Datalog-rewritings. However, naively applying such approaches will hardly be practical. More insight into the shape and construction of rewritings might be gained from the theory of obstructions, which have received significant attention in the CSP world.

A set of relational structures Γ is an *obstruction set* for a relational structure T (all in signature Σ) if for all finite Σ-structures S, we have $S \not\to T$ if and only if $O \to S$ for some $O \in \Gamma$. Obstructions are important because if an OMQ Q is equivalent to the complement of $\text{CSP}(T)$ and Γ is an obstruction set of T, then $\bigvee_{O \in \Gamma} q_O$ is a (potentially infinitary) rewriting of Q where q_O is the Boolean CQ whose graph is O. Therefore, the CSP result that FO-definability of $\text{coCSP}(T)$ implies the existence of finite set of finite tree-shaped obstructions for T [4, 28] translates into the OMQ result that any FO-rewritable OMQ has an FO-rewriting which is a UCQ that consists of tree-shaped CQs. Such results can potentially be used to guide the design of algorithm that construct rewritings and to study the succinctness of rewritings. Many other results about obstructions are available. For example, Datalog-rewritability is related to the existence of (an infinite set of) obstructions of bounded treewidth [19]. It is interesting to note that allowing answer variables in OMQs (which is roughly the same as extending CSPs with constants) changes the shape of obstructions in non-trivial ways, see [2, 9].

Theorem 3 only mentions atomic queries, but neither CQs nor UCQs. In fact, FO-definability and Datalog-definability of the complements of MMSNP sentences does not seem to have been studied in the CSP literature. We have recently observed that the problem is harder than for CSPs.

▶ **Theorem 4** ([13]). *In* $(\mathcal{ALC}, \text{CQ})$ *and in coMMSNP, FO-rewritability and Datalog-rewritability are* 2NExpTime*-hard.*

In very recent (and yet unpublished) work with Cristina Feier, we were able to show that FO-rewritability and monadic Datalog rewritability are decidable for $(\mathcal{ALC}, \text{CQ})$ and coMMSNP, with a 2NExpTime upper bound.

5 Summary

The connection between OMQs and CSPs brings interesting new results and techniques for OMQs. It also provides additional motivation for studying CSPs and underlines the importance of MMSNP which, despite having been the subject of some very interesting studiessuch as [32, 11, 12], has so far received much less attention than CSP. The motivation provided by the OMQ connection does not even stop at MMSNP. Some natural OMQ languages such as (GF, UCQ) with GF denoting the guarded fragment of FO have the same expressive power as an extension of MMSNP known as MMSNP_2 or GMSNP. Intuititively, the transition from MMSNP to GMSNP corresponds to replacing monadicity with guardedness, which increases expressive power [9]. Very little is known about the computational properties of this extended language.

There are other classes of database problems that are potentially quite closely connected to CSPs and related formalisms. In principle, it is interesting to consider the CSP connection for all querying problems that are, implicitly or explicitly, based on a certain answers semantics. One example is view-based query processing, for which a CSP connection has been established in [15]. Another one is *consistent query answering (CQA)*, where a query is answered over a set of databases that emerges from repairing a given database which violates its integrity constraints [3]. First observations on the connection between CSP and CQA have been made in [20, 31].

References

1 Foto N. Afrati, Stavros S. Cosmadakis, and Mihalis Yannakakis. On datalog vs. polynomial time. *J. Comput. Syst. Sci.*, 51(2):177–196, 1995.

2 Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang Chiew Tan. Characterizing schema mappings via data examples. *ACM Trans. Database Syst.*, 36(4):23, 2011.

3 Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Proc of PODS*, pages 68–79. ACM Press, 1999.

4 Albert Atserias. On digraph coloring problems and treewidth duality. *Eur. J. Comb.*, 29(4):796–820, 2008.

5 Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proc. of IJCAI*, pages 364–369, 2005.

6 Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2nd edition, 2010.

7 Libor Barto and Marcin Kozik. Constraint satisfaction problems of bounded width. In *Proc. of FOCS*, pages 595–603, 2009.

8 Meghyn Bienvenu and Magdalena Ortiz. Ontology-mediated query answering with data-tractable description logics. In *Proc. of Reasoning Web*, volume 9203 of *LNCS*, pages 218–307. Springer, 2015.

9 Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014.

10 Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge University Press, 2001.

11 Manuel Bodirsky, Hubie Chen, and Tomás Feder. On the complexity of MMSNP. *SIAM J. Discrete Math.*, 26(1):404–414, 2012.

12 Manuel Bodirsky and Víctor Dalmau. Datalog and constraint satisfaction with infinite templates. *J. Comput. Syst. Sci.*, 79(1):79–100, 2013.

13 Pierre Bourhis and Carsten Lutz. Containment in monadic disjunctive datalog, mmsnp, and expressive description logics. *Submitted*, 2016.

14 Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, and Andreas Pieris. Datalog+/-: A family of languages for ontology querying. In *Proc. of Datalog Reloaded*, volume 6702 of *LNCS*, pages 351–368. Springer, 2010.

15 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query processing and constraint satisfaction. In *Proc. of LICS*, pages 361–371. IEEE Computer Society, 2000.

16 Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, and Riccardo Rosati. Ontologies and databases: The DL-Lite approach. In *Reasoning Web*, volume 5689 of *LNCS*, pages 255–356, 2009.

17 Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.

18 Tomás Feder. Classification of homomorphisms to oriented cycles and of k-partite satisfiability. *SIAM J. Discrete Math.*, 14(4):471–480, 2001.

19 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.

20 Gaëlle Fontaine. Why is it hard to obtain a dichotomy for consistent query answering? *ACM Trans. Comput. Log.*, 16(1):7:1–7:24, 2015.

21 Ralph Freese, Marcin Kozik, Andrei Krokhin, Miklós Maróti, Ralph KcKenzie, and Ross Willard. On Maltsev conditions associated with omitting certain types of local structures. In preparation. Manuscript available from `http://www.math.hawaii.edu/~ralph/Classes/619/OmittingTypesMaltsev.pdf`, 2009.

22 Valentin Goranko and Martin Otto. *Handbook of Modal Logic*, chapter Model Theory of Modal Logic, pages 255–325. Elsevier, 2006.

23 Peter Hansen, Carsten Lutz, Inanç Seylan, and Frank Wolter. Efficient query rewriting in the description logic $\mathcal{EL}$ and beyond. In *Proc. of IJCAI*, pages 3034–3040. AAAI Press, 2015.

24 Pavol Hell and Jaroslav Nesetril. On the complexity of H-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990.

25 Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: the making of a web ontology language. *J. Web Sem.*, 1(1):7–26, 2003.

26 Roman Kontchakov and Michael Zakharyaschev. An introduction to description logics and query rewriting. In *Proc. of Reasoning Web*, volume 8714 of *LNCS*, pages 195–244. Springer, 2014.

27 Richard E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, 1975.

28 Benoit Larose, Cynthia Loten, and Claude Tardif. A characterisation of first-order constraint satisfaction problems. *Logical Methods in Comp. Sci.*, 3(4), 2007.

29 Carsten Lutz, Robert Piro, and Frank Wolter. Description logic TBoxes: Model-theoretic characterizations and rewritability. In *Proc. of IJCAI*, pages 983–988, 2011.

30 Carsten Lutz and Frank Wolter. Non-uniform data complexity of query answering in description logics. In *Proc. of KR*, 2012.

31 Carsten Lutz and Frank Wolter. On the relationship between consistent query answering and constraint satisfaction problems. In *Proc. of ICDT*, pages 363–379, 2015.

32 Florent R. Madelaine and Iain A. Stewart. Constraint satisfaction, logic and forbidden patterns. *SIAM J. Comput.*, 37(1):132–163, 2007.

33 Marie-Laure Mugnier and Michaël Thomazo. An introduction to ontology-based query answering with existential rules. In *Proc. of Reasoning Web*, volume 8714 of *LNCS*, pages 245–278. Springer, 2014.

34 W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 2009. Available at `http://www.w3.org/TR/owl2-overview/`.

35 W3C RDF Working Group. *RDF Primer*. W3C Recommendation, 2004. Available at `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`.

36 Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., 2006.

37 Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artif. Intell.*, 48(1):1–26, 1991.

38 Kent Spackman. Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with SNOMED-RT. *J. of the Am. Med. Inf. Ass.*, 2000.

39 Despoina Trivela, Giorgos Stoilos, Alexandros Chortaras, and Giorgos B. Stamou. Optimising resolution-based rewriting algorithms for OWL ontologies. *J. Web Sem.*, 33:30–49, 2015.

40 Yujiao Zhou, Bernardo Cuenca Grau, Yavor Nenov, Mark Kaminski, and Ian Horrocks. Pagoda: Pay-as-you-go ontology query answering using a datalog reasoner. *J. Artif. Intell. Res. (JAIR)*, 54:309–367, 2015.

Fine-Grained Algorithms and Complexity

Virginia Vassilevska Williams

Stanford University, USA
virgi@cs.stanford.edu

Abstract

A central goal of algorithmic research is to determine how fast computational problems can be solved in the worst case. Theorems from complexity theory state that there are problems that, on inputs of size n, can be solved in $t(n)$ time but not in $t(n)^{1-\epsilon}$ time for $\epsilon > 0$. The main challenge is to determine where in this hierarchy various natural and important problems lie. Throughout the years, many ingenious algorithmic techniques have been developed and applied to obtain blazingly fast algorithms for many problems. Nevertheless, for many other central problems, the best known running times are essentially those of the classical algorithms devised for them in the 1950s and 1960s.

Unconditional lower bounds seem very difficult to obtain, and so practically all known time lower bounds are conditional. For years, the main tool for proving hardness of computational problems have been NP-hardness reductions, basing hardness on $P \neq NP$. However, when we care about the exact running time (as opposed to merely polynomial vs non-polynomial), NP-hardness is not applicable, especially if the problem can already be solved in polynomial time. In recent years, a new theory has been developed, based on "fine-grained reductions" that focus on exact running times. The goal of these reductions is as follows. Suppose problem A is solvable in $a(n)$ time and problem B in $b(n)$ time, and no $a(n)^{1-\epsilon}$ and $b(n)^{1-\epsilon}$ algorithms are known for A and B respectively, for any $\epsilon > 0$. Then if A is fine-grained reducible to problem B (for $a(n)$ and $b(n)$), then a $b(n)^{1-\epsilon}$ time algorithm for B (for any $\epsilon > 0$) implies an $a(n)^{1-\epsilon'}$ algorithm for A (for some $\epsilon' > 0$). Now, mimicking NP-hardness, the approach is to (1) select a key problem X that is conjectured to require $t(n)^{1-o(1)}$ time for some $t(n)$, and (2) reduce X in a fine-grained way to many important problems. This approach has led to the discovery of many meaningful relationships between problems, and even sometimes to equivalence classes.

In this talk I will give an overview of the current progress in this area of study, and will highlight some new exciting developments.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, F.1.3 Complexity Measures and Classes

Keywords and phrases fine-grained reductions, polynomial time problems, equivalence, hardness

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.3

Category Invited Talk

© Virginia V. Williams;
licensed under Creative Commons License CC-BY
33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 3; pp. 3:1–3:1

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

Tutorial on Cellular Automata and Tilings

Jarkko Kari

Mathematics Department, University of Turku, Finland
jkari@utu.fi

Abstract

Cellular automata (CA) are massively parallel systems where a regular grid of finite symbols is updated according to a synchronous application of the same local update rule everywhere. A closely related concept is that of Wang tiles where a local relation between neighboring symbols determines allowed combinations of symbols in the grid. In this tutorial we start with classical results on cellular automata, such as the Garden-of-Eden theorems, the Curtis-Hedlund-Lyndon-theorem and the balance property of surjective cellular automata. We then discuss Wang tiles and, in particular, the concept of aperiodicity and the undecidability of the domino problem. The domino problem is the decision problem to determine if a given Wang tile set admits any valid tilings of the grid. We relate Wang tiles to cellular automata, and establish a number of undecidability results for cellular automata.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Cellular Automata, Tilings, Domino Problem, Undecidability, Dynamical Systems

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.4

Category Tutorial

© Jarkko Kari;
licensed under Creative Commons License CC-BY
33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 4; pp. 4:1–4:1

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

Graph Reconstruction with a Betweenness Oracle

Mikkel Abrahamsen[*1], **Greg Bodwin**[2], **Eva Rotenberg**[3], **and Morten Stöckel**[†4]

1 **University of Copenhagen Department of Computer Science, Denmark**
roden@di.ku.dk
2 **Stanford University, Department of Computer Science, USA**
gbodwin@cs.stanford.edu
3 **University of Copenhagen Department of Computer Science, Denmark**
miab@di.ku.dk
4 **University of Copenhagen Department of Computer Science, Denmark**
most@di.ku.dk

Abstract

Graph reconstruction algorithms seek to learn a hidden graph by repeatedly querying a black-box oracle for information about the graph structure. Perhaps the most well studied and applied version of the problem uses a distance oracle, which can report the shortest path distance between any pair of nodes.

We introduce and study the *betweenness* oracle, where $\text{bet}(a, m, z)$ is true iff m lies on a shortest path between a and z. This oracle is strictly weaker than a distance oracle, in the sense that a betweenness query can be simulated by a constant number of distance queries, but not vice versa. Despite this, we are able to develop betweenness reconstruction algorithms that match the current state of the art for distance reconstruction, and even improve it for certain types of graphs. We obtain the following algorithms:

1. Reconstruction of general graphs in $O(n^2)$ queries
2. Reconstruction of degree-bounded graphs in $\tilde{O}(n^{3/2})$ queries
3. Reconstruction of geodetic degree-bounded graphs in $\tilde{O}(n)$ queries

In addition to being a fundamental graph theoretic problem with some natural applications, our new results shed light on some avenues for progress in the distance reconstruction problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases graph reconstruction, bounded degree graphs, query complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.5

1 Introduction

Background and Applications

A major subfield of graph algorithms is that of *graph reconstruction*, in which one must determine the edges of a hidden graph using a black-box oracle that reveals a certain type of information about the graph. This model is typically used to study systems in which it is costly or slow to make measurements on the graph; the subject of reconstruction research is therefore to find strategies that learn the graph with low worst-case *query complexity*. The

* Research partially supported by Mikkel Thorup's Advanced Grant from the Danish Council for Independent Research under the Sapere Aude research career programme.
† Research supported by Villum Fonden.

© Mikkel Abrahamsen, Greg Bodwin, Eva Rotenberg, and Morten Stöckel;
licensed under Creative Commons License CC-BY

offline computation time of these algorithms is not a central point of concern, although it is generally expected to be polynomial.

This framework captures a number of important problems in computational biology. In one well-studied example, researchers wish to learn evolutionary trees, but only have tools to query for the distance in the unknown tree between an arbitrary pair of species (see [25, 13, 16, 22] for work on this problem). Each of these queries requires some research effort, and so one hopes to choose queries efficiently so as to maximize the information gained from each one. Another reconstruction problem is implicitly studied in genome sequencing; in this version, the underlying oracle takes a node subset S and reports whether or not this set is independent (see [1, 2, 4, 3, 11] for work on this problem). Yet another version is central to bioinformatics, in which the underlying oracle must count the number of internal edges in a node subset S (see [6, 12] for work on this problem). There is also the *network reconstruction problem*, in which one runs tests on a system to learn the topology of a decentralized network – for example, one might hope to discover the graph of the internet by querying for connectivity information between routers (see [5, 8, 9, 10] for work on this problem). Reconstruction problems also appear in network tomography [7, 23], probability theory [19], and other fields. In parallel, there has been work on the closely related *graph verification problem* in which one must simply confirm that the oracle matches a graph taken on input. For some examples of work on this problem, see [15, 5, 7, 10].

We introduce a new oracle in this paper, which is most similar to the distance oracle. We will therefore proceed through this introduction with our attention restricted to the distance reconstruction problem. If the reader is interested in more exposition on reconstruction/verification under the *other* oracles, we refer them to [15, 21, 3, 5, 1, 2, 4, 18] and the citations therein.

Our New Oracle

We introduce and study the *betweenness* oracle, which is defined such that $\text{bet}(a, m, z)$ is true iff there is a shortest path between a and z that contains m.

There are a few reasons why we consider this oracle worthy of study. The first is that this oracle generalizes the distance oracle due to the following equivalence:

$$\text{bet}(a, m, z) \qquad \Leftrightarrow \qquad \text{dist}(a, m) + \text{dist}(m, z) = \text{dist}(a, z)$$

It is therefore possible to simulate a betweenness query with a constant number of distance queries, and so any query complexity obtained for the betweenness reconstruction problem is automatically achieved for the distance reconstruction problem as well. It is not hard to see that one *cannot* in general simulate a distance query with a constant number of betweenness queries: in fact, given no other information about the graph, at least $\Omega(n)$ betweenness queries are required to learn a single pairwise distance. In this sense, we regard the betweenness oracle as *much weaker* than the distance oracle.

Given this intuition, it is very natural to expect that betweenness reconstruction would require a higher query complexity than distance reconstruction. However, as of our new results in this paper, this is not the case! We are able to match the most important state of the art results for distance reconstruction, and even improve them for certain classes of hidden graphs.

This surprise sheds some light on the central open problem [17, 15] of distance reconstruction, which is to obtain an algorithm with query complexity $n^{1+o(1)}$ for degree-bounded graphs. We are able to match the current best query complexity ($\tilde{O}(n^{3/2})$ from [17]) for this

problem using only a betweenness oracle. This suggests that the current distance reconstruction algorithms are unable to exploit the additional power of their oracle in a meaningful way, and that further progress can perhaps be made through using this information in a more careful manner.

Our second reason for studying this oracle is purely practical: it is useful to study reconstruction problems in a context free of a distance model. For example, in the evolutionary tree literature, the distance reconstruction approach asks biologists to devise a model of evolutionary distance, and then use this to reconstruct the order of speciation events [13]. There is no single definitive method for modeling evolutionary distance, and all popular methods for measuring these distances are somewhat error-prone and require certain underlying assumptions to be made (see [20] for a discussion of evolutionary distance estimation). In contrast, the betweenness reconstruction approach allows one to reconstruct the tree without having to worry about a model of evolutionary distance. Instead, one only needs to answer yes-or-no betweenness queries about the tree structure. Per the distance/betweenness relationship given above, this is a strictly easier task.

Our third reason for valuing the betweenness oracle is purely theoretical: intuitively, we consider the betweenness oracle to be by far the weakest oracle under which nontrivial reconstruction results have been obtained. Previously, this distinction belonged to the distance oracle. All other oracles receiving significant attention had non-constant arity and/or returned a polynomial number of bits on output, and the best reconstruction results for these other oracles had significantly lower query complexity (typically $\tilde{O}(n)$) than was known for distance oracles ($O(n^2)$ for general graphs); see [21] for a survey. Since our betweenness oracle is strictly weaker than a distance oracle, it should be regarded as the new weakest studied oracle.

Our Results

Our first main result is:

▶ **Theorem.** *There is a betweenness reconstruction algorithm with a query complexity of $O(n^2)$ (this algorithm makes no assumptions about the hidden graph).*

For general graphs, the trivial brute force distance reconstruction algorithm uses $O(n^2)$ queries, but the brute force betweenness reconstruction algorithm uses $O(n^3)$ queries, so some cleverness is required to avoid making all possible queries. An interesting consequence of our new oracle is apparent here: because our oracle returns only a single bit of information, there is a simple matching information-theoretic lower bound matching this upper bound. A graph encodes $\Theta(n^2)$ bits of information, each betweenness query reports a single bit of information, and so at least $\Omega(n^2)$ queries are required in the worst case.

This particular argument fails for distance reconstruction, which reports $\log n$ bits of information per query. However, a matching lower bound is still known. A simple lower bound is a graph with $n-2$ isolated vertices plus a pair of vertices connected by a single edge; the only possible reconstruction algorithm here is a brute-force search for the missing edge. To forbid this construction, it is common to assume that the hidden graph is connected. However, this is still not enough: Reyzin & Srivastava [21] have exhibited a constant-depth tree that requires $\Omega(n^2)$ distance queries to reconstruct. This tree has a root of degree $\Omega(n)$; therefore, the most natural way to forbid this construction is to parametrize the solution quality on the maximum degree Δ of the graph. In many natural applications Δ is constant or at most $n^{o(1)}$ [17, 13], so in this parametrization it is considered much more important to reduce the dependence on n than to reduce the dependence on Δ.

Our next main result fits within this parametrization. We prove:

▶ **Theorem.** *When the hidden graph is connected with maximum degree Δ, there is a betweenness reconstruction algorithm with a query complexity of $\tilde{O}(n^{3/2} \cdot \Delta^4)$.*

This matches the query complexity obtained by Mathieu & Zhou [17] for distance reconstruction. It is generally considered to be the foremost open problem [17, 15] to obtain distance reconstruction for degree-bounded graphs with a query complexity of $n^{1+o(1)} \cdot f(\Delta)$ for any function f. We consider it quite interesting that the state-of-the-art against this open problem can be achieved by our betweenness oracle: it suggests that the current algorithms for this problem do not integrally exploit the fact that they receive full distance information, and that an avenue for progress is to devise an algorithm that uses distance information in a new way.

While this $n^{3/2}$ bound remains unbroken, the problem has been solved for certain specific types of graphs. In particular, degree-bounded distance reconstruction algorithms with query complexities of $\tilde{O}(n \cdot f(\Delta))$ are known when the hidden graph is outerplanar [17] or chordal [15]. Our third main result is in this regime. We prove:

▶ **Theorem.** *When the hidden graph is connected, geodetic (i.e., there is a unique shortest path between every pair of nodes), and has maximum degree Δ, there is a betweenness reconstruction algorithm with a query complexity of $\tilde{O}\left(n \cdot \Delta^3\right)$.*

Geodetic graphs can be seen as a generalization of trees; in this vein, our upper bound comes very close to the distance reconstruction lower bound of $\tilde{\Omega}(n \cdot \Delta)$ for trees given in [16]. Due to the relationship between betweenness and distance queries discussed above, an identical upper bound is immediate for distance reconstruction, thereby improving the distance query complexity for these graphs by a factor of $O(\sqrt{n})$ over [17].

Finally, we remark that our algorithms in the two latter cases are Monte Carlo, and that the error probability can be tuned to an arbitrarily small inverse polynomial in n by increasing the hidden constant.

2 Terminology

We will first collect some previously used notations and definitions from the world of graph theory that will be used repeatedly throughout the paper.

We reserve $G = (V, E)$ to refer to the hidden graph being reconstructed, and $|V| = n$. Unless otherwise noted, the graph is undirected and unweighted. Given a node subset $U \subset V$, $G[U]$ is the subgraph induced by U. The neighborhood of a node v (i.e. all nodes u such that $(v, u) \in E$) is denoted by $N(v)$. The neighborhood of a node set S (i.e. the union of $N(s) - S$ for all $s \in S$) is denoted by $N(S)$.

▶ **Definition 1** (Starshaped)**.** A node subset $X \subset V$ is *starshaped* with respect to a *center* $x \in X$ if, for all $v \in X$, every shortest path from x to v is entirely contained in X.

Note that the same subset may be starshaped with respect to several centers.

▶ **Definition 2** (Layer Structure)**.** Given a starshaped set $X \subset V$ with center x, a node $v \in X$ is said to be in layer i of X if $\text{dist}(x, v) = i$. The set of nodes in layer i is denoted L_i^X. The X superscript is suppressed when clear from context. (See Figure 1.)

▶ **Definition 3** (Spanning Tree)**.** Given a starshaped set $X \subset V$ with center x, a subgraph $\mathcal{T}_X$ of $G[X]$ is a *spanning tree* of X if it is a tree with the property that for all $v \in X$, $\mathcal{T}_X$ includes a shortest path in X from x to v.

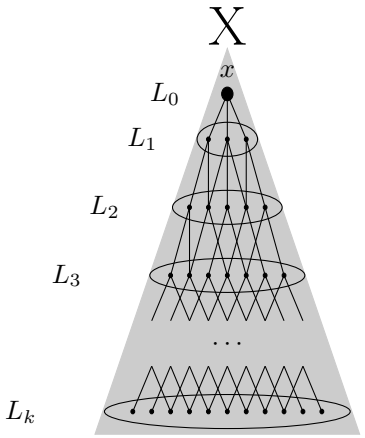


Figure 1 The layer structure for a starshaped set X with center x, as well as the shortest path graph.

▶ **Definition 4** (Shortest Path Graph)**.** Given a starshaped set $X \subset V$ with center x, the *shortest path graph* of X is the subgraph $\mathcal{S}_X$ of $G[X]$ defined by removing all edges (a, b) where a and b are in the same layer of X.

By orienting all edges of the Shortest Path Graph away from x, one obtains a DAG with source x, and thus, a notion of ancestry and parenthood between the vertices of X. We formalize in the following way:

▶ **Definition 5** (Tree Definitions)**.** Let X be a starshaped set with center x. If $v \in L_i^X$, then u is a *parent* of v if $u \in N(v) \cap L_{i-1}^X$, or u is a *child* of v if $u \in N(v) \cap L_{i+1}^X$. The *ancestor* relation is the transitive closure of the parent relation, and the *descendant* relation is the transitive closure of the child relation. By convention, a node is both its own ancestor and its own descendant. The set of descendants of a node v is denoted $\mathcal{D}(v)$, and the set of ancestors of a node v is denoted $\mathcal{A}(v)$. A node is a *leaf* if it has no children.

▶ **Definition 6** (Geodetic Graphs)**.** A graph is *geodetic* if, for each node pair, there is a unique shortest path between these nodes.

Note that, for a geodetic graph, there is a unique spanning tree of any starshaped set X, and this spanning tree is also the shortest path graph.

▶ **Definition 7** (Betweenness)**.** We define the relation $\text{bet}(\cdot,\cdot,\cdot)$ such that $\text{bet}(a, m, z)$ is true iff there exists a shortest path in G between a and z that includes m.

▶ **Definition 8** (Ancestry)**.** For a "root node" $r \in V$, we define the relation $\text{anc}_r(\cdot,\cdot)$ such that $\text{anc}_r(u, v) \leftrightarrow \text{bet}(r, u, v)$. We will sometimes use this query in place of a bet query to emphasize that our goal is to test membership of u in $\mathcal{A}(v)$.

3 Reconstruction of General Graphs

There are $\Omega(n^3)$ different betweenness queries that one can ask on any given graph. However, we prove:

▶ **Theorem 9.** *One can learn the edges of a hidden graph using $O(n^2)$ betweenness queries (with no assumptions made about the hidden graph; G can be disconnected and have any maximum degree).*

Note that there is a simple matching information-theoretic lower bound for general graphs: a graph encodes $\Omega(n^2)$ bits of information, and since each betweenness query returns a single bit of information, at least this many queries are required to learn the graph.

The upper bound proceeds in two steps. The first is to discover the connected components of the graph:

▶ **Claim 10.** *One can discover the connected components of a hidden graph using $O(n^2)$ betweenness queries.*

Proof. For all $u, v \in V$, query $\text{bet}(u, u, v)$. The query will be true iff there exists a shortest path between u and v, which holds only if u and v belong to the same connected component of the graph. ◀

The second step in the upper bound is to discover the edges of a single connected component. We will prove this result at a higher level of generality, as the techniques will be useful again later in the paper.

▶ **Claim 11.** *Given a starshaped set X with center x, as well as the shortest path graph of X, one can decide whether or not there exists an edge between any two nodes u, v in the hidden graph using $O(1)$ betweenness queries.*

Proof. If u, v belong to different layers, then the task is trivial: if there is an edge between them, then it is in the shortest path graph, so one must simply consult the shortest path graph and no queries need to be executed. So assume u, v are in the same layer. Let node a be a least common ancestor of u and v; i.e. choose a node $a \in \mathcal{A}(u) \cap \mathcal{A}(v)$ with the highest level. Since u, v are in the same layer, we then have $\text{dist}(u, a) = \text{dist}(v, a) =: h$. Let b be a child of a along a shortest path between a and u. Then $\text{dist}(b, u) = h - 1$ and $\text{dist}(b, v) \in \{h, h+1\}$. Query $\text{bet}(v, a, b)$. If this query is true, then we have $\text{dist}(b, v) = h+1$, which then implies that the edge (u, v) does *not* exist in the hidden graph. If this query is false, then we have $\text{dist}(b, v) = h$. We then query $\text{bet}(v, u, b)$, which will be true iff the edge (u, v) exists in the hidden graph. ◀

▶ **Claim 12.** *Let $X \subset V$ be a starshaped set with center x. One can discover all edges in $G[X]$ in $O(|X|^2)$ betweenness queries.*

Note that if X is an entire connected component of the graph, then it is trivially starshaped with respect to any $x \in X$, and so along with Claim 10, this result implies Theorem 9.

Proof. The algorithm proceeds in two steps. First, we query $\text{anc}_x(u, v)$ for all $u, v \in X$, and we claim that this information is sufficient to learn the shortest path graph $\mathcal{S}_X$. We can learn the layers by the following recursive formula:

$$L_0 = \{x\}\,,$$
$$L_i = \left\{ v \in (X - \bigcup_{j<i} L_j) \quad | \quad (\mathcal{A}(v) - \{v\}) \subset \bigcup_{j<i} L_j \right\}.$$

This formula states that if layers $\{0, \ldots, i-1\}$ are known, then the i^{th} layer can be determined as the set of nodes v such that all ancestors (besides the node itself) belong to these first layers. Once the layers are known, for any node $v \in L_i$, we can determine its neighbors in L_{i-1} as the set

$$N(v) \cap L_{i-1} = \mathcal{A}(v) \cap L_{i-1}\,.$$

Applied to all $v \in X$, this is sufficient to learn every edge in $\mathcal{S}_X$. We can now use Claim 11 on all pairs of edges in the graph to detect the existence/nonexistence of every possible edge in $O(|X|^2)$ queries. ◀

4 Finding Splitting Nodes

For the rest of this paper, we will assume that G is connected with maximum degree Δ.

▶ **Lemma 13.** *Every starshaped set X with center x has a node $s \in X$ with the property*

$$\left\lceil \frac{|X|}{3\Delta} \right\rceil \leq |\mathcal{D}(s)| \leq \left\lceil \frac{|X|}{3} \right\rceil.$$

Proof. Let ℓ be a leaf of X. We then have

$$|\mathcal{D}(\ell)| = |\{\ell\}| = 1.$$

And so if $|X| \leq 3\Delta$, then ℓ satisfies the property. Otherwise, assume $|X| > 3\Delta$.

We will now proceed with an intermediate value type argument. Initialize a node $p \leftarrow x$, and then repeatedly find the child $c_{\max}$ of p with the greatest number of descendants and set $p \leftarrow c_{\max}$. Stop this process once p is set to a leaf. At each step, we have

$$\left| \bigcup_{c \text{ is a child of } p} \mathcal{D}(c) \right| = |\mathcal{D}(p) - \{p\}| = |\mathcal{D}(p)| - 1.$$

Since p has at most Δ children, the union is taken over Δ elements, and so by a (inverse) union bound, the average child has at least $(|\mathcal{D}(p)| - 1)/\Delta$ descendants, and so $c_{\max}$ has at least this many descendants. So if

$$|\mathcal{D}(p)| > \left\lceil \frac{|X|}{3} \right\rceil$$

then

$$|\mathcal{D}(c_{\max})| \geq \left\lceil \left\lceil \frac{|X|}{3} \right\rceil / \Delta \right\rceil \geq \left\lceil \frac{|X|}{3\Delta} \right\rceil.$$

Note that the size of the set $\{v \in X \mid \text{anc}_x(p, v)\}$ is strictly decreasing in each step. Its initial value is $|X|$ and its final value is 1. Therefore, at some point during this process, we have

$$|\mathcal{D}(p)| > \left\lceil \frac{|X|}{3} \right\rceil \quad \text{and} \quad |\mathcal{D}(c_{\max})| \leq \left\lceil \frac{|X|}{3} \right\rceil.$$

From the above argument, we also have

$$|\mathcal{D}(c_{\max})| \geq \left\lceil \frac{|X|}{3\Delta} \right\rceil.$$

and so $c_{\max}$ satisfies our property. ◀

A node is a *splitting node* if it satisfies a relaxed version of this condition (see Figure 2).

▶ **Definition 14.** A node s satisfying

$$\left\lceil \frac{|X|}{4\Delta} \right\rceil \leq |\mathcal{D}(s)| \leq \left\lceil \frac{|X|}{2} \right\rceil$$

is called a *splitting node*.

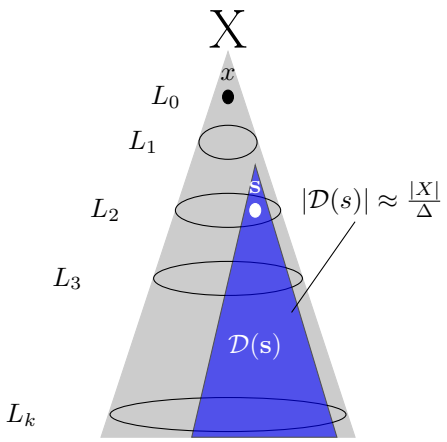


Figure 2 A representation of a splitting node s of the starshaped set X.

▶ **Lemma 15.** *Given a starshaped set X with center x with hidden edges, there is a randomized algorithm that uses $\tilde{O}(|X| \cdot \Delta)$ betweenness queries*[1] *and, with high probability, finds a splitting node $s \in X$.*

Proof. Iterate over each node $v \in X$. Let R be a uniform random sample S of $k \cdot \log|X| \cdot \log\log|X| \cdot \Delta$ nodes, where k is a constant whose size determines the probability of success. We then query $\text{anc}_x(v, r)$ for each $r \in R$, and we estimate $|\mathcal{D}(v)|$ as:

$$\widehat{|\mathcal{D}(v)|} = |\mathcal{D}(v) \cap R| \cdot \frac{|X|}{|R|}$$

By standard Chernoff bounds, for sufficiently large k, we have

$$|\mathcal{D}(v)| - \frac{|X|}{29\Delta} \leq \widehat{|\mathcal{D}(v)|} \leq |\mathcal{D}(v)| + \frac{|X|}{29\Delta}$$

with high probability. We then accept v as a splitting node iff

$$\left\lceil \frac{|X|}{3.5\Delta} \right\rceil \leq \widehat{|\mathcal{D}(v)|} \leq \left\lceil \frac{|X|}{2.5} \right\rceil.$$

Then, for an accepted node v, if $|X|$ is sufficiently large then the following inequalities hold with high probability:

$$\left\lceil \frac{|X|}{4\Delta} \right\rceil < \left\lceil \frac{|X|}{3.5\Delta} \right\rceil - \frac{|X|}{29\Delta} \leq |\mathcal{D}(v)| \leq \left\lceil \frac{|X|}{2.5} \right\rceil + \frac{|X|}{29\Delta} < \left\lceil \frac{|X|}{2} \right\rceil,$$

and hence v is a splitting node with high probability. Similarly, we see that a non-splitting node is rejected with high probability. Additionally, with high probability, we will accept any node satisfying the condition in Lemma 13. Since at least one such node exists, this process will find a splitting node with high probability.

We use $O(\log|X| \cdot \log\log|X| \cdot \Delta)$ queries per node in X, so the total number of queries used by this process is $O(|X| \cdot \log|X| \cdot \log\log|X| \cdot \Delta)$. ◀

In the context of graph reconstruction, the technique of random sampling and querying over the sample for the sake of set size estimation was first used by Mathieu & Zhou in [17] for a different purpose.

[1] When the notation $\tilde{O}(f(|X|))$ is used, the $\tilde{O}$ hides $\mathsf{polylog}(|X|)$ factors, not $\mathsf{polylog}(n)$.

5 Reconstruction of Spanning Trees

We next prove:

▶ **Lemma 16.** *Given a starshaped set X with center x, there is a randomized algorithm that uses $\tilde{O}(|X| \cdot \Delta^2)$ betweenness queries to learn a spanning tree of X.*

We first need the following technical results:

▶ **Claim 17.** *Let X be a starshaped set with center x, and let $v \in X$. Then $\mathcal{D}(v)$ is starshaped with center v.*

Proof. Let $u \in \mathcal{D}(v)$, let $\rho(u, v)$ be any shortest path between u and v, and let $w \in \rho(u, v)$. We then have $\text{dist}(v, w) + \text{dist}(w, u) = \text{dist}(v, u)$. Since $u \in \mathcal{D}(v)$, we also have $\text{dist}(x, v) + \text{dist}(v, u) = \text{dist}(x, u)$, and so combining these equations, we have

$$\text{dist}(x, v) + \text{dist}(v, w) = \text{dist}(x, u) - \text{dist}(w, u).$$

By the triangle inequality, we have $\text{dist}(x, u) \leq \text{dist}(x, w) + \text{dist}(w, u)$, and so

$$\text{dist}(x, v) + \text{dist}(v, w) \leq \text{dist}(x, w).$$

Again by the triangle inequality, we also have

$$\text{dist}(x, v) + \text{dist}(v, w) \geq \text{dist}(x, w)$$

and so

$$\text{dist}(x, v) + \text{dist}(v, w) = \text{dist}(x, w).$$

This implies that $w \in \mathcal{D}(v)$, and so $\mathcal{D}(v)$ is starshaped with center v. ◀

We omit the proofs of the following claims, as they are generally similar to the previous one.

▶ **Claim 18.** *Let X be a starshaped set with center x, and let $v \in X$. Then $X - \mathcal{D}(v)$ is starshaped with center x.*

▶ **Claim 19.** *Let X be a starshaped set with center x, and let $v \in X$. Then $(X - \mathcal{D}(v)) \cup \{v\}$ is starshaped with center x.*

These allow us to prove the following result:

▶ **Claim 20.** *Let X be a starshaped set with center x, and let $v \in X$. Let $\mathcal{T}_{(X-\mathcal{D}(v))\cup\{v\}}$ be a spanning tree of the set $(X - \mathcal{D}(v)) \cup \{v\}$ with center x, and let $\mathcal{T}_{\mathcal{D}(v)}$ be a spanning tree of $\mathcal{D}(v)$ with center v. Then $\mathcal{T}_{(X-\mathcal{D}(v))\cup\{v\}} \cup \mathcal{T}_{\mathcal{D}(v)}$ is a spanning tree of X.*

Proof. First, note that $\mathcal{T}_{(X-\mathcal{D}(v))\cup\{v\}}$ and $\mathcal{T}_{\mathcal{D}(v)}$ have only the node v in common; therefore, their union is a tree.

Let $u \in X$. If $u \notin \mathcal{D}(v)$, then (by Claim 19) every shortest path between u and x is contained in $(X - \mathcal{D}(v)) \cup \{v\}$, and so $\mathcal{T}_{(X-\mathcal{D}(v))\cup\{v\}}$ contains a shortest path between u and x.

Otherwise, suppose $u \in \mathcal{D}(v)$. Then $\mathcal{T}_{\mathcal{D}(v)}$ contains a shortest path between v and u, and $\mathcal{T}_{(X-\mathcal{D}(v))\cup\{v\}}$ contains a shortest path between x and v. Since u is a descendant of v, we have $\text{dist}(x, u) = \text{dist}(x, v) + \text{dist}(v, u)$. Therefore, the union of these two shortest paths is a shortest path between x and u.

We then have that $\mathcal{T}_{(X-\mathcal{D}(v))\cup\{v\}} \cup \mathcal{T}_{\mathcal{D}(v)}$ is a tree that contains a shortest path between x and any $v \in X$, so it is a spanning tree of X. ◀

We can now return to Lemma 16.

Proof of Lemma 16. The algorithm is by recursive divide-and-conquer. The base case is when X consists of 2Δ or fewer nodes; in this case, it is possible to learn all edges of X in $O(\Delta^2)$ queries using Claim 12, and then find a spanning tree within these edges without any additional queries.

Repeat the following process. Use Lemma 15 to find a splitting node $s \in X$ in $\tilde{O}(|X| \cdot \Delta)$ queries. Next, query $\text{anc}_x(s, v)$ for all $v \in X$ in order to determine the set $\mathcal{D}(s)$. Next, recursively compute a spanning tree of the sets $\mathcal{D}(s)$ and $(X - \mathcal{D}(s)) \cup \{s\}$ (these sets are starshaped by Claims 17 and 19), and then return the union of these two spanning trees. By Claim 20, this is a spanning tree of X.

The size of each set decreases by at least a factor of $(1 - \Theta(\frac{1}{\Delta}))$ at each round of the recursion, and so the recursion depth is $O(\log |X| \cdot \Delta)$. The number of top-level queries in each round is $\tilde{O}(|X| \cdot \Delta)$. This implies that $\tilde{O}(|X| \cdot \Delta^2)$ queries in total are used. ◀

▶ Remark. We note that the unweightedness of the hidden graph has not been exploited in any way, and therefore, Lemma 16 applies even if the hidden graph is weighted. More specifically, it is impossible for a betweenness query to learn the weight of a weighted edge, but one can find the edge set of a spanning tree using only $\tilde{O}(|X| \cdot \Delta^2)$ betweenness queries.

6 Reconstruction of Degree-Bounded Graphs

We closely follow the algorithm developed by Mathieu & Zhou in [17] for distance reconstruction. At a high level our algorithm is exactly the same as theirs (which is, in turn, based on techniques from [14] and [24]); our contribution is that we are able to use two of our previously established techniques to replace their distance queries with betweenness queries.

We prove:

▶ **Theorem 21.** *One can learn the edges of a hidden connected graph with maximum degree Δ in $\tilde{O}(n^{3/2} \cdot \Delta^4)$ betweenness queries.*

The following definition is critical to the proof:

▶ **Definition 22** (Cluster, [24])**.** Given a set of "centers" $A \subset V$ and a node $w \in V$, the *cluster* of w, denoted $C^A(w)$, is defined as $\{v \in V \mid \text{dist}(w, v) < \text{dist}(A, v)\}$.[2]

Mathieu & Zhou prove the following reduction (it is implicit in their Lemma 2):

▶ **Lemma 23** ([17])**.** *Let s be a parameter. Suppose there is an algorithm that learns the distance from a node v to each other node in a hidden graph, and that this algorithm uses $\leq f(n, \Delta)$ queries of some type. Then there is a randomized algorithm on this hidden graph that, with high probability, produces a set A of size $O(s \log n)$ such that $C^A(w) = O(n/s)$ for all $w \in V$. This randomized algorithm uses $\tilde{O}(f(n, \Delta) \cdot s)$ queries of the same type.*

Mathieu & Zhou use distance queries, and so trivially $f(n, \Delta) \leq n$. However, our Lemma 16 implies that for betweenness queries, we have $f(n, \Delta) = \tilde{O}(n \cdot \Delta^2)$. We can therefore find a set A as in Lemma 23 using $\tilde{O}(n \cdot \Delta^2 \cdot s)$ betweenness queries. The first step in the proof of Theorem 21 is to find A as in Lemma 23, and to find all pairwise distances in $A \times V$.

We next borrow some more methodology from Mathieu & Zhou.

[2] Where $\text{dist}(A, v)$ is a shorthand for $\min_{a \in A} \text{dist}(a, v)$.

▶ **Definition 24** ([17]). For each $a \in A$, define $\mathcal{C}(a)$ as

$$\{v \in V \mid \text{dist}(v, a) < \text{dist}(v, a') + 2 \text{ for all } a' \in A \setminus \{a\}\}$$

▶ **Claim 25** ([17]). *For all (hidden) edges $(u, v) \in G$, there exists an $a \in A$ such that $u, v \in \mathcal{C}(a)$.*

▶ **Claim 26** ([17]). $|\mathcal{C}(a)| = O(n/s \cdot \Delta^2)/$

The next claim is ours:

▶ **Claim 27.** *For all $a \in A$, the set $\mathcal{C}(a)$ is starshaped with center a.*

Proof. Let $u \in \mathcal{C}(a)$, let $\rho(a, u)$ be a shortest path between a and u, and let $w \in \rho(a, u)$. Suppose towards a contradiction that $w \notin \mathcal{C}(a)$; that is, there exists $a' \in A$ such that $\text{dist}(a, w) \geq \text{dist}(a', w) + 2$. We then have $\text{dist}(a, u) = \text{dist}(a, w) + \text{dist}(w, u)$, and by the triangle inequality, $\text{dist}(a', u) \leq \text{dist}(a', w) + \text{dist}(w, u)$. Combining these equations, we have

$$\begin{aligned} \text{dist}(a, u) - \text{dist}(a, w) &\geq \text{dist}(a', u) - \text{dist}(a', w) \\ &\geq \text{dist}(a', u) - (\text{dist}(a, w) - 2), \end{aligned}$$

and hence $\text{dist}(a, u) \geq \text{dist}(a', u) + 2$. Therefore $u \notin \mathcal{C}(a)$, which is a contradiction. We then have $w \in \mathcal{C}(a)$, and so $\mathcal{C}(a)$ is starshaped. ◀

We now return to Theorem 21.

Proof of Theorem 21. Compute the set A as in Lemma 23; by Lemma 16 this requires $\tilde{O}(n \cdot \Delta^2 \cdot s)$ betweenness queries. Once again use Lemma 16 to compute all pairwise distances in $A \times V$, and use this information to compute $\mathcal{C}(a)$ for all $a \in A$. Next, since each $\mathcal{C}(a)$ is starshaped (Claim 27), we can invoke Claim 12 to learn all edges in each set $\mathcal{C}(a)$. By Claim 26, the query complexity of this step is $|A| \cdot \tilde{O}(n^2/s^2 \cdot \Delta^4) = \tilde{O}(n^2/s \cdot \Delta^6)$. So the total query complexity of this process is $\tilde{O}(n \cdot \Delta^2 \cdot s) + \tilde{O}(n^2/s \cdot \Delta^6)$. The claim now follows by setting $s = n^{1/2}\Delta^2$. ◀

▶ Remark. A reasonable objection that can be made here is that we should model Δ as an unknown quantity, and so it is unfair to choose the value of s based on Δ. There is a way around this: one can make the assumption $\Delta = 2$ and run the algorithm as stated, aborting as soon as the runtime bound corresponding to $\Delta = 2$ is exceeded. If the bound is exceeded, then double the value of Δ and try again until the algorithm terminates successfully.

7 Reconstruction of Geodetic Graphs

Recall that a graph is *geodetic* if there is a unique shortest path between every pair of nodes. We next prove:

▶ **Theorem 28.** *One can learn the edges of a hidden geodetic connected graph with maximum degree Δ in $\tilde{O}(n \cdot \Delta^3)$ betweenness queries.*

This result was previously unknown, even for distance oracles.

We begin with a reduction of the problem. The input of our new problem is a starshaped set X with center x, a node $s \in X$, and a node $v \in (X - \mathcal{D}(s))$. Additionally, we are given the shortest path graph of X. The problem is to learn all edges with one endpoint at v and the other in $\mathcal{D}(s)$. We call this the *boundary edge problem*. Our reduction to this problem is as follows:

▶ **Lemma 29.** *Assume G is geodetic. Suppose there is a (possibly randomized) algorithm that solves the boundary edge problem using $f(|X|, \Delta)$ betweenness queries in the worst case. Then there is a randomized algorithm that, given a starshaped set X with center x, learns all edges in $G[X]$ using $\tilde{O}(|X| \cdot \Delta^3 \cdot f(|X|, \Delta))$ betweenness queries.*

Proof. We can learn the graph as follows. Use Lemma 16 to construct a spanning tree of X using $\tilde{O}(|X| \cdot \Delta^2)$ queries. Because the graph is geodetic, the spanning tree is also the shortest path graph of X. Next, find a splitting node s of X, using the algorithm outlined in Lemma 15. Solve the boundary edge problem for each node $v \in (X - \mathcal{D}(s))$; this requires $O(|X| \cdot f(|X|, \Delta))$ queries. We have now learned all edges with one endpoint in $\mathcal{D}(s)$ and the other endpoint in $X - \mathcal{D}(s)$; we still need to learn the edges with both endpoints in $\mathcal{D}(s)$ or both endpoints in $X - \mathcal{D}(s)$. We know that $\mathcal{D}(s)$ is starshaped with center s, and $X - \mathcal{D}(s)$ is starshaped with center x (by Claims 17 and 18), so this can be done recursively.

In each round of the recursion, we partition the node set into two subsets, each of which has at most a $1 - \Theta(1/\Delta)$ fraction as many nodes as in the previous round of the recursion. Therefore, the maximum recursion depth is $\tilde{O}(\Delta)$. The number of top-level queries made is $\tilde{O}(|X| \cdot \Delta^2 + |X| \cdot f(|X|, \Delta))$ and so the total query complexity of the algorithm is $\tilde{O}(|X| \cdot \Delta^3 + |X| \cdot \Delta \cdot f(|X|, \Delta))$. ◀

What remains is to place bounds on $f(|X|, \Delta)$, the query complexity of the boundary edge problem. To this end, the following claims are useful:

▶ **Claim 30.** *If the hidden graph is geodetic, then any edge (v, u) with $v \in (X - \mathcal{D}(s))$ and $u \in \mathcal{D}(s)$ are in the same layer in the set X.*

Proof. Let $u \in \mathcal{D}(s)$ be a neighbor of v. Let L_u^X be the layer of u, and let L_v^X be the layer of v. By the triangle inequality, we have that (1) $L_u^X = L_v^X + 1$, or (2) $L_u^X = L_v^X$, or (3) $L_u^X + 1 = L_v^X$. In fact, (3) is impossible because it implies that $v \in \mathcal{D}(s)$ and we have assumed $v \in (X - \mathcal{D}(s))$. Additionally, (1) is impossible because the hidden graph is geodetic. Specifically, since $u \in \mathcal{D}(s)$ there is a shortest path from x to u that passes through s; since $L_v^X + 1 = L_u^X$ there is another shortest path from x to u that passes through v (and therefore doesn't pass through s, since $v \notin \mathcal{D}(s)$); these shortest paths must be distinct. So (2) is the only possible case: we have $L_u^X = L_v^X$. ◀

▶ **Claim 31.** *If the hidden graph is geodetic, then for any edge (v, u) with $v \in (X - \mathcal{D}(s))$ and $u \in \mathcal{D}(s)$, we have* $\text{dist}(v, s) = \text{dist}(u, s) + 1$.

Proof. Let $L_u = L_v$ be the layer of u and v (these are the same by Claim 30), and let L_s be the layer of s. Since $u \in \mathcal{D}(s)$, we have $\text{dist}(u, s) = L_u - L_s$. Therefore, it cannot be the case that $\text{dist}(v, s) = \text{dist}(u, s) - 1$: this would imply that $\text{dist}(v, s) = L_v - L_s - 1$, which violates the triangle inequality. Additionally, it cannot be the case that $\text{dist}(v, s) = \text{dist}(u, s)$: this would imply that $\text{dist}(v, s) = L_v - L_s$, and so $v \in \mathcal{D}(s)$, which we have assumed is not true. The only possibility that remains is that $\text{dist}(v, s) = \text{dist}(u, s) + 1$. ◀

▶ **Claim 32.** *If the hidden graph is geodetic, then each node $u \in (X - \mathcal{D}(s))$ has at most one incident edge in $\mathcal{D}(s)$.*

Proof. Let v be a neighbor of u in $\mathcal{D}(s)$. From Claim 31, we have that $\text{dist}(u, s) = \text{dist}(v, s) + 1$, and so u lies on a shortest path from v to s. If there are two distinct neighbors $u, u' \in \mathcal{D}(s)$ of u, then both of these neighbors lie on a shortest path from v to s, implying the existence of two distinct shortest paths from v to s. This violates our geodetic assumption. ◀

We can now prove:

▶ **Lemma 33.** *The boundary edge problem can be solved for hidden geodetic graphs in $\tilde{O}(\Delta)$ betweenness queries.*

Proof. The algorithm runs in two stages. First, we use $\tilde{O}(\Delta)$ betweenness queries to find a "candidate node" u, which is defined to be the node in $\mathcal{D}(s)$ furthest from x that lies on the shortest path between v and s. Note that if there exists an edge (v, u) with $u \in \mathcal{D}(s)$, then u will certainly be the candidate node by Claim 31. Therefore, once we have identified a candidate node u, it only remains to test whether or not the edge (u, v) is in the hidden graph, and we have detected the unique edge from v into $\mathcal{D}(s)$ or refuted its existence.

We can find the candidate node using our standard splitting node technique. Find a splitting node t of $\mathcal{D}(s)$ (we already have a shortest path graph of X, of which the shortest path graph of $\mathcal{D}(s)$ is a subgraph, so this can be computed offline and requires no queries). Query $\text{bet}(v, t, s)$. If true, then we recurse on the starshaped set $\mathcal{D}(t)$; if false, then we recurse on the starshaped set $\mathcal{D}(s) - \mathcal{D}(t)$, and repeat until only one node remains; this node is our candidate node. As usual, the recursion depth is $\tilde{O}(\Delta)$, and so we use this many betweenness queries.

Once we have our candidate node u, we can test for the existence of the edge (u, v) in a constant number of queries using Claim 11. ◀

Jointly, Lemmas 29 and 33 imply Theorem 28.

References

1 N. Alon and V. Asodi. Learning a hidden subgraph. *SIAM Journal of Discrete Math*, 18:697–712, 2005.

2 N. Alon, R. Beigel, S. Kasif, S. Riduch, and B. Sudakov. Learning a hidden matching. *SIAM Journal of Computing*, 33:487–501, 2004.

3 D. Angluin and J. Chen. Learning a hidden graph using $o(\log n)$ queries per edge. *COLT*, pages 210–223, 2004.

4 D. Angluin and J. Chen. Learning a hidden hypergraph. *Journal of Machine Learning Research*, 7:2215–2236, 2006.

5 Z. Beerliova, F. Eberhard, T. Erlebach, E. Hall, M. Hoffman, and L. S. Ram. Network discovery and verification. *WG*, pages 127–138, 2005.

6 M. Bouvel, V. Grebinski, and G. Kucherov. Combinatorial search on graphs motivated by bioinformatics applications: A brief survey. *Graph-Theoretic Concepts in Computer Science*, pages 16–27, 2005.

7 R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu. Network tomography: recent developments. *Statistical Science*, 19:499–517, 2004.

8 D. Chen, L. J. Guibas, J. Hershberger, and J. Sun. Road network reconstruction for organizing paths. *SODA*, pages 1309–1320, 2010.

9 L. Dall'Asta, I. Alvarez-Hamelin, A. Barrat, A. Vázquez, and A. Vespignani. Exploring networks with traceroute-like probes: Theory and simulations. *Theoretical Computer Science*, 355(1):6–24, 2006.

10 T. Erlebach, A. Hall, M. Hoffmann, and M. Mihal'ák. Network discovery and verification with distance queries. *Algorithms and Complexity*, pages 69–80, 2006.

11 V. Grebinski and G. Kucherov. Reconstring a hamiltonian cycle by querying the graph: Application to dna physical mapping. *Discrete Applied Mathematics 88*, pages 147–165, 1998.

12 V. Grebinski and G. Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica 28*, pages 104–124, 200.

13 J. J. Hein. An optimal algorithm to reconstruct trees from additive distance data. *Bulletin of Mathematical Biology*, 51(5):597–603, 1989.

14 S. Honiden, M. E. Houle, and C. Sommer. Balancing graph voronoi diagrams. *ISVD*, pages 183–191, 2009.

15 S. Kannan, C. Mathieu, and H. Zhou. Near-linear query complexity for graph inference. *Automata, Languages, and Programming*, 2015.

16 V. King, L. Zhang, and Y. Zhou. On the complexity of distance-based eviolutionary tree reconstruction. *SODA*, pages 444–453, 2003.

17 C. Mathieu and H. Zhou. Graph reconstruction via distance oracles. *ICALP*, pages 733–744, 2013.

18 H. Mazzawi. Optimally reconstructing weighted graphs using queries. *SODA*, pages 608–615, 2010.

19 S. Micali and Z. Allen-Zhu. Reconstructing markov processes from independent and anonymous experiments. *Discrete Applied Mathematics*, 2015.

20 M. Nei and J. Zhang. Evolutionary distance: Estimation. *Encyclopedia of Life Sciences*, 2005.

21 L. Reyzin and N. Srivastava. Learning and verifying graphs using queries with a focus on edge counting. *Proc. 18th Algorithmic Learning Theory*, pages 285–297, 2007.

22 L. Reyzin and N. Srivastava. On the longest path algorithm for reconstructing trees from distance matrices. *Information processing letters*, 101(3):98–100, 2007.

23 F. Tarissan, M. Latapy, and C. Prieur. Efficient measurement of complex networks using link queries. *INFOCOM Workshops*, pages 254–259, 2009.

24 M. Thorup and U. Zwick. Compact routing schemes. *SPAA*, pages 1–10, 2001.

25 M. S. Waterman, T. F. Smith, M. Singh, and W. Beyer. Additive evolutionary trees. *Journal of Theoretical Biology*, 64(2):199–213, 1977.

Airports and Railways: Facility Location Meets Network Design*

Anna Adamaszek[1], Antonios Antoniadis[2], and Tobias Mömke[3]

1 **University of Copenhagen, Denmark**
anad@di.ku.dk

2 **Max-Planck-Institut für Informatik, Saarbrücken, Germany**
aantonia@mpi-inf.mpg.de

3 **Saarland University, Germany**
moemke@cs.uni-saarland.de

Abstract

We introduce a new framework of Airport and Railway Problems, which combines capacitated facility location with network design. In this framework we are given a graph with weights on the vertices and on the edges, together with a parameter k. The vertices of the graph represent cities, and weights denote respectively the costs of opening airports in the cities and building railways that connect pairs of cities. The parameter k can be thought of as the capacity of an airport. The goal is to construct a minimum cost network of airports and railways connecting the cities, where each connected component in the network spans at most k vertices, contains an open airport, and the network satisfies some additional requirements specific to the problem in the framework.

We consider two problems in this framework. In the $\mathrm{AR_F}$ problem there are no additional requirements for the network. This problem is related to capacitated facility location. In the $\mathrm{AR_P}$ problem, we require each component to be a path with airports at both endpoints. $\mathrm{AR_P}$ is a relaxation of the capacitated vehicle routing problem (CVRP).

We consider the problems in the two-dimensional Euclidean setting. We show that both $\mathrm{AR_F}$ and $\mathrm{AR_P}$ are NP-hard, even for uniform vertex weights (i. e., when the cost of building an airport is the same for all cities). On the positive side, we provide polynomial time approximation schemes for $\mathrm{AR_F}$ and $\mathrm{AR_P}$ when vertex weights are uniform. We also investigate $\mathrm{AR_F}$ and $\mathrm{AR_P}$ for $k = \infty$. In this setting we present an exact polynomial time algorithm for $\mathrm{AR_F}$ with general vertex costs, which also works for general edge costs. In contrast to $\mathrm{AR_F}$, $\mathrm{AR_P}$ remains NP-hard when $k = \infty$, and we present a polynomial time approximation scheme for general vertex weights.

We believe that our PTAS for $\mathrm{AR_P}$ with uniform vertex weights and arbitrary k brings us closer towards a PTAS for Euclidean CVRP, for which the main difficulty is to deal with paths of length at most k.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases approximation algorithms, geometric approximation, facility location, network design, PTAS

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.6

* Partially supported by the Danish Council for Independent Research DFF-MOBILEX mobility grant, and by Deutsche Forschungsgemeinschaft grant BL511/10-1 and MO 2889/1-1.

© Anna Adamaszek, Antonios Antoniadis, and Tobias Mömke;
licensed under Creative Commons License CC-BY

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 6; pp. 6:1–6:14

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

1 Introduction

We introduce a new framework of Airport and Railway Problems, which combines capacitated facility location with network design. In this framework, an input instance (G, a, r, k) consists of a complete n-vertex graph $G = (V, E)$ with vertex costs $a\colon V(G) \to \mathbb{R}$ and edge costs $r\colon E(G) \to \mathbb{R}_{\geq 0}$, and a parameter k. The goal is to compute a minimum cost network of airports $A \subseteq V(G)$ and railways $R \subseteq E(G)$ connecting all the cities (i. e., each vertex in V is connected with some vertex from A via a path of edges from R), where each connected component of the network contains at most k vertices. Problems from this framework can have additional constraints on the network, e. g., requiring a fixed number κ of connected components, or enforcing special structure of the connected components. Unless stated differently, we assume that r specifies distances in the two-dimensional Euclidean space.

Paths. We define $\mathrm{AR_P}$ as an airport and railway problem with an additional property that each connected component is a path with airports at the endpoints. Formally, we want to find subsets $A \subseteq V(G)$ and $R \subseteq E(G)$ of minimum total cost $a(A) + r(R)$[1] such that (i) each connected component of the graph (V, R) is a path of length at most k, and (ii) for each $v \in V$ we have $v \in A$ if and only if v is an endpoint of a path in R. For consistency we assume that each path contains two airports, i. e., when accounting for the cost of A, we consider each vertex that forms a singleton path twice.[2] For a solution H to $\mathrm{AR_P}$, we define $\tilde{a}(H) = \sum_{v \in A} a(v)$, where A is the multiset of all endpoints of the paths of H.

The $\mathrm{AR_P}$ problem relaxes the capacitated vehicle routing problem (CVRP), which is a fundamental vehicle routing problem (cf. Dantzig and Ramser [8]). The input to CVRP is an edge-weighted graph with a special vertex called the *depot*, and a parameter k, the capacity. The goal is to find a minimum cost tour that (i) starts and ends at the depot, (ii) visits all vertices, and (iii) visits at most k vertices between any two consecutive visits to the depot. CVRP is a special case of $\mathrm{AR_P}$, where for each vertex $v \in V$ the cost $a(v)$ equals the distance between v and the depot. The existence of a PTAS for Euclidean CVRP remains an important open problem.

Notice that in some applications where goods need to be distributed or a service has to be provided to the customers, unlike in CVRP, the cost of moving the salesmen to and from the depot is not proportional to the distance from the depot. For example, a salesman can move to and from the depot using public transport, and then $a(v)$ will denote the cost of the public transport ticket. These applications motivate a model where the airport costs $a(v)$ are uniform (which we later denote by $1\mathrm{AR_P}$), as well as the general $\mathrm{AR_P}$ model.

Trees. We define $\mathrm{AR_F}$ as a basic airport and railway problem, with no additional properties of the network. Formally, we want to find subsets $A \subseteq V(G)$ and $R \subseteq E(G)$ of minimum total cost $a(A) + r(R)$ such that each connected component of the graph (V, R) has at most k vertices, and contains one vertex from A (i. e., one open airport).[3] We remark that since r is nonnegative, any optimal solution to $\mathrm{AR_F}$ is always a forest, and that it opens the cheapest airport in each component. As for paths, for a solution H we define $\tilde{a}(H) = \sum_{v \in A} a(v)$, but here A is the set that contains the vertex with the cheapest airport of each component.

---

[1] We write $a(A)$ and $r(R)$ as shorthands for $\sum_{v \in A} a(v)$ and $\sum_{e \in R} r(e)$, respectively.

[2] We remark that our results can be easily adapted for the case where we count the cost of an airport at a singleton vertex once instead of twice.

[3] We restrict the number of airports per component for technical reasons. It is straightforward to adapt our algorithms to allow multiple airports per component.

$\mathrm{AR_F}$ is related to the well known capacitated facility location problem, where instead of connecting each client directly to an open facility we construct a network which jointly connects all clients to a facility, which seems more appropriate for some applications. Another interesting interpretation of $\mathrm{AR_F}$ is that each subset U of at most k vertices forms a hyperedge. The cost of each such hyperedge is $r(H) + \tilde{a}(H)$, where H is a minimum spanning tree on U, and the goal is to find a perfect hyper-matching of minimum cost.

1.1 Our Results

We consider the $\mathrm{AR_P}$ and $\mathrm{AR_F}$ problems in the two-dimensional Euclidean setting. For both $\mathrm{AR_P}$ and $\mathrm{AR_F}$ we consider the uniform airport cost case, i. e., the case where $a(v) = a(w)$ for all $v, w \in V$, and the unrestricted airport capacity case, when $k = \infty$. We call the resulting problems $1\mathrm{AR_P}$, $1\mathrm{AR_F}$ and ${\mathrm{AR_P}}^{\infty}$, ${\mathrm{AR_F}}^{\infty}$.

Using a reduction from the Euclidean TSP path problem, i.e., the problem of finding a minimum cost Hamiltonian path in a given graph, we can prove the following result.

▶ **Theorem 1.** *The* $\mathrm{AR_P}$ *problem is NP-hard, even in Euclidean graphs with unit airport costs and* $k = \infty$.

In a similar manner, the APX-hardness of metric-TSP [14] implies that $\mathrm{AR_P}$ is APX-hard in metric graphs. The hardness result holds even when we have unit airport costs and $k = \infty$.

Theorem 1 implies in particular that both $1\mathrm{AR_P}$ and ${\mathrm{AR_P}}^{\infty}$ are NP-hard. On the positive side we develop polynomial time approximation schemes for these two problems. In fact, for ${\mathrm{AR_P}}^{\infty}$ we present a stronger result: we may specify the exact number of paths κ required by the solution. We denote the corresponding instance by (G, a, r, k, κ).

▶ **Theorem 2.** *For any* $\kappa \in \mathbb{N}$ *there is a PTAS for the two-dimensional Euclidean* ${\mathrm{AR_P}}^{\infty}$ *problem, when we require both the optimal and the algorithmic solution to have exactly* κ *connected components.*

Our technique for proving Theorem 2 is an intricate extension of the well known PTAS of Arora [4] for the Euclidean TSP. One obstacle is that unlike in Arora's result, in ${\mathrm{AR_P}}^{\infty}$ one cannot in general restrict the bounding box to be of linear size. We cope with this by scaling the instance with a carefully chosen scaling parameter and afterwards dissecting it into a collection of separate instances. A further difficulty is caused by crossings in the solution. We prove that paths can be uncrossed without an increase in the cost of the solution. After this preparation, we are able to extend Arora's dynamic program to be applicable for our setting. Theorem 2 is discussed in Section 3.2.

We can prove that the $1\mathrm{AR_F}$ problem is NP-hard, which in particular implies NP-hardness of the general $\mathrm{AR_F}$.

▶ **Theorem 3.** *The* $\mathrm{AR_F}$ *problem is NP-hard, even in Euclidean graphs with unit airport costs.*

The proof of the theorem consists of a reduction from the NP-hard planar monotone cubic One-in-Three Satisfiability problem (PMC-1-in-3-SAT). Notice that, in contrast to Theorem 1, this hardness result requires a bound on the airport capacity k.

We show that the ${\mathrm{AR_F}}^{\infty}$ problem, unlike ${\mathrm{AR_P}}^{\infty}$, can be solved exactly in polynomial time, even when we require that the solution has exactly κ connected components.

▶ **Theorem 4.** *For an arbitrary* $\kappa \in \mathbb{N}$ *and edge-cost function* $r\colon E \to \mathbb{R}_{\geq 0}$, *there is an exact polynomial-time algorithm for the* ${\mathrm{AR_F}}^{\infty}$ *problem when we require both the optimal and the algorithmic solution to have exactly* κ *connected components.*

To prove Theorem 4, we augment the input graph by a special vertex corresponding to the airports, and then we reduce the problem to the matroid intersection problem. Notice that here r is not required to be a metric. We discuss Theorem 4 in detail in Section 3.1.

We will use our algorithms for ${\rm AR_P}^\infty$ and ${\rm AR_F}^\infty$ with a requirement of κ on the number of connected components to obtain a PTAS for $1{\rm AR_F}$ and $1{\rm AR_P}$, which is the main result of this paper.

▶ **Theorem 5.** *There is a PTAS for the two-dimensional Euclidean* $1{\rm AR_P}$ *and* $1{\rm AR_F}$ *problem.*

We note that a generalization of the above result for $\rm AR_P$ from uniform airport costs to a constant range of airport costs would yield a PTAS for Euclidean CVRP, using a result of Adamaszek et al. [1].

In order to prove Theorem 5, we first partition the problem instance into a collection of independent subinstances. The partitioning borrows ideas from the random shift method used by Arora in his PTAS for Euclidean TSP. However, we obtain a collection of independent subinstances where each subinstance has a bounding box of constant size, instead of linear as Arora's random shift would give. We distinguish two types of subinstances: sparse and dense ones, depending on the number of connected components in an optimal solution for the instance. We obtain a PTAS for the sparse instances by again slightly adapting the PTAS by Arora. In contrast, developing an algorithm for dense instances is much more involved. We first obtain an initial solution by running our algorithms for ${\rm AR_P}^\infty$ (${\rm AR_F}^\infty$, respectively). This solution can be infeasible for $1{\rm AR_P}$ ($1{\rm AR_F}$, respectively), and we proceed by dissecting each connected component into smaller pieces of similar size. Finally, we show how these smaller pieces can be reassembled into a feasible solution which has cost not much greater than the cost of an optimal solution. Theorem 5 is discussed in Section 4.

We believe that our technique of dissecting a cheap but infeasible solution and reassembling the pieces to form a cheap feasible solution can be applied to other geometric graph problems, including Euclidean CVRP.

1.2 Related Work

Our new framework of Airport and Railway Problems combines capacitated facility location with network design. In the capacitated facility location (CFL) problem we are given a complete graph $G = (V, E)$ on n vertices $v_1, \ldots, v_n$, with edge costs $d\colon E(G) \to \mathbb{R}_{\geq 0}$ and vertex costs $c\colon V(G) \to \mathbb{R}_{\geq 0}$, together with a capacity parameter k. Here $d(v_i, v_j)$ denotes the distance between vertices v_i and v_j, and $c(v_i)$ denotes the cost of opening a facility at v_i. A solution to the problem consists of a set of facilities $F \subseteq V$ to be opened, and an assignment of each vertex v_i to some open facility $f(v_i)$ such that each facility has at most k vertices assigned to it. The goal is to find a solution minimizing the total cost of opening the facilities and connecting all vertices to the assigned facilities, i.e., $\sum_{v \in V : v \in F} c(v) + \sum_{v \in V} d(v, f(v))$. Here all vertices are connected to the open facilities by *direct links*.

In the network design problems we are given a graph with weights on the edges and possibly also on the vertices, and the goal is to find a minimum cost set of edges satisfying some given constraints, e.g., connectivity constraints. That might be for example constructing a network that allows routing flow from all the vertices to a *specified* set of sinks. Our framework combines the two settings above.

The $\rm AR_F$ problem is related to the problems of *capacitated facility location* and *capacitated minimum spanning tree* (CMST). In CMST the goal is to construct a minimum cost collection of trees, each spanning at most k vertices and connected to a single pre-specified root, covering all the input vertices. Jothi and Raghavachari [13] give a 3.15-approximation algorithm for

CMST instances in the Euclidean plane, allowing demands on the vertices. Other problems (more remotely) related to $\mathrm{AR_F}$ can be found in [15, 10] and [17].

The $\mathrm{AR_P}$ problem relaxes the capacitated vehicle routing problem (CVRP), which has been introduced by Dantzig and Ramser [8] in 1959, and has been studied extensively by the Computer Science and Operations Research communities. When the underlying graph G resembles a metric, CVRP is known to be APX-hard [6]. However, Arora [4] conjectured that when G resembles the Euclidean metric, then CVRP admits a PTAS. Das and Mathieu [9] developed a quasi-polynomial time approximation scheme (QPTAS) for Euclidean CVRP, which gives raise to expect the existence of a PTAS for this setting. There have been several results in this direction. There is a PTAS for the cases of very large capacity $k = \Omega(n)$ [6], and small capacity $k \leq 2^{\log^{o(1)} n}$ [1]. Despite these efforts, the exact approximability of CVRP in the Euclidean metric remains an open question. As Arora [4] and Das and Mathieu [9] point out, the main difficultly in adapting Arora's algorithm to CVRP with arbitrary parameter k lies in controlling the interface of the dynamic program. In particular, one cannot recombine different tours arbitrarily since an "uncrossing" may transform two tours of length at most k into one longer tour and one shorter tour. A second difficulty is due to the cost associated with returning to the depot. In the problems considered in this paper we were able to circumvent these obstacles, and we believe that our results could provide a step towards obtaining a PTAS for CVRP with an arbitrary parameter k.

Finally, the *capacitated location routing problem* (CLR, see [12]) and the *k-location capacitated vehicle routing* (k-LocVRP, see [11]) can be seen as problems in our framework, although they have been studied for general graphs and allow demands on the vertices. CLR is a generalization of multiple depot CVRP, with costs of opening the depots. k-LocVRP is also related to multiple depot CVRP, where we can choose k depots to be opened, one per vehicle. Harks et al. [12] and Gørtz and Nagarajan [11] provide constant-factor approximation algorithms for CLR and k-LocVRP, respectively. Another related problem is *capacitated geometric network design* (CGND) where the goal is to create a network of capacitated links which allows sending flow from all the vertices to the sink. CGND resembles $\mathrm{AR_F}$, when instead of a bound on the airport capacity we have a bound on the railroad capacity. In CGND the sink is pre-specified, and the optimal network can be more complicated than a tree. Adamaszek et al. [2] provide a PTAS for the two-dimensional Euclidean CGND for link capacities $k \leq 2^{O(\sqrt{\log n})}$, where the network can use Steiner vertices anywhere in the plane.

2 Preliminaries: Arora's Scheme

Some of our results build on Arora's PTAS for Euclidean TSP [3]. Therefore we provide a short reminder of the main steps of the algorithm.

Step 1 (Perturbation): Perturb the instance such that it becomes "well rounded", i. e., (i) all nodes are at integer coordinates, and (ii) the maximum inter-node distance is $O_\epsilon(n)$. Such an operation can be performed because the cost of an optimal TSP solution is at least as large as the maximum inter-node distance.

Step 2 (Shifted Quadtree): Starting from a square that contains all nodes of the instance, we recurse over a logarithmic number of levels, each time subdividing the square into four smaller subsquares. That gives us a quadtree of logarithmic depth, where the root corresponds to the square containing all nodes of the instance, and all other nodes of the tree correspond to the subsquares of the dissection. The leaves correspond to unit squares.

We introduce randomness and shift the quadtree in the following way. Choose two integers $a, b \in [0, L)$ uniformly at random, where L is the length of the bounding box side. Shift the basic quadtree by a units in the horizontal and b units in the vertical direction and wrap around the boundaries. Note that this does not affect the position of the nodes.

Step 3 (Dynamic Programming): Fix $m = O_\epsilon(\log n)$ and $r = O_\epsilon(1)$. We introduce portals on the plane in the following way. For the quadtree constructed in the previous step, for each side of each dissection square, we create m equidistant portals along the side. We call a TSP solution (m, r)-light if (i) it is portal-respecting (i.e., it enters and exits dissection squares only at the portals), and (ii) any edge of a dissection square is crossed at most r times by the solution. Arora showed that with respect to the random shift of Step 2, an optimal (m, r)-light TSP tour has an expected cost not much greater than the optimal TSP tour.

We can find an optimal (m, r)-light TSP tour using dynamic programming (DP). There is a DP cell for each combination of (i) a dissection square of the quadtree, and (ii) the description of pairs of portals used by the tour to enter and exit the square.

In the remaining document, when referring to Steps 1–3 of Arora's algorithm, we refer to the above steps.

3 Unrestricted Airport Capacity

In this section we will consider the problems $\text{AR}_\text{F}^\infty$ and $\text{AR}_\text{P}^\infty$.

In general, given some algorithm Alg, we denote by alg the cost of the solution computed by Alg. Similarly, an optimal solution Opt has a cost of opt.

3.1 An Exact Algorithm for $\text{AR}_\text{F}^\infty$ and Theorem 4

We first introduce a simple algorithm MST-AR_F for the $\text{AR}_\text{F}^\infty$ problem, for any edge-cost function $r\colon E \to \mathbb{R}_{\geq 0}$, and without the restriction on the number of connected components κ. The algorithm is a slight extension of finding a minimum spanning tree, and for any instance $(G, a, r, k = \infty)$ of $\text{AR}_\text{F}^\infty$ it performs the following operations.

1. Construct a minimum spanning tree instance (G', r') by augmenting (G, r) by an auxiliary vertex v', and an edge (v', v) with weight $a(v)$ for each $v \in V(G)$.
2. Compute a minimum spanning tree MST' of G'.
3. Output $A = \{v \in V : \{v, v'\} \in E(MST')\}$, $R = \{\{v_1, v_2\} \in E(MST') : v_1, v_2 \neq v'\}$.

It is easy to see that for each solution for the $\text{AR}_\text{F}^\infty$ instance there is a corresponding spanning tree of G' with the same cost, and vice versa. That gives us the following result.

▶ **Corollary 6.** MST-AR_F *is an exact polynomial time algorithm for* $\text{AR}_\text{F}^\infty$*, for any edge-cost function* $r\colon E \to \mathbb{R}_{\geq 0}$.

If we enforce a given number of connected components in the solution, as required by Theorem 4, the algorithm becomes more involved. It uses the augmented graph (G', r') and matroid intersection.

Proof of Theorem 4. The theorem follows from matroid intersection. Let $(E', \mathcal{I}_1)$ be the set system with $E' = E(G')$ and $\mathcal{I}_1$ the collection of the edge sets of all forests in G', where G' is the graph constructed in step one of MST-AR_F. Then $(E', \mathcal{I}_1)$ is the cycle matroid of G' and its bases are the spanning trees of G', of size $|V(G') - 1|$. To define a second matroid, let $E'' \subset E'$ be the set of all edges incident to v'. Then we define the second set system

$(E', \mathcal{I}_2)$ where a set $I \subseteq E'$ is in $\mathcal{I}_2$ if (i) $|E'' \cap I| \leq \kappa$ and (ii) $|(E' \setminus E'') \cap I| \leq |V(G')| - 1 - \kappa$. Clearly, $(E', \mathcal{I}_2)$ is the union of two uniform matroids and therefore a matroid itself. The bases of $(E', \mathcal{I}_2)$ are edge sets of size $|V(G') - 1|$ with exactly κ edges incident to v'.

The common bases of $(E', \mathcal{I}_1)$ and $(E', \mathcal{I}_2)$ are all spanning trees in G' with exactly κ edges from E'': they are spanning trees due to $(E', \mathcal{I}_1)$, and the κ edges incident to v' are ensured by $(E', \mathcal{I}_2)$. We can find a minimum cost common base in strongly polynomial time (see, for instance, Theorem 41.8 in Schrijver's book [16]). ◀

3.2 A PTAS for $\mathrm{AR_P}^\infty$ and Theorem 2

The $\mathrm{AR_P}^\infty$ problem is a generalization of the s,t-path TSP problem. To transform an s,t-path TSP instance to an instance of $\mathrm{AR_P}^\infty$, we set the airport costs of s and t to zero, and all other airport costs to infinity. This already implies NP-hardness of $\mathrm{AR_P}^\infty$, but in Theorem 1 we show that the problem is NP-hard even with unit airport costs. We extend Arora's scheme for Euclidean TSP to obtain a PTAS for the more general setting of $\mathrm{AR_P}^\infty$.

Perturbation. First, we need to obtain a well-rounded instance. We cannot immediately use the bounding box of Step 1 in Arora's algorithm for TSP, because there may be large gaps between distinct connected components of an optimal solution, and the length of the sides of the bounding box can be arbitrarily large with respect to opt. A similar problem occurs for example when trying to apply Arora's algorithm to the Euclidean k-median problem. Arora et al. [5] obtain a PTAS for the Euclidean k-median, where for the perturbation step they apply a min-max clustering algorithm of Bern and Eppstein [7]. Unlike in the case of k-median, we do not have such an algorithm. For $\mathrm{AR_P}^\infty$, we can prove the following result.

▶ **Lemma 7.** *For an arbitrary $\epsilon > 0$ and an instance $(G, a, r, k = \infty, \kappa)$ of $\mathrm{AR_P}^\infty$ requiring κ connected components, one can compute in polynomial time a collection of independent subproblems $(G_i, a_i, r_i, k = \infty, \kappa_i)$ such that (i) for each subproblem all vertices have integer coordinates and the maximum inter-node distance is bounded by $O_\epsilon(n^2)$, and (ii) if we know $(1+\epsilon)$-approximate solutions for all the subproblems we can compute in polynomial time a $(1 + O(\epsilon))$-approximate solution for the original problem instance.*

The main idea of the proof of Lemma 7 is to guess the longest edge e of an optimal solution (by trying all possibilities). Then $r(e)$ is a lower bound on opt. Using this lower bound and after an appropriate scaling of the costs, we can move all vertices to the nearest integer coordinates. We can also cluster the vertices such that the maximum inter-node distance in each cluster is $O_\epsilon(n^2)$, and the distance between any two vertices from different clusters is at least $2r(e)$. This ensures that we do not split any connected components of Opt between multiple clusters. That gives us independent problem instances, and from their solutions we can restore a solution for the original instance.

Random shift. We perform a random shift of the quadtree, as in Step 2 of Arora's algorithm. Let α, β be the random variables describing the offsets of the random shift. Note that in the quadtree, there is a new vertex at each portal. We refer to these vertices as *portal vertices*. Unlike all other vertices, the portal vertices are not required to be contained in the solution.

Uncrossing and (m, r)-thin solutions. Two paths P, P' are intersecting, if $P \cap P' \neq \emptyset$. We have to consider a technical detail here: there may be several vertices located at the same position, as well as pairs of edges whose intersection is an interval. We call an intersection

of two paths (i.e., a connected component of $P \cap P'$) a *crossing* if the intersection cannot be avoided by moving the vertices of the paths by infinitesimal distances. Similarly, a self-intersection of a single path that cannot be avoided in this way is also called a *crossing*.

We create m equidistant portals on each side of each dissection square, the same way as in the Arora scheme. We call a solution (m, r)-*thin* if it is portal-respecting, and each portal is crossed at most r times by the paths of the solution. Note that this is different from the notion of (m, r)-light, where each side of a square of the quadtree is entered at most r times.

The following lemma shows that there is a near-optimal solution which is (m, r)-thin with some good parameters m, r, and which has no crossings.

▶ **Lemma 8.** *Let $(G, a, r, k = \infty, \kappa)$ be a well-rounded instance of $\mathrm{AR_P}^\infty$ with an optimal solution* Opt*. Then, for an arbitrary $\epsilon > 0$, there is an $m = O_\epsilon(\log n)$ such that an optimal $(m, 2)$-thin solution* Opt$'$ *to this instance which has no crossings has expected cost of at most* $(1 + \epsilon)$opt.

Dynamic program. With this preparation we are ready to construct a dynamic program which computes a near-optimal solution for an instance of the $\mathrm{AR_P}^\infty$ problem.

▶ **Lemma 9.** *Let $(G, a, r, k = \infty, \kappa)$ be a well-rounded instance of $\mathrm{AR_P}^\infty$, and* Opt$'$ *an optimal $(m, 2)$-thin solution for this instance which has no crossings. There is a dynamic program which computes in polynomial time a feasible solution for the problem with cost at most* opt$'$.

The idea of the dynamic program is as follows. We specify the DP cells by the following parameters for each square S of the quadtree, where Π denotes the set of portals of S.

1. A number κ' of paths entirely contained in S.
2. For each portal $\pi \in \Pi$, a number η_π of paths within S which use π and have one endpoint inside S.
3. A multiset P of pairs (π, π') of portals such that there is a path within S connecting π and π', and such that the portal pairs do not generate crossings within S.

The DP extends Arora's DP by adding the parameters κ' and η_π. The parameter κ' is needed in order to control the total number of paths in the solution. The parameters η_π are essential since they enable building airports within the squares of the quadtree. For each dynamic program cell $DP(S, \kappa', (\eta_\pi)_{\pi\in\Pi}, P)$, corresponding to a dissection square S and parameters κ', $(\eta_\pi)_{\pi\in\Pi}$ and P, the DP computes the value of a minimum cost solution for S which is consistent with the above parameters.

Proof of Theorem 2. Using Lemma 7 we can reduce our problem instance to a collection of well-rounded instances. By Lemma 8, each such instance I has an $(m, 2)$-thin solution without crossings with an expected cost of at most $(1 + \epsilon)\mathrm{opt}_I$. By Lemma 9, there is a dynamic program which computes a solution for each I with an expected cost of at most $(1 + \epsilon)\mathrm{opt}_I$. That gives us a PTAS for $\mathrm{AR_P}$ with $k = \infty$, where additionally we can enforce the required number κ of connected components. ◀

4 $\mathrm{AR_P}$ and $\mathrm{AR_F}$ with Uniform Airport Costs

For simplifying the presentation, in this section we assume that there is only one airport required per component (even for $\mathrm{AR_P}$). Note that this is without the loss of generality, since all the airport costs are identical and we can just scale the $\mathrm{AR_P}$ instance down by a factor of 2. In this section we describe how to subdivide any given instance in the two-dimensional

Euclidean space into two classes of independent subinstances, and how to derive a PTAS for each of these classes. We assume without loss of generality that all airports have unit cost: for positive cost we can achieve this by scaling both the railway and airport costs by the same factor; otherwise we obtain an optimal solution by opening airports at all the vertices.

4.1 Preprocessing: Subdividing the Instance

The goal of this subsection is to simplify the input instance. We will describe how any given instance can be partitioned into a collection of smaller subinstances, such that each subinstance I_i can be bounded by an $\ell_i \times \ell_i$ bounding box, with $\ell_i \leq 1/\epsilon$. Furthermore, we will show that it suffices to solve each such subinstance independently. We will introduce a shifted *main grid* with cells of width $1/\epsilon$, and cut the instance along this main grid. We show that one can find shifting offsets such that it suffices to consider each cell of the main grid as a separate instance.

Let the original instance be inside an $L \times L$ bounding box, and let the point $(0, 0)$ be at the bottom left corner of the box. We divide the instance into subinstances, each bounded by a square of size $1/\epsilon \times 1/\epsilon$, as follows. Let $0 \leq \alpha, \beta < 1/\epsilon$ be real numbers. We introduce horizontal lines with a y-coordinate of $\beta + i \cdot 1/\epsilon$ and vertical lines with an x-coordinate of $\alpha + i \cdot 1/\epsilon$, where $i = 0, \ldots, \epsilon \cdot \lfloor L \rfloor$. We say that this creates a *main grid* with a shift (α, β). This main grid splits the bounding box into squares of size at most $1/\epsilon \times 1/\epsilon$. The following lemma implies that, for the case of unit airport costs, it suffices to consider each of these squares separately. In other words, in contrast to Arora's scheme for the Euclidean TSP where the bounding box has sides of length $L = O_\epsilon(n)$, for us it is enough to consider subinstances where the bounding box has sides of length $1/\epsilon = O_\epsilon(1)$.

▶ **Lemma 10.** *Let S be a solution of cost s to an instance I of either 1$\mathrm{AR_F}$ or 1$\mathrm{AR_P}$. Then there exists a shift (α, β) of the main grid defined above, and a solution S' of cost s' to I, obtained from S by removing some edges and adding new airports to maintain feasibility, such that* (i) *$s' \leq (1 + 2\epsilon)s$, and* (ii) *S' does not cross any line of the shifted main grid.*

Furthermore, we can compute in polynomial time a collection of shifts (α_i, β_i) such that at least one of them satisfies the above conditions.

For the rest of the section we may restrict our discussion to instances I that are within an $\ell \times \ell$ bounding square, with $\ell \leq 1/\epsilon$.

4.2 A PTAS for Sparse Subinstances

When dividing the original instance into subinstances, some subinstances may have optimal solutions with only a small number of components (at most $\frac{1}{\epsilon^7}$). This kind of instances can be handled with a slight adaption of Arora's scheme, as has already been observed by Asano et al. [6] for the capacitated vehicle routing problem. It can be easily seen that the adaptions of Arora's PTAS performed by Asano et al. [6] for CVRP can also be applied in our case.

4.3 A PTAS for Dense Subinstances

This subsection copes with the subinstances that are not covered above, i. e., for which any optimal solution has a large number of connected components (more than $\frac{1}{\epsilon^7}$). The idea is to start with a solution for $\mathrm{AR_F}^\infty$ ($\mathrm{AR_P}^\infty$, respectively), and appropriately divide each connected component of the solution into *chunks/subpaths*, of roughly ϵk points each. We show that there exists a solution, not much more expensive than the optimal one, that

contains all the edges within chunks/subpaths. Since in each such solution the number of edges between different chunks/subpaths is a small constant for each component, and each component has a total cost of at least 1 (due to the airport cost), we can connect the chunks/subpaths in a way that keeps the increase in cost within the limits.

Our algorithm for handling the dense subinstances consists of the following steps.

Step 1: Creating a grid. We partition the $\ell \times \ell$ bounding box into **cells**, by creating an ϵ^2-grid. Note that by Lemma 10 we have $\ell \leq 1/\epsilon$, and therefore the total number of cells is bounded from above by $1/\epsilon^6$.

Step 2: Splitting components. In this step we construct a solution $\textsc{Opt}'$ which is not necessarily feasible, but which will be needed for Step 3. The construction of $\textsc{Opt}'$ assumes that we know the exact number X of components in $\textsc{Opt}$. This is without the loss of generality, as we can run the algorithm for all choices of X (from 1 to n) and output the minimum cost solution obtained.

$\mathrm{AR_F}$: We run an exact algorithm for ${\mathrm{AR_F}}^{\infty}$ (see Theorem 4) with $\kappa = X$. This procedure returns a forest $F = \{T_1, T_2, \ldots, T_X\}$.

$\mathrm{AR_P}$: We run a PTAS for ${\mathrm{AR_P}}^{\infty}$ (see Theorem 2) with $\kappa = X$. This procedure returns a collection of X paths.

Note that in both cases $\mathrm{opt}' \leq (1+\epsilon)\mathrm{opt}$, by Theorems 2 and 4.

Step 3: Cutting into chunks/subpaths.

$\mathrm{AR_F}$: For each tree $T_i \in F$, we partition it into chunks by calling a procedure called $\textsc{GetChunks}(T_i)$, described below. As we will see in Lemma 11, each of the returned chunks will span least ϵk and at most $6\epsilon k$ vertices, except perhaps one smaller chunk per tree.

$\mathrm{AR_P}$: For each path of the solution returned by the PTAS used in Step 2, we split the path into subpaths of ϵk points each, except perhaps one shorter subpath per component.

Step 4: Associate chunks with cells.

$\mathrm{AR_F}$: We associate each chunk c_i with a cell, denoted by $\mathrm{cell}(c_i)$. We do this by selecting an arbitrary vertex $p \in c_i$, and associating c_i with the cell in which p lies.

$\mathrm{AR_P}$: We associate each of the two endpoints of each subpath with the cell that contains it.

Step 5: Assembling chunks/subpaths into a solution.

$\mathrm{AR_F}$: For each cell c we repeatedly collect arbitrary chunks associated with c to create a single connected component, until we obtain a component of size between $(1-6\epsilon)k$ and k, or until we run out of chunks. We connect these chunks into one connected component by adding edges within c, connecting any two vertices of the different chunks.

$\mathrm{AR_P}$: We start with a cell that contains an endpoint of a not yet considered subpath, follow the subpath to its other endpoint v and connect it to an endpoint v' of some other unconsidered subpath that is associated with $\mathrm{cell}(v)$ (if such a v' exists). We do this until either there is no other subpath starting at $\mathrm{cell}(v)$, or the current component has between $(1-\epsilon)k$ and k many vertices. We repeat the above as long as there are unconsidered subpaths in the instance.

We now describe the algorithm $\textsc{GetChunks}(T)$ (Algorithm 1) that splits a tree T into chunks. The *height* $h(v)$ of a vertex v is defined as the minimum number of edges on a path between v and a leaf (if v is a leaf then $h(v) = 0$). For some vertex v, let $\mathit{subtree}(v, T)$ be the (downward) subtree of T rooted at v, and let $\mathrm{childOf}(v, T)$ be the set of children of v in T. Processing the vertices of the tree bottom-up, the algorithm identifies an internal vertex v such that the subtree of T rooted at v contains at least ϵk and at most $6\epsilon k$ vertices. In

Algorithm 1: GetChunks(T)

Input: A tree T rooted at r
Output: A set of chunks comprising T

```
CH := ∅;
while T ≠ ∅ do
    chunk := Cut-A-Chunk(T);
    CH := CH ∪ {chunk};
    T := T \ chunk;
end
return CH
```

Algorithm 2: Cut-A-Chunk(T)

Input: A tree T rooted at r
Output: A subtree of T

```
for every vertex v of T do
    value(v) := 1
end
i := 0;
while |T| > 6ϵk do
    if there exists a vertex v with h(v) = i
     and value(v) ≥ ϵk then
        return subtree(v, T)
    else
        for every vertex v with h(v) = i + 1
         do
            value(v) :=
              ∑_{v'∈childOf(v,T)} value(v') + 1
        end
    end
    i := i + 1;
end
return T
```

Lemma 11 we will show that such a vertex v always exists (its existence can be derived by the fact that kissing number in the Euclidean plane is 6). The corresponding subtree rooted at v is then removed from T and the whole process is repeated. The algorithm ends when the whole tree T contains at most $6\epsilon k$ vertices, and this last chunk may contain less than ϵk many vertices.
We show the following property of the solutions output by the algorithm GetChunks(T).

▶ **Lemma 11.** *The algorithm* GetChunks(T) *outputs a partition of the tree T into chunks of size in the interval $[\epsilon k, 6\epsilon k]$, and possibly one smaller chunk.*

We are now ready to prove the following result.

▶ **Lemma 12.** *The algorithm presented in Steps* 1-5 *above is a PTAS for dense instances of* 1$\mathrm{AR_F}$ *and* 1$\mathrm{AR_P}$.

Proof. We first observe that all the steps of the algorithm run in polynomial time.

Denote by $\textsc{Opt}_F$ ($\textsc{Opt}_P$) an optimal solution for a given instance of $1\mathrm{AR_F}$ ($1\mathrm{AR_P}$, respectively). Let $\textsc{Alg}_F$ ($\textsc{Alg}_P$) be the solutions output by the algorithm, and $\textsc{Alg}'_F$ ($\textsc{Alg}'_P$) the intermediate, and possibly infeasible, solutions constructed in Step 2 of the algorithm for $1\mathrm{AR_F}$ ($1\mathrm{AR_P}$, respectively). When a statement applies to both $\mathrm{AR_P}$ and $\mathrm{AR_F}$ then we skip the subscript P and F. Recall that $\tilde{a}(S)$ and $r(S)$ refer to the airport cost (which is equal to the number of connected components of S), and the edge cost of a solution S.

First, consider Step 2 of the algorithm and the respective solutions $\textsc{Alg}'_F$ and $\textsc{Alg}'_P$. As $\textsc{Opt}_F$ ($\textsc{Opt}_P$) is a feasible solution for the ${\mathrm{AR_F}}^{\infty}$ (${\mathrm{AR_F}}^{\infty}$, respectively) problem, and in Step 2 the algorithm outputs an exact solution for ${\mathrm{AR_F}}^{\infty}$ (a $(1+\epsilon)$-approximate solution for

$\mathrm{AR_P}^{\infty}$, respectively), we obtain $\mathrm{alg}' \leq (1+\epsilon)\mathrm{opt}$. As we assumed that $\textsc{Alg}'$ has the same number of connected components as $\textsc{Opt}$ (i.e., $\tilde{a}(\textsc{Alg}') = \tilde{a}(\textsc{Opt})$), we get

$$r(\textsc{Alg}') \leq (1+\epsilon)r(\textsc{Opt}) + \epsilon\tilde{a}(\textsc{Opt}) \ . \tag{1}$$

In Step 3 the algorithm removes some edges from the solution $\textsc{Alg}'$, splitting the connected components into chunks (subpaths). We now turn our attention to Step 5 of the algorithm, i.e., reassembling the chunks (subpaths) into a solution for $1\mathrm{AR_F}$ ($1\mathrm{AR_P}$). In this step we repeatedly merge arbitrary chunks (subpaths) until we end up with a connected component of size between $(1-6\epsilon)k$ and k ($(1-\epsilon)k$ and k, respectively), or until we run out of chunks (subpaths) in the current cell c. Since we can only run out of chunks (subpaths) once per cell, we know that we can have at most $1/\epsilon^6$ components with a size less than $(1-6\epsilon)k$ in the final solution. As we are in the unit airport cost setting and we consider a dense problem instance, we have $\tilde{a}(\textsc{Opt}) > 1/\epsilon^7$, which gives us

$$\tilde{a}(\textsc{Alg}) \leq (1+O(\epsilon))\tilde{a}(\textsc{Opt}) \ . \tag{2}$$

By Lemma 11 and Step 5 of the algorithm, no component in the resulting solutions for $\mathrm{AR_F}$ and $\mathrm{AR_P}$ has size more than k. This, along with the structure of the constructed solutions (the assembled chunks form a tree, and the assembled subpaths form a path), implies the feasibility of the solution output by the algorithm.

The assembling in Step 5 is performed by adding at most one new edge per each chunk (subpath), and each added edge has cost of at most $\sqrt{2}\epsilon^2$. As each connected component of $\textsc{Alg}'$ has been split into at most $1/\epsilon$ chunks (subpaths), that gives us

$$r(\textsc{Alg}) \leq r(\textsc{Alg}') + \epsilon\sqrt{2}\tilde{a}(\textsc{Alg}') = r(\textsc{Alg}') + \epsilon\sqrt{2}\tilde{a}(\textsc{Opt}) \ . \tag{3}$$

The above inequalities imply the lemma, because

$$\begin{aligned}\mathrm{alg} = \tilde{a}(\textsc{Alg}) + r(\textsc{Alg}) &\leq (1+O(\epsilon))\tilde{a}(\textsc{Opt}) + r(\textsc{Alg}) \\ &\leq (1+O(\epsilon))\tilde{a}(\textsc{Opt}) + r(\textsc{Alg}') \leq (1+O(\epsilon))\mathrm{opt}.\end{aligned}$$

◀

4.4 A PTAS for $1\mathrm{AR_F}$ and $1\mathrm{AR_P}$

We will now show how the above results can be combined in order to prove Theorem 5.

Proof of Theorem 5. Given an arbitrary instance of $1\mathrm{AR_F}$ or $1\mathrm{AR_P}$ we first apply the preprocessing step (see Lemma 10). The problem instance is split into a collection of independent subinstances, such that each subinstance I is contained in a bounding box of size $1/\epsilon \times 1/\epsilon$. Each instance is either sparse or dense. Since we do not know which subinstances are sparse and which are dense (as this definition depends on the structure of an optimal solution for I), we will run both the algorithm for sparse instances (i.e., the PTAS from Section 4.2) and the algorithm for dense instances (i.e., the PTAS from Section 4.3) on each subinstance, and return the solution with lower cost for each subinstance. That will yield a PTAS for the original input instance. ◀

5 Open Problems

Two-dimensional Euclidean setting. In this paper we resolved the complexity of the $\mathrm{AR_P}^{\infty}, \mathrm{AR_F}^{\infty}, 1\mathrm{AR_F}$ and $1\mathrm{AR_P}$ problems in the two-dimensional Euclidean setting. It

would be interesting to know how well the general AR_P and AR_F problems can be approximated in this setting. Another open problem is generalizing the PTAS for $1AR_F$ and $1AR_P$ to work for more general families of airport costs. In particular, as stated earlier in this paper, a generalization of the PTAS for $1AR_P$ to a constant range of airport costs would yield a PTAS for Euclidean CVRP, which is an important open problem.

Other metrics. Another interesting question is getting positive and negative approximability results for the AR_P and AR_F problems in other metrics, for example for planar graphs, H-minor free graphs, or for metric graphs.

Other problems in the Airport and Railway framework. Another open question is to investigate other problems in the Airport and Railway framework introduced in our paper. It is not difficult to come up with several different specific requirements on the connected components, that model natural scenarios.

Acknowledgment. We could like to thank the anonymous reviewers for suggesting the use of an auxiliary vertex which simplified the proof of Theorem 4.

References

1 Anna Adamaszek, Artur Czumaj, and Andrzej Lingas. PTAS for k-tour cover problem on the plane for moderately large values of k. *Int. J. Found. Comput. Sci.*, 21(6):893–904, 2010. URL: `http://dx.doi.org/10.1142/S0129054110007623`, `doi:10.1142/S0129054110007623`.

2 Anna Adamaszek, Artur Czumaj, Andrzej Lingas, and Jakub Onufry Wojtaszczyk. Approximation schemes for capacitated geometric network design. In *Automata, Languages and Programming – 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, pages 25–36, 2011. URL: `http://dx.doi.org/10.1007/978-3-642-22006-7_3`, `doi:10.1007/978-3-642-22006-7_3`.

3 Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

4 Sanjeev Arora. Approximation schemes for NP-hard geometric optimization problems: a survey. *Math. Program.*, 97(1–2):43–69, 2003.

5 Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for Euclidean k-medians and related problems. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 106–113, 1998. URL: `http://doi.acm.org/10.1145/276698.276718`, `doi:10.1145/276698.276718`.

6 Tetsuo Asano, Naoki Katoh, Hisao Tamaki, and Takeshi Tokuyama. Covering points in the plane by k-tours: Towards a polynomial time approximation scheme for general k. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 275–283, 1997. URL: `http://doi.acm.org/10.1145/258533.258602`, `doi:10.1145/258533.258602`.

7 Marshall Wayne Bern and David Eppstein. Approximation algorithms for geometric problems. In Dorit Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, chapter 8, pages 296–345. PWS Publishing, 1996.

8 G.B. Dantzig and Ramser J.H. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

9 Aparna Das and Claire Mathieu. A quasipolynomial time approximation scheme for euclidean capacitated vehicle routing. *Algorithmica*, 73(1):115–142, 2015. URL: http://dx.doi.org/10.1007/s00453-014-9906-4, doi:10.1007/s00453-014-9906-4.

10 Friedrich Eisenbrand, Fabrizio Grandoni, Thomas Rothvoß, and Guido Schäfer. Connected facility location via random facility sampling and core detouring. *J. Comput. Syst. Sci.*, 76(8):709–726, 2010. URL: http://dx.doi.org/10.1016/j.jcss.2010.02.001, doi:10.1016/j.jcss.2010.02.001.

11 Inge Li Gørtz and Viswanath Nagarajan. Locating depots for capacitated vehicle routing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques – 14th International Workshop, APPROX 2011, and 15th International Workshop, RANDOM 2011, Princeton, NJ, USA, August 17-19, 2011. Proceedings*, pages 230–241, 2011. URL: http://dx.doi.org/10.1007/978-3-642-22935-0_20, doi:10.1007/978-3-642-22935-0_20.

12 Tobias Harks, Felix G. König, and Jannik Matuschke. Approximation algorithms for capacitated location routing. *Transportation Science*, 47(1):3–22, 2013. URL: http://dx.doi.org/10.1287/trsc.1120.0423, doi:10.1287/trsc.1120.0423.

13 Raja Jothi and Balaji Raghavachari. Approximation algorithms for the capacitated minimum spanning tree problem and its variants in network design. *ACM Transactions on Algorithms (TALG)*, 1(2):265–282, 2005.

14 Christos H Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

15 R. Ravi and Amitabh Sinha. Approximation algorithms for problems combining facility location and network design. *Operations Research*, 54(1):73–81, 2006. URL: http://dx.doi.org/10.1287/opre.1050.0228, doi:10.1287/opre.1050.0228.

16 Alexander Schrijver. *Combinatorial Optimization*. Springer, 2003.

17 Chaitanya Swamy and Amit Kumar. Primal-dual algorithms for connected facility location problems. *Algorithmica*, 40(4):245–269, 2004. URL: http://dx.doi.org/10.1007/s00453-004-1112-3, doi:10.1007/s00453-004-1112-3.

Simultaneous Feedback Vertex Set: A Parameterized Perspective*

Akanksha Agrawal[1], Daniel Lokshtanov[2], Amer E. Mouawad[3], and Saket Saurabh[4]

1 **Department of Informatics, University of Bergen, Bergen, Norway**
akanksha.agrawal@uib.no
2 **Department of Informatics, University of Bergen, Bergen, Norway**
daniel.lokshtanov@uib.no
3 **Department of Informatics, University of Bergen, Bergen, Norway**
a.mouawad@uib.no
4 **Department of Informatics, University of Bergen, Bergen, Norway; and Institute of Mathematical Sciences, Chennai, India**
saket@imsc.res.in

Abstract

For a family of graphs $\mathcal{F}$, a graph G, and a positive integer k, the $\mathcal{F}$-Deletion problem asks whether we can delete at most k vertices from G to obtain a graph in $\mathcal{F}$. $\mathcal{F}$-Deletion generalizes many classical graph problems such as Vertex Cover, Feedback Vertex Set, and Odd Cycle Transversal. A graph $G = (V, \cup_{i=1}^{\alpha} E_i)$, where the edge set of G is partitioned into α color classes, is called an α-edge-colored graph. A natural extension of the $\mathcal{F}$-Deletion problem to edge-colored graphs is the α-Simultaneous $\mathcal{F}$-Deletion problem. In the latter problem, we are given an α-edge-colored graph G and the goal is to find a set S of at most k vertices such that each graph $G_i \setminus S$, where $G_i = (V, E_i)$ and $1 \leq i \leq \alpha$, is in $\mathcal{F}$. In this work, we study α-Simultaneous $\mathcal{F}$-Deletion for $\mathcal{F}$ being the family of forests. In other words, we focus on the α-Simultaneous Feedback Vertex Set (α-SimFVS) problem. Algorithmically, we show that, like its classical counterpart, α-SimFVS parameterized by k is fixed-parameter tractable (FPT) and admits a polynomial kernel, for any fixed constant α. In particular, we give an algorithm running in $2^{\mathcal{O}(\alpha k)} n^{\mathcal{O}(1)}$ time and a kernel with $\mathcal{O}(\alpha k^{3(\alpha+1)})$ vertices. The running time of our algorithm implies that α-SimFVS is FPT even when $\alpha \in o(\log n)$. We complement this positive result by showing that for $\alpha \in \mathcal{O}(\log n)$, where n is the number of vertices in the input graph, α-SimFVS becomes W[1]-hard. Our positive results answer one of the open problems posed by Cai and Ye (MFCS 2014).

1998 ACM Subject Classification G.2.2 Graph Algorithms, I.1.2 Analysis of Algorithms

Keywords and phrases parameterized complexity, feedback vertex set, kernel, edge-colored graphs

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.7

1 Introduction

In graph theory, one can define a general family of problems as follows. Let $\mathcal{F}$ be a collection of graphs. Given an undirected graph G and a positive integer k, is it possible to perform at most k edit operations to G so that the resulting graph does not contain a graph from

* 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. 306992.

© Akanksha Agrawal, Daniel Lokshtanov, Amer E. Mouawad, and Saket Saurabh;
licensed under Creative Commons License CC-BY

$\mathcal{F}$? Here one can define edit operations as either vertex/edge deletions, edge additions, or edge contractions. Such problems constitute a large fraction of problems considered under the parameterized complexity framework. When edit operations are restricted to vertex deletions this corresponds to the $\mathcal{F}$-Deletion problem, which generalizes classical graph problems such as Vertex Cover [6], Feedback Vertex Set [5, 8, 18], Vertex Planarization [24], Odd Cycle Transversal [19, 21], Interval Vertex Deletion [4], Chordal Vertex Deletion [22], and Planar $\mathcal{F}$-Deletion [11, 17]. The topic of this paper is a generalization of $\mathcal{F}$-Deletion problems to "edge-colored graphs". In particular, we do a case study of an edge-colored version of the classical Feedback Vertex Set problem [12].

A graph $G = (V, \cup_{i=1}^{\alpha} E_i)$, where the edge set of G is partitioned into α color classes, is called an *α-edge-colored graph*. As stated by Cai and Ye [3], "edge-colored graphs are fundamental in graph theory and have been extensively studied in the literature, especially for alternating cycles, monochromatic sub-graphs, heterochromatic subgraphs, and partitions". A natural extension of the $\mathcal{F}$-Deletion problem to edge-colored graphs is the α-Simultaneous $\mathcal{F}$-Deletion problem. In the latter problem, we are given an α-edge-colored graph G and the goal is to find a set S of at most k vertices such that each graph $G_i \setminus S$, where $G_i = (V, E_i)$ and $1 \leq i \leq \alpha$, is in $\mathcal{F}$. Cai and Ye [3] studied several problems restricted to 2-edge-colored graphs, where edges are colored either red or blue. In particular, they consider the Dually Connected Induced Subgraph problem, i.e. find a set S of k vertices in G such that both induced graphs $G_{\text{red}}[S]$ and $G_{\text{blue}}[S]$ are connected, and the Dual Separator problem, i.e. delete a set S of at most k vertices to simultaneously disconnect the red and blue graphs of G. They show, among other results, that Dual Separator is NP-complete and Dually Connected Induced Subgraph is W[1]-hard even when both G_{red} and G_{blue} are trees. On the positive side, they prove that Dually Connected Induced Subgraph is solvable in time polynomial in the input size when G is a complete graph. One of the open problems they state is to determine the parameterized complexity of α-Simultaneous $\mathcal{F}$-Deletion for $\alpha = 2$ and $\mathcal{F}$ the family of forests, bipartite graphs, chordal graphs, or planar graphs. The focus in this work is on one of those problems, namely α-Simultaneous Feedback Vertex Set– an interesting, and well-motivated [2, 3, 16], generalization of Feedback Vertex Set on edge-colored graphs.

A *feedback vertex set* is a subset S of vertices such that $G \setminus S$ is a forest. For an α-colored graph G, an *α-simultaneous feedback vertex set* (or *α-simfvs* for short) is a subset S of vertices such that $G_i \setminus S$ is a forest for each $1 \leq i \leq \alpha$. The α-Simultaneous Feedback Vertex Set is stated formally as follows.

α-Simultaneous Feedback Vertex Set (α-SimFVS) **Parameter:** k
Input: (G, k), where G is an undirected α-colored graph and k is a positive integer
Question: Is there a subset $S \subseteq V(G)$ of size at most k such that for $1 \leq i \leq \alpha$, $G_i \setminus S$ is a forest?

Given a graph $G = (V, E)$ and a positive integer k, the classical Feedback Vertex Set (FVS) problem asks whether there exists a set S of at most k vertices in G such that the graph induced on $V(G) \setminus S$ is acyclic. In other words, the goal is to find a set of at most k vertices that intersects all cycles in G. FVS is a classical NP-complete [12] problem with numerous applications and is by now very well understood from both the classical and parameterized complexity [10] view points. For instance, the problem admits a 2-approximation algorithm [1], an exact (non-parameterized) algorithm running in $\mathcal{O}^\star(1.736^n)$ time [28], a deterministic algorithm running in $\mathcal{O}^\star(3.619^k)$ time [18], a randomized algorithm

running in $\mathcal{O}^\star(3^k)$ time [8], and a kernel on $\mathcal{O}(k^2)$ vertices [27] (see Section 2 for definitions). We use the $\mathcal{O}^\star$ notation to describe the running times of our algorithms. A running time $\mathcal{O}^\star(f(k))$ means that the running time is upper bounded by $f(k)n^{\mathcal{O}(1)}$, where n is the input size. That is, the $\mathcal{O}^\star$ notation suppresses polynomial factors in the running-time expression.

Our results and methods. We show that, like its classical counterpart, α-SimFVS parameterized by k is FPT and admits a polynomial kernel, for any fixed constant α. In particular, we obtain the following results.

- An FPT algorithm running in $\mathcal{O}^\star(23^{\alpha k})$ time. For the special case of $\alpha = 2$, we give a faster algorithm running in $\mathcal{O}^\star(81^k)$ time.
- For constant α, we obtain a kernel with $\mathcal{O}(\alpha k^{3(\alpha+1)})$ vertices.
- The running time of our algorithm implies that α-SimFVS is FPT even when $\alpha \in o(\log n)$. We complement this positive result by showing that for $\alpha \in \mathcal{O}(\log n)$, where n is the number of vertices in the input graph, α-SimFVS becomes W[1]-hard.

Our algorithms and kernel build on the tools and methods developed for FVS [7]. However, we need to develop both new branching rules as well as new reduction rules. The main reason why our results do not follow directly from earlier work on FVS is the following. Many (if not all) parameterized algorithms, as well as kernelization algorithms, developed for the FVS problem [7] exploit the fact that vertices of degree two or less in the input graph are, in some sense, irrelevant. In other words, vertices of degree one or zero cannot participate in any cycle and every cycle containing any degree-two vertex must contain both of its neighbors. Hence, if this degree-two vertex is part of a feedback vertex set then it can be replaced by either one of its neighbors. Unfortunately (or fortunately for us), this property does not hold for the α-SimFVS problem, even on graphs where edges are bicolored either red or blue. For instance, if a vertex is incident to two red edges and two blue edges, it might in fact be participating in two distinct cycles. Hence, it is not possible to neglect (or shortcut) this vertex in neither G_{red} nor G_{blue}. As we shall see, most of the new algorithmic techniques that we present deal with vertices of exactly this type. Although very tightly related to one another, we show that there are subtle and interesting differences separating the FVS problem from the α-SimFVS problem, even for $\alpha = 2$. For this reason, we also believe that studying α-Simultaneous $\mathcal{F}$-Deletion for different families of graphs $\mathcal{F}$, e.g. bipartite, chordal, or planar graphs, might reveal some new insights about the classical underlying problems.

In Section 3, we present an algorithm solving the α-SimFVS problem, parameterized by solution size k, in $\mathcal{O}^\star(23^{\alpha k})$ time. Our algorithm follows the iterative compression paradigm introduced by Reed et al. [26] combined with new reduction and branching rules. Our main new branching rule can be described as follows: Given a maximal degree-two path in some G_i, $1 \leq i \leq \alpha$, we branch depending on whether there is a vertex from this path participating in an α-simultaneous feedback vertex set or not. In the branch where we guess that a solution contains a vertex from this path, we construct a color i cycle which is isolated from the rest of the graph. In the other branch, we are able to follow known strategies by "simulating" the classical FVS problem. Observe that we can never have more than k isolated cycles of the same color. Hence, by incorporating this fact into our measure we are guaranteed to make "progress" in both branches. For the base case, each G_i is a disjoint union of cycles (though not G) and to find an α-simultaneous feedback vertex set for G we cast the remaining problem as an instance of Hitting Set parameterized by the size of the family. For $\alpha = 2$, we can instead use an algorithm for finding maximum matchings in an auxiliary graph. Using this fact we give a faster, $\mathcal{O}^\star(81^k)$ time, algorithm for the case $\alpha = 2$. In Section 4, we tackle

the question of kernelization and present a polynomial kernel for the problem, for constant α. Our kernel has $\mathcal{O}(\alpha k^{3(\alpha+1)})$ vertices and requires new insights into the possible structures induced by those special vertices discussed above. In particular, we enumerate all maximal degree-two paths in each G_i after deleting a feedback vertex set in G_i and study how such paths interact with each other. Using marking techniques, we are able to "unwind" long degree-two paths by making a private copy of each unmarked vertices for each color class. This unwinding leads to "normal" degree-two paths on which classical reduction rules can be applied and hence we obtain the desired kernel.

Finally, we consider the dependence between α and both the size of our kernel and the running time of our algorithm in Section 5. We show that even for $\alpha \in \mathcal{O}(\log n)$, where n is the number of vertices in the input graph, α-SimFVS becomes W[1]-hard. We show hardness via a new problem of independent interest which we denote by α-Partitioned Hitting Set. The input to this problem consists of a tuple $(\mathcal{U}, \mathcal{F} = \mathcal{F}_1 \cup \ldots \cup \mathcal{F}_\alpha, k)$, where $\mathcal{F}_i$, $1 \leq i \leq \alpha$, is a collection of subsets of the finite universe $\mathcal{U}$, k is a positive integer, and all the sets within a family $\mathcal{F}_i$, $1 \leq i \leq \alpha$, are pairwise disjoint. The goal is to determine whether there exists a subset X of $\mathcal{U}$ of cardinality at most k such that for every $f \in \mathcal{F} = \mathcal{F}_1 \cup \ldots \cup \mathcal{F}_\alpha$, $f \cap X$ is nonempty. We show that $\mathcal{O}(\log |\mathcal{U}||\mathcal{F}|)$-Partitioned Hitting Set is W[1]-hard via a reduction from Partitioned Subgraph Isomorphism and we show that $\mathcal{O}(\log n)$-SimFVS is W[1]-hard via a reduction from $\mathcal{O}(\log |\mathcal{U}||\mathcal{F}|)$-Partitioned Hitting Set. Along the way, we also show, using a somewhat simpler reduction from Hitting Set, that $\mathcal{O}(n)$-SimFVS is W[2]-hard.

Most of the technical details and proofs have been omitted from this extended abstract.

2 Preliminaries

We start with some basic definitions and introduce terminology from graph theory and algorithms. We also establish some of the notation that will be used throughout.

For a graph G, by $V(G)$ and $E(G)$ we denote its vertex set and edge set, respectively. We only consider finite graphs possibly having loops and multi-edges. In the following, let G be a graph and let H be a subgraph of G. By $d_H(v)$, we denote the degree of vertex v in H. For any non-empty subset $W \subseteq V(G)$, the subgraph of G induced by W is denoted by $G[W]$; its vertex set is W and its edge set consists of all those edges of E with both endpoints in W. For $W \subseteq V(G)$, by $G \setminus W$ we denote the graph obtained by deleting the vertices in W and all edges which are incident to at least one vertex in W.

A *path* in a graph is a sequence of distinct vertices $v_0, v_1, \ldots, v_k$ such that (v_i, v_{i+1}) is an edge for all $0 \leq i < k$. A *cycle* in a graph is a sequence of distinct vertices $v_0, v_1, \ldots, v_k$ such that $(v_i, v_{(i+1) \mod k})$ is an edge for all $0 \leq i \leq k$. We note that both a double edge and a loop are cycles. We also use the convention that a loop at a vertex v contributes 2 to the degree of v.

An edge α-colored graph is a graph $G = (V, \cup_{i=1}^{\alpha} E_i)$. We call G_i the color i (or i-color) graph of G, where $G_i = (V, E_i)$. For notational convenience we sometimes denote an α-colored graph as $G = (V, E_1, E_2, ..., E_\alpha)$. For an α-colored graph G, the *total degree* of a vertex v is $\sum_{i=1}^{\alpha} d_{G_i}(v)$. By color i edge (or i-color edge) we refer to an edge in E_i, for $1 \leq i \leq \alpha$. A vertex $v \in V(G)$ is said to have a color i neighbor if there is an edge (v, u) in E_i, furthermore u is a color i neighbor of v. We say a path or a cycle in G is *monochromatic* if all the edges on the path or cycle have the same color. Given a vertex $v \in V(G)$, a *v-flower* of order k is a set of k cycles in G whose pairwise intersection is exactly $\{v\}$. If all cycles in a v-flower are monochromatic then we have a *monochromatic v-flower*. An α-colored graph

$G = (V, E_1, E_2, \cdots, E_\alpha)$ is an *α-forest* if each G_i is a forest, for $1 \leq i \leq \alpha$. We refer the reader to [9] for details on standard graph theoretic notation and terminology we use in the paper.

3 FPT Algorithm for α-Simultaneous Feedback Vertex Set

We give an algorithm for the α-SimFVS problem using the method of iterative compression [26, 7]. We only describe the algorithm for the disjoint version of the problem. The existence of an algorithm running in $c^k \cdot n^{\mathcal{O}(1)}$ time for the disjoint variant implies that α-SimFVS can be solved in time $(1+c)^k \cdot n^{\mathcal{O}(1)}$ [7]. In the Disjoint α-SimFVS problem, we are given an α-colored graph $G = (V, E_1, E_2, \ldots, E_\alpha)$, an integer k, and an α-simfvs W in G of size $k+1$. The objective is to find an α-simfvs $X \subseteq V(G) \setminus W$ of size at most k, or correctly conclude the non-existence of such an α-simfvs.

3.1 Algorithm for Disjoint α-SimFVS

Let $(G = (V, E_1, E_2, \ldots, E_\alpha), W, k)$ be an instance of Disjoint α-SimFVS and let $F = G \setminus W$. We start with some simple reduction rules that clean up the graph. Whenever some reduction rule applies, we apply the lowest-numbered applicable rule.

- Reduction α-SimFVS.R1. Delete isolated vertices as they do not participate in any cycle.
- Reduction α-SimFVS.R2. If there is a vertex v which has only one neighbor u in G_i, for some $i \in \{1, 2, \ldots, \alpha\}$, then delete the edge (v, u) from E_i.
- Reduction α-SimFVS.R3. If there is a vertex $v \in V(G)$ with exactly two neighbors u, w (the total degree of v is 2), delete edges (v, u) and (v, w) from E_i and add an edge (u, w) to E_i, where i is the color of edges (v, u) and (v, w). Note that after reduction α-SimFVS.R2 has been applied, both edges (v, u) and (v, w) must be of the same color.
- Reduction α-SimFVS.R4. If for some i, $i \in \{1, 2, \ldots, \alpha\}$, there is an edge of multiplicity larger than 2 in E_i, reduce its multiplicity to 2.
- Reduction α-SimFVS.R5. If there is a vertex v with a self loop, then add v to the solution set X, delete v (and all edges incident on v) from the graph and decrease k by 1.

Note that all of the above reduction rules can be applied in polynomial time. Moreover, after exhaustively applying all rules, the resulting graph G satisfies the following properties:

(P1) G contains no loops,
(P2) Every edge in G_i, for $i \in \{1, 2, \ldots, \alpha\}$ is of multiplicity at most two.
(P3) Every vertex in G has either degree zero or degree at least two in each G_i, for $i \in \{1, 2, \ldots, \alpha\}$.
(P4) The total degree of every vertex in G is at least 3.

Algorithm. We give an algorithm for the decision version of the Disjoint α-SimFVS problem, which only verifies whether a solution exists or not. Such an algorithm can be easily modified to find an actual solution X. We follow a branching strategy with a nontrivial measure function. Let (G, W, k) be an instance of the problem, where G is an α-colored graph. If $G[W]$ is not an α-forest then we can safely return that (G, W, k) is a no-instance. Hence, we assume that $G[W]$ is an α-forest in what follows. Whenever any of our reduction rules α-SimFVS.R1 to α-SimFVS.R5 apply, the algorithm exhaustively does so (in order). If at any point in our algorithm the parameter k drops below zero, then the resulting instance is again a no-instance.

Recall that initially F is an α-forest, as W is an α-simfvs. We will consider each forest F_i, for $i \in \{1, 2, \ldots, \alpha\}$, separately (where F_i is the color i graph of the α-forest F). For $i \in \{1, 2, \ldots, \alpha\}$, we let $W_i = (W, E_i(G[W]))$ and η_i be the number of components in W_i. Some of the branching rules that we apply create special vertex-disjoint cycles. We will maintain this set of special cycles in $\mathcal{C}_i$, for each i, and we let $\mathfrak{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_\alpha\}$. Initially, $\mathcal{C}_i = \emptyset$, for each $i \in \{1, 2, \ldots, \alpha\}$. Each cycle that we add to $\mathcal{C}_i$ will be vertex disjoint from previously added cycles. Hence, if at any point $|\mathcal{C}_i| > k$, for any i, then we can stop exploring the corresponding branch. Moreover, whenever we "guess" that some vertex v must belong to a solution, we also traverse the family $\mathfrak{C}$ and remove any cycles containing v. For the running time analysis of our algorithm we will consider the following measure:

$$\mu = \mu(G, W, k, \mathfrak{C}) = \alpha k + (\sum_{i=1}^{\alpha} \eta_i) - (\sum_{i=1}^{\alpha} |\mathcal{C}_i|)\,.$$

The input to our algorithm consists of a tuple $(G, W, k, \mathfrak{C})$. For clarity, we will denote a reduced input by $(G, W, k, \mathfrak{C})$ (the one where reduction rules do not apply).

We root each tree in F_i at some arbitrary vertex. Assign an index t to each vertex v in the forest F_i, which is the distance of v from the root of the tree it belongs to (the root is assigned index zero). A vertex v in F_i is called *cordate* if one of the following holds:

- v is a leaf (or degree-zero vertex) in F_i with at least two color i neighbors in W_i.
- The subtree T_v^i rooted at v contains two vertices u and w which have at least one color i neighbor in W_i (v can be equal to u or w).

▶ **Lemma 1.** *For $i \in \{1, 2, \ldots, \alpha\}$, let v_c be a cordate vertex of highest index in some tree of the forest F_i and let $\mathcal{T}_{v_c}$ denote the subtree rooted at v_c. Furthermore, let u_c be one of the vertices in $\mathcal{T}_{v_c}$ such that u_c has a neighbor in W_i. Then, in the path $P = u_c, x_1, \ldots, x_t, v_c$ (t could be equal to zero) between u_c and v_c the vertices $x_1, \ldots, x_t$ are degree-two vertices in G_i.*

We consider the following cases depending on whether there is a cordate vertex in F_i or not.

Case 1: There is a cordate vertex in F_i. Let v_c be a cordate vertex with the highest index in some tree in F_i and let the two vertices with neighbors in W_i be u_c and w_c (v_c can be equal to u_c or w_c). Let $P = u_c, x_1, x_2, \cdots, x_t, v_c$ and $P' = v_c, y_1, y_2, \cdots, y_{t'}, w_c$ be the unique paths in F_i from u_c to v_c and from v_c to w_c, respectively. Let $P_v = u_c, x_1, \cdots, x_t, v_c, y_1, \cdots, y_{t'}, w_c$ be the unique path in F_i from u_c to w_c. Consider the following sub-cases:

Case 1.a: u_c and w_c have neighbors in the same component of W_i. In this case one of the vertices from path P_v must be in the solution. We branch as follows:

- v_c belongs to the solution. We delete v_c from G and decrease k by 1. In this branch μ decreases by α.

  When v_c does not belong to the solution, then at least one vertex from $u_c, x_1, x_2, \cdots, x_t$ or $y_1, y_2, \cdots, y_{t'}, w_c$ must be in the solution. But note that these are vertices of degree at most two in G_i by Lemma 1. So with respect to color i, it does not matter which vertex is chosen in the solution. The only issue comes from some color j cycle, where $j \neq i$, in which choosing a particular vertex from $u_c, x_1, \cdots, x_t$ or $y_1, y_2, \cdots, y_{t'}, w_c$ would be more beneficial. We consider the following two cases.
- One of the vertices from $u_c, x_1, x_2, \cdots, x_t$ is in the solution. In this case we add an edge (u_c, x_t) (or (u_c, u_c) when u_c and v_c are adjacent) to G_i and delete the edge (x_t, v_c) from G_i. This creates a cycle C in $G_i \setminus W$, which is itself a component in $G_i \setminus W$. We remove the edges in C from G_i and add the cycle C to $\mathcal{C}_i$. We will be

handling these sets of cycles independently. In this case $|\mathcal{C}_i|$ increases by 1, so the measure μ decreases by 1.

- One of the vertices from $y_1, y_2, \cdots, y_t, w_c$ is in the solution. In this case we add an edge (y_1, w_c) to G_i and delete the edge (v_c, y_1) from G_i. This creates a cycle C in $G_i \setminus W$ as a component. We add C to $\mathcal{C}_i$ and delete edges in C from $G_i \setminus W$. In this branch $|\mathcal{C}_i|$ increases by 1, so the measure μ decreases by 1.

  The resulting branching vector is $(\alpha, 1, 1)$.

Case 1.b: u_c and w_c do not have neighbors in the same component. We branch as follows:

- v_c belongs to the solution. We delete v_c from G and decrease k by 1. In this branch μ decreases by α.
- One of the vertices from $u_c, x_1, x_2, \cdots, x_t$ is in the solution. In this case we add an edge (u_c, x_t) to G_i and delete the edge (x_t, v_c) from G_i. This creates a cycle C in $G_i \setminus W$ as a component. As in Case 1, we add C to $\mathcal{C}_i$ and delete edges in C from $G_i \setminus W$. $|\mathcal{C}_i|$ increases by 1, so the measure μ decreases by 1.
- One of the vertices from $y_1, y_2, \cdots, y_t, w_c$ is in the solution. In this case we add an edge (y_1, w_c) to G_i and delete the edge (v_c, y_1) from G_i. This creates a cycle C in $G_i \setminus W$ as a component. We add C to $\mathcal{C}_i$ and delete edges in C from $G_i \setminus W$. In this branch $|\mathcal{C}_i|$ increases by 1, so the measure μ decreases by 1.
- No vertex from path P_v is in the solution. In this case we add the vertices in P_v to W, the resulting instance is $(G \setminus P_v, W \cup P_v, k)$. The number of components in W_i decreases and we get a drop of 1 in η_i, so μ decreases by 1. Note that if $G[W \cup P_v]$ is not acyclic we can safely ignore this branch.

  The resulting branching vector is $(\alpha, 1, 1, 1)$.

Case 2: There is no cordate vertex in F_i. Let $\mathcal{F}$ be a family of sets containing a set $f_C = V(C)$ for each $C \in \cup_{i=1}^{\alpha} \mathcal{C}_i$ and let $\mathcal{U} = \cup_{i=1}^{\alpha}(\cup_{C \in \mathcal{C}_i} V(C))$. Note that $|\mathcal{F}| \leq \alpha k$. We find a subset $U \subseteq \mathcal{U}$ (if it exists) which hits all the sets in $\mathcal{F}$, such that $|U| \leq k$.

Note that in Case 1, if the cordate vertex v_c is a leaf, then $u_c = w_c = v_c$. Therefore, from Case 1.a we are left with one branching rule. Similarly, we are left with the first and the last branching rules for Case 1.b. If v_c is not a leaf but v_c is equal to u_c or w_c, say $v_c = w_c$, then for both Case 1.a and Case 1.b we do not have to consider the third branch. Finally, when none of the reduction or branching rules apply, we solve the problem by invoking an algorithm for the Hitting Set problem as a subroutine.

▶ **Lemma 2.** Disjoint α-SimFVS *is solvable in time* $\mathcal{O}^\star(22^{\alpha k})$.

▶ **Theorem 3.** α-Simultaneous Feedback Vertex Set *is solvable in time* $\mathcal{O}^\star(23^{\alpha k})$.

4 Polynomial Kernel for α-Simultaneous Feedback Vertex Set

In this section we give a kernel with $\mathcal{O}(\alpha k^{3(\alpha+1)})$ vertices for α-SimFVS. Let (G, k) be an instance of α-SimFVS, where G is an α-colored graph and k is a positive integer. We assume that reduction rules α-SimFVS.R1 to α-SimFVS.R5 have been exhaustively applied. The kernelization algorithm then proceeds in two stages. In stage one, we bound the maximum degree of G. In the second stage, we present new reduction rules to deal with degree-two vertices and conclude a bound on the total number of vertices.

To bound the total degree of each vertex $v \in V(G)$, we bound the degree of v in G_i, for $i \in \{1, 2, \ldots, \alpha\}$. To do so, we need the Expansion Lemma [7] as well as the 2-approximation algorithm for the classical Feedback Vertex Set problem [1].

A *q-star*, $q \geq 1$, is a graph with $q+1$ vertices, one vertex of degree q and all other vertices of degree 1. Let G be a bipartite graph with vertex bipartition (A, B). A set of edges $M \subseteq E(G)$ is called a *q-expansion* of A into B if (i) every vertex of A is incident with exactly q edges of M and (ii) M saturates exactly $q|A|$ vertices in B.

▶ **Lemma 4** (Expansion Lemma [7]). *Let q be a positive integer and G be a bipartite graph with vertex bipartition (A, B) such that $|B| \geq q|A|$ and there are no isolated vertices in B. Then, there exist nonempty vertex sets $X \subseteq A$ and $Y \subseteq B$ such that:*
1. *X has a q-expansion into Y and*
2. *no vertex in Y has a neighbour outside X, i.e. $N(Y) \subseteq X$.*

Furthermore, the sets X and Y can be found in time polynomial in the size of G.

4.1 Bounding the Degree of Vertices in G_i

We now describe the reduction rules that allow us to bound the maximum degree of a vertex $v \in V(G)$. We make use of the following lemma which easily follows by adapting Lemma 6.8 from the work of Misra et al. [25].

▶ **Lemma 5.** *Let G be an undirected α-colored multi-graph and x be a vertex without a self loop in G_i, for $i \in \{1, 2, \ldots, \alpha\}$. Then in polynomial time we can either decide that (G, k) is a no-instance of* α-Simultaneous Feedback Vertex Set *or check whether there is an x-flower of order $k+1$ in G_i, or find a set of vertices $Z \subseteq V(G) \setminus \{x\}$ of size at most $3k$ intersecting every cycle in G_i.*

After applying reduction rules α-SimFVS.R1 to α-SimFVS.R5 exhaustively, we know that the degree of a vertex in each G_i is either 0 or at least 2 and no vertex has a self loop. Now consider a vertex v whose degree in G_i is more than $3k(k+4)$. By Proposition 5, we know that one of three cases must apply:
1. (G, k) is a no-instance of α-SimFVS,
2. we can find (in polynomial time) a v-flower of order $k+1$ in G_i, or
3. we can find (in polynomial time) a set $H_v \subseteq V(G_i)$ of size at most $3k$ such that $v \notin H_v$ and $G_i \setminus H_v$ is a forest.

The following reduction rule allows us to deal with case (2). The safeness of the rule follows from the fact that if v in not included in the solution then we need to have at least $k+1$ vertices in the solution.

Reduction α-SimFVS**.R6.** For $i \in \{1, 2, \ldots, \alpha\}$, if G_i has a vertex v such that there is a v-flower of order at least $k+1$ in G_i, then include v in the solution X and decrease k by 1. The resulting instance is $(G \setminus \{v\}, k-1)$.

When in case (3), we bound the degree of v as follows. Consider the graph $G_i' = G_i \setminus (H_v \cup \{v\} \cup V_0^i)$, where V_0^i is the set of degree 0 vertices in G_i. Let $\mathcal{D}$ be the set of components in the graph G_i' which have a vertex adjacent to v . Note that each $D \in \mathcal{D}$ is a tree and v cannot have two neighbors in D, since H_v is a feedback vertex set in G_i. We will now argue that each component $D \in \mathcal{D}$ has a vertex u such that u is adjacent to a vertex in H_v. Suppose for a contradiction that there is a component $D \in \mathcal{D}$ such that D has no vertex which is adjacent to a vertex in H_v. $D \cup \{v\}$ is a tree with at least 2 vertices, so D has a vertex w, such that w is a degree-one vertex in G_i, contradicting the fact that each vertex in G_i is either of degree zero or of degree at least two.

After exhaustive application of α-SimFVS.R4, every pair of vertices in G_i can have at most two edges between them. In particular, there can be at most two edges between $h \in H_v$

and v. If the degree of v in G_i is more than $3k(k+4)$, then the number of components $|\mathcal{D}|$, in G'_i is more than $3k(k+2)$, since $|H_v| \leq 3k$.

Consider the bipartite graph $\mathcal{B}$, with bipartition (H_v, Q), where Q has a vertex q_D corresponding to each component $D \in \mathcal{D}$. We add an edge between $h \in H_v$ and $q_D \in Q$ to $E(\mathcal{B})$ if and only if D has a vertex d which is adjacent to h in G_i.

Reduction α-SimFVS.**R7.** Let v be a vertex of degree at least $3k(k+4)$ in G_i, for $i \in \{1, 2, \ldots, \alpha\}$, and let H_v be a feedback vertex set in G_i not containing v and of size at most $3k$.

- Let $Q' \subseteq Q$ and $H \subseteq H_v$ be the sets of vertices obtained after applying Lemma 4 with $q = k+2$, $A = H_v$, and $B = Q$, such that H has a $(k+2)$-expansion into Q' in $\mathcal{B}$;
- Delete all the edges (d, v) in G_i, where $d \in V(D)$ and $q_D \in Q'$;
- Add double edges between v and h in G_i, for all $h \in H$ (unless such edges already exist).

After exhaustively applying all reductions α-SimFVS.R1 to α-SimFVS.R7, the degree of a vertex $v \in V(G_i)$ is at most $3k(k+4) - 1$ in G_i, for $i \in \{1, 2, \ldots, \alpha\}$.

4.2 Bounding the Number of Vertices in G

Having bounded the maximum total degree of a vertex in G, we now focus on bounding the number of vertices in the entire graph. To do so, we first compute an approximate solution for the α-SimFVS instance using the polynomial-time 2-approximation algorithm of Bafna et al. [1] for the Feedback Vertex Set problem in undirected graphs. In particular, we compute a 2-approximate solution S_i in G_i, for $i \in \{1, 2, \ldots, \alpha\}$. We let $S = \cup_{i=1}^{\alpha} S_i$. Note that S is an α-simfvs in G and has size at most $2\alpha|S_{OPT}|$, where $|S_{OPT}|$ is an optimal α-simfvs in G. Let $F_i = G_i \setminus S_i$. Let $T^i_{\leq 1}$, T^i_2, and $T^i_{\geq 3}$, be the sets of vertices in F_i having degree at most one in F_i, degree exactly two in F_i, and degree greater than two in F_i, respectively.

Later, we shall prove that bounding the maximum degree in G is sufficient for bounding the sizes of $T^i_{\leq 1}$ and $T^i_{\geq 3}$, for all $i \in \{1, 2, \ldots, \alpha\}$. We now focus on bounding the size of T^i_2 which, for each $i \in \{1, 2, \ldots, \alpha\}$, corresponds to a set of degree-two paths. In other words, for a fixed i, the graph induced by the vertices in T^i_2 is a set of vertex-disjoint paths. We say a set of distinct vertices $P = \{v_1, \ldots, v_\ell\}$ in T^i_2 forms a *degree-two path* if (v_j, v_{j+1}) is an edge, for all $1 \leq j \leq \ell$, and all vertices $\{v_1, \ldots, v_\ell\}$ have degree exactly two in G_i. We say P is a *maximal degree-two path* if no proper superset of P also forms a degree-two path.

We enumerate all the maximal degree-two paths in $G_i \setminus S_i$, for $i \in \{1, 2, \ldots, \alpha\}$. Let this set of paths in $G_i \setminus S_i$ be $\mathcal{P}_i = \{P^i_1, P^i_2, \ldots, P^i_{n_i}\}$, where n_i is the number of maximal degree-two paths in $G_i \setminus S_i$. We introduce a special symbol ϕ and add ϕ to each set $\mathcal{P}_i$, for $i \in \{1, 2, \ldots, \alpha\}$. The special symbol will be used later to indicate that no path is chosen from the set $\mathcal{P}_i$.

Let $\mathfrak{S} = \mathcal{P}_1 \times \mathcal{P}_2 \times \cdots \times \mathcal{P}_\alpha$ be the set of all tuples of maximal degree-two paths of different colors. For $\tau \in \mathfrak{S}$, $j \in \{1, 2, \ldots, \alpha\}$, $j(\tau)$ denotes the element from the set $\mathcal{P}_j$ in the tuple τ, i.e. for $\tau = (Q_1, \phi, \ldots, Q_j, \ldots, Q_\alpha)$, $j(\tau) = Q_j$ (for example $2(\tau) = \phi$).

For a maximal degree-two path $P^i_j \in \mathcal{P}_i$ and $\tau \in \mathfrak{S}$, we define $\mathit{Intercept}(P^i_j, \tau)$ to be the set of vertices in path P^i_j which are present in all the paths in the tuple (of course a ϕ entry does not contribute to this set). Formally, $\mathit{Intercept}(P^i_j, \tau) = \emptyset$ if $P^i_j \notin \tau$ otherwise $\mathit{Intercept}(P^i_j, \tau) = \{v \in V(P^i_j) |$ for all $1 \leq t \leq \alpha$, if $t(\tau) \neq \phi$ then $v \in V(t(\tau))\}$.

We define the notion of *unravelling* a path $P^i_j \in \mathcal{P}_i$ from all other paths of different colors in $\tau \in \mathfrak{S}$ at a vertex $u \in \mathit{Intercept}(P^i_j, \tau)$ by creating a separate copy of u for each path.

Formally, for a path $P_j^i \in \mathcal{P}_i$, $\tau \in \mathfrak{S}$, and a vertex $u \in Intercept(P_j^i, \tau)$, the $Unravel(P_j^i, \tau, u)$ operation does the following. For each $t \in \{1, 2, \ldots, \alpha\}$ let x_t and y_t be the unique neighbors of u on path $t(\tau)$. Create a vertex $u_{t(\tau)}$ for each path $t(\tau)$, for $1 \leq t \leq \alpha$, delete the edges (x_t, u) and (u, y_t) from G_t and add the edges $(x_t, u_{t(\tau)})$ and $(u_{t(\tau)}, y_t)$ in G_t.

Reduction α-SimFVS**.R8.** For a path $P_j^i \in \mathcal{P}_i$, $\tau \in \mathfrak{S}$, if $|Intercept(P_j^i, \tau)| > 1$, then for a vertex $u \in Intercept(P_j^i, \tau)$, $Unravel(P_j^i, \tau, u)$.

▶ **Theorem 6.** *α-SimFVS admits a kernel on $\mathcal{O}(\alpha k^{3(\alpha+1)})$ vertices.*

Proof. Consider an α-colored graph G on which reduction rules α-SimFVS.R1 to α-SimFVS.R8 have been exhaustively applied. For $i \in \{1, 2, \ldots, \alpha\}$, the degree of a vertex $v \in G_i$ is either 0 or at least 2 in G_i. Hence, in what follows, we do not count the vertices of degree 0 in G_i while counting the vertices in G_i; since the total degree of a vertex $v \in V(G)$ is at least three, there is some $j \in \{1, 2, \ldots, \alpha\}$ such that the degree of $v \in V(G_j)$ is at least 2.

Let S_i be a 2-approximate feedback vertex set in G_i, for $i \in \{1, 2, \ldots, \alpha\}$. Note that $S = \cup_{i=1}^{\alpha} S_i$ is a 2α-approximate α-simfvs in G. Let $F_i = G_i \setminus S_i$. Let $T_{\leq 1}^i$, T_2^i, and $T_{\geq 3}^i$, be the sets of vertices in F_i having degree at most one in F_i, degree exactly two in F_i, and degree greater than two in F_i, respectively.

The degree of each vertex $v \in V(G_i)$ is bounded by $\mathcal{O}(k^2)$ in G_i, for $i \in \{1, 2, \ldots, \alpha\}$. In particular, the degree of each $s \in S$ is bounded by $\mathcal{O}(k^2)$ in G_i. Moreover, each vertex $v \in T_{\leq 1}^i$ has degree at least 2 in G_i and must therefore be adjacent to some vertex in S. It follows that $|T_{\leq 1}^i| \in \mathcal{O}(k^3)$.

In a tree, the number t of vertices of degree at least three is bounded by $l - 2$, where l is the number of leaves. Hence, $|T_{\geq 3}^i| \in \mathcal{O}(k^3)$. Also, in a tree, the number of maximal degree-two paths is bounded by $t + l$. Consequently, the number of degree-two paths in $G_i \setminus S_i$ is in $\mathcal{O}(k^3)$. Moreover, no two maximal degree-two paths in a tree intersect.

Note that there are at most $\mathcal{O}(k^3)$ maximal degree-two paths in $\mathcal{P}_i$, for $i \in \{1, 2, \ldots, \alpha\}$, and therefore $|\mathfrak{S}| = \mathcal{O}(k^{3\alpha})$. After exhaustive application of α-SimFVS.R8, for each path $P_j^i \in \mathcal{P}_i$, $i \in \{1, 2, \ldots, \alpha\}$, and $\tau \in \mathfrak{S}$, there is at most one vertex in $Intercept(P_j^i, \tau)$. Also note that after exhaustive application of reductions α-SimFVS.R1 to α-SimFVS.R7, the total degree of a vertex in G is at least 3. Therefore, there can be at most $\mathcal{O}(k^{3\alpha})$ vertices in a degree-two path $P_j^i \in \mathcal{P}_i$. Furthermore, there are at most $\mathcal{O}(k^3)$ degree-two maximal paths in G_i, for $i \in \{1, 2, \ldots, \alpha\}$. It follows that $|T_2^i| \in \mathcal{O}(k^{3(\alpha+1)})$ and $|V(G_i)| \leq |T_{\leq 1}^i| + |T_2^i| + |T_{\geq 3}^i| + |S_i| = \mathcal{O}(k^3) + \mathcal{O}(k^{3(\alpha+1)}) + \mathcal{O}(k^3) + 2k \in \mathcal{O}(k^{3(\alpha+1)})$. Therefore, the number of vertices in G is in $\mathcal{O}(\alpha k^{3(\alpha+1)})$. ◀

5 Hardness Results

In this section we show that $\mathcal{O}(\log n)$-SimFVS, where n is the number of vertices in the input graph, is W[1]-hard. We give a reduction from a special version of the Hitting Set (HS) problem, which we denote by α-Partitioned Hitting Set (α-PHS). We believe this version of Hitting Set to be of independent interest with possible applications for showing hardness results of similar flavor. We prove W[1]-hardness of α-Partitioned Hitting Set by a reduction from a restricted version of the Partitioned Subgraph Isomorphism (PSI) problem.

Before we delve into the details, we start with a simpler reduction from Hitting Set showing that $\mathcal{O}(n)$-SimFVS parameterized by solution size is W[2]-hard . The reduction

closely follows that of Lokshtanov [20] for dealing with the Wheel-Free Deletion problem. Intuitively, starting with an instance $(\mathcal{U}, \mathcal{F}, k)$ of HS, we first construct a graph G on $2|\mathcal{U}||\mathcal{F}|$ vertices consisting of $|\mathcal{F}|$ vertex-disjoint cycles. Then, we use $|\mathcal{F}|$ colors to uniquely map each set to a separate cycle; carefully connecting these cycles together guarantees equivalence of both instances.

▶ **Theorem 7.** *$\mathcal{O}(n)$-SimFVS parameterized by solution size is W[2]-hard.*

Notice that if we assume that $|\mathcal{U}|$ and $|\mathcal{F}|$ are linearly dependent, then Theorem 7 in fact shows that $\mathcal{O}(\sqrt{n})$-SimFVS is W[2]-hard. However, the construction of Theorem 7 crucially relies on the fact that each cycle is "uniquely identified" by a separate color. In order to get around this limitation and prove W[1]-hardness of $\mathcal{O}(\log n)$-SimFVS we need, in some sense, to group separate sets of a Hitting Set instance into $\mathcal{O}(\log(|\mathcal{U}||\mathcal{F}|))$ families such that sets inside each family are pairwise disjoint. By doing so, we can modify the reduction of Theorem 7 to identify all sets inside a family using the same color, for a total of $\mathcal{O}(\log n)$ colors (instead of $\mathcal{O}(n)$ or $\mathcal{O}(\sqrt{n})$). We achieve exactly this in what follows. We refer the reader to the work of Impagliazzo et al. [14, 15] for details on the Exponential Time Hypothesis (ETH).

α-Partitioned Hitting Set **Parameter:** k
Input: A tuple $(\mathcal{U}, \mathcal{F} = \mathcal{F}_1 \cup \ldots \cup \mathcal{F}_\alpha, k)$, where $\mathcal{F}_i$, $1 \leq i \leq \alpha$, is a collection of subsets of the finite universe $\mathcal{U}$ and k is a positive integer. Moreover, all the sets within a family $\mathcal{F}_i$, $1 \leq i \leq \alpha$, are pairwise disjoint.
Question: Is there a subset X of $\mathcal{U}$ of cardinality at most k such that for every $f \in \mathcal{F} = \mathcal{F}_1 \cup \ldots \cup \mathcal{F}_\alpha$, $f \cap X$ is nonempty?

Partitioned Subgraph Isomorphism **Parameter:** $k = |E(G)|$
Input: A graph H, a graph G with $V(G) = \{g_1, \ldots, g_\ell\}$, and a coloring function $col : V(H) \to [\ell]$.
Question: Is there an injection $inj : V(G) \to V(H)$ such that for every $i \in [\ell]$, $col(inj(g_i)) = i$ and for every $(g_i, g_j) \in E(G)$, $(inj(g_i), inj(g_j)) \in E(H)$?

▶ **Theorem 8** ([13, 23])**.** Partitioned Subgraph Isomorphism *parameterized by $|E(G)|$ is W[1]-hard, even when the maximum degree of the smaller graph G is three. Moreover, the problem cannot be solved in time $f(k)n^{o(\frac{k}{\log k})}$, where f is an arbitrary function, $n = |V(H)|$, and $k = |E(G)|$, unless ETH fails.*

▶ **Theorem 9.** *$\mathcal{O}(\log(|\mathcal{U}||\mathcal{F}|))$-Partitioned Hitting Set parameterized by solution size is W[1]-hard. Moreover, the problem cannot be solved in time $f(k)n^{o(\frac{k}{\log k})}$, where f is an arbitrary function, $n = |\mathcal{U}|$, and k is the required solution size, unless ETH fails.*

We are now ready to state the main result of this section. The proof of Theorem 10 follows the same steps as the proof of Theorem 7 with one exception, i.e we reduce from $\mathcal{O}(\log(|\mathcal{U}||\mathcal{F}|))$-Partitioned Hitting Set and use $\mathcal{O}(\log(|\mathcal{U}||\mathcal{F}|))$ colors instead of $|\mathcal{F}|$.

▶ **Theorem 10.** *$\mathcal{O}(\log n)$-SimFVS parameterized by solution size is W[1]-hard.*

Proof. Given an instance $(\mathcal{U}, \mathcal{F} = \mathcal{F}_1 \cup \ldots \cup \mathcal{F}_\alpha, k)$ of α-PHS, we let $\mathcal{U} = \{u_1, \ldots, u_{|\mathcal{U}|}\}$ and $\mathcal{F}_i = \{f_1^i, \ldots, f_{|\mathcal{F}_i|}^i\}$, $1 \leq i \leq \alpha$. We assume, without loss of generality, that each element in $\mathcal{U}$ belongs to at least one set in $\mathcal{F}$.

For each $f_j^i \in \mathcal{F}_i$, $1 \leq i \leq \alpha$ and $1 \leq j \leq |\mathcal{F}_i|$, we create a vertex-disjoint cycle C_j^i on $2|\mathcal{U}|$ vertices and assign all its edges color i. We let $V(C_j^i) = \{c_1^{i,j}, \ldots, c_{2|\mathcal{U}|}^{i,j}\}$ and we define $\beta(i, j, u_p) = c_{2p-1}^{i,j}$, $1 \leq i \leq \alpha$, $1 \leq j \leq |\mathcal{F}_i|$, and $1 \leq p \leq |\mathcal{U}|$. In other words, every odd-numbered vertex of C_j^i is mapped to an element in $\mathcal{U}$. Now for every element $u_p \in \mathcal{U}$, $1 \leq p \leq |\mathcal{U}|$, we create a vertex v_p, we let $\gamma(u_p) = \{c_{2p-1}^{i,j} | 1 \leq i \leq \alpha \wedge 1 \leq j \leq |\mathcal{F}_i| \wedge u_p \in f_j^i\}$, and we add an edge (of some special color, say 0) between v_p and every vertex in $\gamma(u_p)$. To finalize the reduction, we contract all the edges colored 0 to obtain an instance (G, k) of $\mathcal{O}(\log n)$-SimFVS. Note that $|V(G)| = |E(G)| = 2|\mathcal{U}||\mathcal{F}|$ and the total number of used colors is α. Moreover, after contracting all special edges, $|\gamma(u_p)| = 1$ for all $u_p \in \mathcal{U}$.

▶ **Claim 1.** *If $\mathcal{F}$ admits a hitting set of size at most k then G admits an α-simfvs of size at most k.*

Proof. Let $X = \{u_{p_1}, \ldots, u_{p_k}\}$ be such a hitting set. We construct a vertex set $Y = \{\gamma(u_{p_1}), \ldots, \gamma(u_{p_k})\}$. If Y is not an α-simfvs of G then $G[V(G) \setminus Y]$ must contain some monochromatic cycle. By construction, only sets from the same family $\mathcal{F}_i$, $1 \leq i \leq \alpha$, correspond to cycles assigned the same color in G. But since we started with an instance of α-PHS, no two such sets intersect. Hence, the contraction operations applied to obtain G cannot create new monochromatic cycles. Therefore, if $G[V(G) \setminus Y]$ contains some monochromatic cycle then X cannot be a hitting set of $\mathcal{F}$. ◀

▶ **Claim 2.** *If G admits an α-simfvs of size at most k then $\mathcal{F}$ admits a hitting set of size at most k.*

Proof. Let $X = \{v_{p_1}, \ldots, v_{p_k}\}$ be such an α-simfvs. First, note that if some vertex in X does not correspond to an element in $\mathcal{U}$, then we can safely replace that vertex with one that does (since any such vertex belongs to exactly one monochromatic cycle). We construct a set $Y = \{u_{p_1}, \ldots, u_{p_k}\}$. If there exists a set $f_j^i \in \mathcal{F}_i$ such that $Y \cap f_j^i = \emptyset$ then, by construction, there exists an i-colored cycle C_i in G such that $X \cap V(C_i) = \emptyset$, a contradiction. ◀

Combining the previous two claims with the fact that our reduction runs in time polynomial in $|\mathcal{U}|$, $|\mathcal{F}|$, and k, completes the proof of the theorem. ◀

6 Conclusion

We have showed that α-SimFVS parameterized by solution size k is fixed-parameter tractable and can be solved by an algorithm running in $\mathcal{O}^\star(23^{\alpha k})$ time, for any constant α. For the special case of $\alpha = 2$, we gave a faster $\mathcal{O}^\star(81^k)$ time algorithm which follows from the observation that the base case of the general algorithm can be solved in polynomial time when $\alpha = 2$. Moreover, for constant α, we presented a kernel for the problem with $\mathcal{O}(\alpha k^{3(\alpha+1)})$ vertices.

It is interesting to note that our algorithm implies that α-SimFVS can be solved in $(2^{\mathcal{O}(\alpha)})^k n^{\mathcal{O}(1)}$ time. However, we have also seen that α-SimFVS becomes W[1]-hard when $\alpha \in \mathcal{O}(\log n)$. This implies that (under plausible complexity assumptions) an algorithm running in $(2^{o(\alpha)})^k n^{\mathcal{O}(1)}$ time cannot exist. In other words, the running time cannot be subexponential in either k or α.

As mentioned by Cai and Ye [3], we believe that studying generalizations of other classical problems to edge-colored graphs is well motivated and might lead to interesting new insights about combinatorial and structural properties of such problems. Some of the potential candidates are Vertex Planarization, Odd Cycle Transversal, Interval Vertex

Deletion, Chordal Vertex Deletion, Planar $\mathcal{F}$-Deletion, and, more generally, α-Simultaneous $\mathcal{F}$-Deletion.

References

1. Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discret. Math.*, 12(3):289–297, September 1999. URL: http://dx.doi.org/10.1137/S0895480196305124, doi:10.1137/S0895480196305124.
2. Jorgen Bang-Jensen and Gregory Gutin. Alternating cycles and paths in edge-coloured multigraphs: A surve. *Discrete Mathematics*, 165–166:39–60, 1997. Graphs and Combinatorics.
3. Leizhen Cai and Junjie Ye. Dual connectedness of edge-bicolored graphs and beyond. In *Mathematical Foundations of Computer Science 2014*, volume 8635 of *Lecture Notes in Computer Science*, pages 141–152. Springer Berlin Heidelberg, 2014. URL: http://dx.doi.org/10.1007/978-3-662-44465-8_13.
4. Yixin Cao and Dániel Marx. Interval deletion is fixed-parameter tractable. *ACM Trans. Algorithms*, 11(3):21:1–21:35, January 2015. URL: http://doi.acm.org/10.1145/2629595, doi:10.1145/2629595.
5. Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *Journal of Computer and System Sciences*, 74(7):1188–1198, 2008. URL: http://www.sciencedirect.com/science/article/pii/S0022000008000500, doi:http://dx.doi.org/10.1016/j.jcss.2008.05.002.
6. Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, September 2010. URL: http://dx.doi.org/10.1016/j.tcs.2010.06.026, doi:10.1016/j.tcs.2010.06.026.
7. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. URL: http://dx.doi.org/10.1007/978-3-319-21275-3, doi:10.1007/978-3-319-21275-3.
8. Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Joham M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS'11, pages 150–159, Washington, DC, USA, 2011. IEEE Computer Society. URL: http://dx.doi.org/10.1109/FOCS.2011.23, doi:10.1109/FOCS.2011.23.
9. Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
10. Rod G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer-Verlag, 1997.
11. Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-Deletion: Approximation, kernelization and optimal FPT algorithms. In *FOCS*, pages 470–479. IEEE Computer Society, 2012. URL: http://dblp.uni-trier.de/db/conf/focs/focs2012.html#FominLMS12.
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. Martin Grohe and Dániel Marx. On tree width, bramble size, and expansion. *Journal of Combinatorial Theory, Series B*, 99(1):218–228, 2009. URL: http://www.sciencedirect.com/science/article/pii/S0095895608000683, doi:http://dx.doi.org/10.1016/j.jctb.2008.06.004.
14. Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. URL: http://www.

sciencedirect.com/science/article/pii/S0022000000917276, doi:http://dx.doi.org/10.1006/jcss.2000.1727.

15 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. URL: http://www.sciencedirect.com/science/article/pii/S002200000191774X, doi:http://dx.doi.org/10.1006/jcss.2001.1774.

16 Mikio Kano and Xueliang Li. Monochromatic and heterochromatic subgraphs in edge-colored graphs – a survey. *Graphs and Combinatorics*, 24(4):237–263, 2008.

17 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*, volume 7965 of *Lecture Notes in Computer Science*, pages 613–624. Springer Berlin Heidelberg, 2013. URL: http://dx.doi.org/10.1007/978-3-642-39206-1_52.

18 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Information Processing Letters*, 114(10):556–560, 2014. URL: http://www.sciencedirect.com/science/article/pii/S0020019014000969, doi:http://dx.doi.org/10.1016/j.ipl.2014.05.001.

19 Stefan Kratsch and Magnus Wahlstrom. Representative sets and irrelevant vertices: New tools for kernelization. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, 0:450–459, 2012. doi:http://doi.ieeecomputersociety.org/10.1109/FOCS.2012.46.

20 Daniel Lokshtanov. Wheel-free deletion is W[2]-Hard. In Martin Grohe and Rolf Niedermeier, editors, *Parameterized and Exact Computation*, volume 5018 of *Lecture Notes in Computer Science*, pages 141–147. Springer Berlin Heidelberg, 2008. URL: http://dx.doi.org/10.1007/978-3-540-79723-4_14, doi:10.1007/978-3-540-79723-4_14.

21 Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Trans. Algorithms*, 11(2):15:1–15:31, October 2014. URL: http://doi.acm.org/10.1145/2566616, doi:10.1145/2566616.

22 Dániel Marx. Chordal deletion is fixed-parameter tractable. In FedorV. Fomin, editor, *Graph-Theoretic Concepts in Computer Science*, volume 4271 of *Lecture Notes in Computer Science*, pages 37–48. Springer Berlin Heidelberg, 2006. URL: http://dx.doi.org/10.1007/11917496_4, doi:10.1007/11917496_4.

23 Dániel Marx. Can you beat treewidth? In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, FOCS'07, pages 169–179, Washington, DC, USA, 2007. IEEE Computer Society. URL: http://dx.doi.org/10.1109/FOCS.2007.18, doi:10.1109/FOCS.2007.18.

24 Dániel Marx and Ildiko Schlotter. Obtaining a planar graph by vertex deletion. In *Graph-Theoretic Concepts in Computer Science*, volume 4769 of *Lecture Notes in Computer Science*, pages 292–303. Springer Berlin Heidelberg, 2007. URL: http://dx.doi.org/10.1007/978-3-540-74839-7_28.

25 Pranabendu Misra, Venkatesh Raman, M.S. Ramanujan, and Saket Saurabh. Parameterized algorithms for even cycle transversal. In MartinCharles Golumbic, Michal Stern, Avivit Levy, and Gila Morgenstern, editors, *Graph-Theoretic Concepts in Computer Science*, volume 7551 of *Lecture Notes in Computer Science*, pages 172–183. Springer Berlin Heidelberg, 2012. URL: http://dx.doi.org/10.1007/978-3-642-34611-8_19, doi:10.1007/978-3-642-34611-8_19.

26 Bruce Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004. URL: http://www.sciencedirect.com/science/

article/pii/S0167637703001482, doi:http://dx.doi.org/10.1016/j.orl.2003.10.009.

27 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2):32:1–32:8, April 2010. URL: http://doi.acm.org/10.1145/1721837.1721848, doi:10.1145/1721837.1721848.

28 Mingyu Xiao and Hiroshi Nagamochi. An improved exact algorithm for undirected feedback vertex set. In Peter Widmayer, Yinfeng Xu, and Binhai Zhu, editors, *Combinatorial Optimization and Applications*, volume 8287 of *Lecture Notes in Computer Science*, pages 153–164. Springer International Publishing, 2013. URL: http://dx.doi.org/10.1007/978-3-319-03780-6_14, doi:10.1007/978-3-319-03780-6_14.

On Regularity of Unary Probabilistic Automata

S. Akshay[1], Blaise Genest[2], Bruno Karelovic[3], and Nikhil Vyas[4]

1 **Department of Computer Science and Engineering, IIT Bombay, India**
akshayss@cse.iitb.ac.in
2 **CNRS, IRISA, Rennes, France**
blaise.genest@irisa.fr
3 **LIAFA, Université Paris 7, France**
bruno.karelovic@gmail.com
4 **Department of Computer Science and Engineering, IIT Bombay, India**
nikhilvyas@cse.iitb.ac.in

Abstract

The quantitative verification of Probabilistic Automata (PA) is undecidable in general. Unary PA are a simpler model where the choice of action is fixed. Still, the quantitative verification problem is open and known to be as hard as Skolem's problem, a problem on linear recurrence sequences, whose decidability is open for at least 40 years. In this paper, we approach this problem by studying the languages generated by unary PAs (as defined below), whose regularity would entail the decidability of quantitative verification.

Given an initial distribution, we represent the trajectory of a unary PA over time as an infinite word over a finite alphabet, where the n^{th} letter represents a probability range after n steps. We extend this to a language of trajectories (a set of words), one trajectory for each initial distribution from a (possibly infinite) set. We show that if the eigenvalues of the transition matrix associated with the unary PA are all distinct positive real numbers, then the language is *effectively regular*. Further, we show that this result is at the boundary of regularity, as non-regular languages can be generated when the restrictions are even slightly relaxed. The regular representation of the language allows us to reason about more general properties, e.g., robustness of a regular property in a neighbourhood around a given distribution.

1998 ACM Subject Classification F.1.2 Modes of Computation: Probabilistic computation

Keywords and phrases Probabilistic automata, Symbolic dynamics, Markov chains, Skolem problem, Regularity

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.8

1 Introduction

Markov decision processes (MDPs for short) are a standard model for describing probabilistic systems with nondeterminism. The system or controller has a strategy according to which it chooses an action at every step, which is then performed according to a probability distribution defined over the set of possible resultant states. The usual question is whether some property (e.g. reaching a set of *Goal* states) can be achieved with probability at least some threshold γ.

In many interesting settings, the controller cannot observe the state in which it operates or only has partial information regarding the state (Partially Observable MDPs, POMDPs). Probabilistic automata (PAs for short) [21, 20] form the subclass of POMDPs where the controller cannot observe anything. The problem of whether there is a strategy to reach *Goal* with probability at least a threshold γ (also called a cut-point) is already undecidable [5].


© S. Akshay, Blaise Genest, Bruno Karelovic, and Nikhil Vyas;
licensed under Creative Commons License CC-BY
33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 8; pp. 8:1–8:14

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

Even approximating this probability has been shown undecidable in PAs [13]. In fact, deciding whether there exists a sequence of strategies with probability arbitrarily close to $\gamma = 1$ is already undecidable [9], and only very restricted subclasses are known to ensure decidability [8, 7].

A line of work, which we follow, is to consider unary PAs [6, 22], where the alphabet has a single letter. That is, there is a unique strategy, and the model is essentially a Markov chain. Surprisingly, the 'simple' problem of whether there exists a finite number of steps after which the probability to be in *Goal* is higher than the threshold $\gamma \in (0, 1)$ is open and has recently been shown [3] to be as hard as the so-called *Skolem's problem*, which is a long-standing open problem on linear recurrence sequences [14, 12, 16]. One way to tackle the problem is to approximate it, asking whether for all ϵ there exists a number of steps n_ϵ after which the probability to be in *Goal* is at least $\gamma - \epsilon$. The decidability and precise complexity of this problem has been explored in [6]. A more general approximation scheme, valid for much more general questions which can be expressed in some LTL logic, has also been tackled by generating a regular language of *approximated* behaviors [1].

In this paper, we study classes for which the language of *exact* behaviors is (ω-)regular, allowing for the exact resolution of any regular question (e. g. checking any $\mathrm{LTL}_{\mathcal{I}}$ formula [1, 2]). We define the trajectory from a given initial distribution as an (infinite) word over the alphabet $\{A, B\}$. The n^{th} letter of a trajectory being A (for Above, respectively, B for Below) represents that after n steps the probability to be in *Goal* is greater than or equal to (respectively lesser than) the threshold γ. Further, we consider the language of a unary PA as the set of trajectories (words) ranging over a (possibly infinite) set of initial distributions. Thus, we can answer questions such as: does there exist a trajectory from the set of initial distributions satisfying a regular property or do all trajectories satisfy it. We can also tackle more complicated questions such as robustness wrt. a given initial distribution δ_{init}: does a regular property hold for all initial distributions "around" δ_{init}.

As motivation, consider a population of yeast under osmotic stress [15]. The stress level of the population can be studied through a protein which can be marked (by a chemical reagent). For the sake of illustration, consider the following simplistic model of a Markov Chain M_{yeast} with the protein being in 3 different discrete states (namely the concentration of the protein being high (state 1), medium (state 2) and low (state 3)). The transition matrix, also denoted M_{yeast}, gives the proportion of yeast moving from one protein concentration level to another one, in one time step (say, 15 seconds):

$$M_{yeast} = \begin{pmatrix} 0.8 & 0.1 & 0.2 \\ 0.1 & 0.8 & 0.1 \\ 0.1 & 0.1 & 0.7 \end{pmatrix}$$

For instance, 20% of the yeast with low protein concentration will have high protein concentration at the next time step. The marker can be observed optically when the concentration of the protein is high. We know that the original proportion of yeast in state 1 is 1/3 (by counting the marked yeast population), but we are unsure of the mix between low and medium. The initial set of distributions is thus $Init_{yeast} = \{(1/3, x, 2/3 - x) \mid 0 \leq x \leq 2/3\}$. The language of M_{yeast} will tell us how the population evolves wrt the number of marked yeast being above or below the threshold $\gamma_{yeast} = 5/12$, depending on the initial distribution in $Init_{yeast}$. Now, suppose an experiment with yeasts reveals that there are at first less than 5/12 of *marked* yeast (i.e. with high concentration of proteins), then more than 5/12 of marked yeast, and eventually less than 5/12 of marked yeasts. That is, the trajectory is B for a while, then A for a while, then it stabilises at B. Let us call this property as (P_{yeast})

Table 1 A summary of the results in this paper.

Property of eigenvalues of Markov chain	Regular language	Ultimately periodic traj.
Distinct, positive real numbers	✓ (Thm.4)	✓ (from below)
Distinct, roots of real numbers	× (Thm.10)	✓ (Prop.1)
Distinct	× (from above)	× ([2], Thm.3)

(note that this is a regular property). We are interested in checking whether our simplistic model exhibits at least one trajectory with the property (P_{yeast}), and if yes, the range of initial values generating trajectories with this property.

Our contributions as depicted in table 1 are the following: if the eigenvalues of the transition (row-stochastic) matrix associated with the unary PA are distinct roots of real numbers, then any trajectory from a given initial distribution is ultimately periodic. This is tight, in the sense that, there are examples of trajectories which are not ultimately periodic even for unary PAs with 3 states [2, 22] (with some eigenvalue not root of any real number). Our main result is that if, further, the eigenvalues are distinct positive real numbers, then the language generated by a unary PA starting from a convex polytope of initial distributions is *effectively regular*. Surprisingly, this result is also tight: there exist unary PA with eigenvalues being distinct roots of real numbers (starting from a convex initial set) which generate a non-regular language, as we show in Section 6. Due to space constraints, we only present the main ideas in this paper. Full proofs and details can be found in the technical report at [4].

The proof of our main regularity result is surprisingly hard to obtain. First, for each trajectory ρ, one obtains easily a number of steps n_ρ after which the trajectory is constant. However, there is in general no bound on n_ρ *uniform* over all ρ in the language. Thus, while every trajectory is simple to describe, the language turns out to be in general much more complex. We prove that the language does have a representation as a finite union of languages of the form $wA^iA^*B^*A^*B^*A^*\cdots B^*A^\omega$ with a bounded number of alternations. Our method computes effectively the language of M_{yeast}, as M_{yeast} has positive real eigenvalues, answering the question whether there exists an initial trajectory s.t. property (P_{yeast}) holds.

2 Preliminaries and definitions

▶ **Definition 1.** A Probabilistic Automaton (PA) $\mathcal{A}$ is a tuple $(Q, \Sigma, (M_\sigma)_{\sigma\in\Sigma}, Goal)$, where Q is a finite set of states, Σ is a finite alphabet, $Goal \subseteq Q$, and M_σ is the $|Q| \times |Q|$ transition stochastic matrix for each letter $\sigma \in \Sigma$. The PA is called *unary* PA (uPA for short) if $|\Sigma| = 1$.

For a unary PA $\mathcal{A}$ on alphabet $\{\sigma\}$, there is a unique transition matrix $M = M_\sigma$ of $Q \times Q$ with value in $[0, 1]$. For all $x \in Q$, we have $\sum_{y\in Q} M(x, y) = 1$. In other words, M is the Markov chain on set of states Q associated with $\mathcal{A}$.

A distribution δ over Q is a function $\delta : Q \to [0, 1]$ such that $\sum_{q\in Q} \delta(q) = 1$. Given M associated with a uPA, we denote by $M\delta$ the distribution given by $M\delta(q) = \sum_{q'\in Q} \delta(q')M(q', q)$ for all $q \in Q$. Notice that, considering δ and $M\delta$ as row-vectors, this corresponds to performing the matrix multiplication. That is, we consider M as a transformer of probabilities, as in [11, 1]: $(M\delta)(q)$ represents exactly the probability to be in q after applying M once, knowing that the initial distribution is δ. Inductively, $(M^n\delta)(q)$ represents the probability to be in q after applying n times M, knowing that the initial distribution is δ. We now review literature relating several problems on uPA with the Skolem's problem, named after the Skolem-Mahler-Lech Theorem [12],[14].

2.1 Relation with the Skolem problem

We start by defining three basic problems which have been studied extensively in different contexts. Given an initial distribution δ_0 and a uPA $\mathcal{A}$ with Matrix M, target states *Goal* and threshold γ:

Existence problem: Does there exist $n \in \mathbb{N}$ such that the probability to be in *Goal* after n iterations of M from δ_0 is γ (i.e., $\sum_{q \in Goal}(M^n\delta_0)(q) = \gamma$)?

Positivity problem: For all $n \in \mathbb{N}$ is the probability to be in *Goal* after n iterations of M from δ_0 at least γ (i.e., $\sum_{q \in Goal}(M^n\delta_0)(q) \geq \gamma$)?

Ultimate Positivity problem: Does there exist $n \in \mathbb{N}$ s.t., for all $m \geq n$, the probability to be in *Goal* after m iterations of M from δ_0 is at least γ (i.e., $\sum_{q \in Goal}(M^m\delta_0)(q) \geq \gamma$)?

Note that all these problems are defined from a fix initial distribution δ_0. These problems for PAs are specific instances of problems over general recurrence sequences, that have been extensively studied [16, 10]. It turns out that the existence for the special PA case is as hard as the existence (Skolem) problem over general recurrence sequences as shown in [3].

▶ **Theorem 2** ([3, 10]). *For general unary PAs, the existence and positivity are as hard as the Skolem's problem.*

The positivity result comes from the interreducibility of Skolem's problem and the positivity problem for general recurrence sequences [10]. The decidability of Skolem has been open for 40 years, and it has been shown that solving positivity, ultimate positivity or existence for general uPAs even for a small number of states (<50, depending on the problem considered) would entail major breakthroughs in diophantine approximations [18].

2.2 Simple unary PAs

In order to obtain decidability, we will consider restrictions over the matrix M associated with the uPA. The first restriction, fairly standard, is that M has distinct eigenvalues, which makes M diagonalizable.

▶ **Definition 3.** A stochastic matrix is *simple* if all its eigenvalues are distinct. A uPA is *simple* if its associated transition matrix is.

Some decidability results [19, 17] have been proved in the case of distinct eigenvalues for variants of the Skolem, which implies the following for *simple* uPAs:

▶ **Theorem 4.**
- *For simple unary PAs, ultimate positivity is decidable [19].*
- *For simple unary PAs with at most 9 states, positivity is decidable [17].*

We will consider the *simple* uPA restriction. Notice that the decidability restrictions in Theorem 4 for these two closely related problems have led to two different papers [17],[19] in the same conference, using different techniques. As we want to answer in a uniform way any regular question (subsuming among others the above three problems and regular properties such as (P_{yeast})) for uPAs of all sizes, we will later impose more restrictions. We start with the simple well-known observation that a simple unary PA has a unique stationary distribution.

▶ **Lemma 5.** *Let M be a simple stochastic matrix. Then there exists a unique distribution δ_{stat} such that $M\delta_{stat} = \delta_{stat}$.*

Proof. We give a sketch of proof here. We will later get an analytical explanation of this result. We have $M\delta = \delta$ iff $(M - Id)\delta = 0$. As M is diagonalizable and 1 is a eigenvalue of M of multiplicity 1, we have $Ker(M - Id)$ is of dimension 1. The intersection of distributions and of $Ker(M - Id)$ is of dimension 0, that is, it is a single point. ◀

As usual with PAs, we consider the probability to be in the set of states *Goal*, that is $\sum_{q\in Goal}(M^n\delta)(q)$. We consider only one threshold γ, for simplicity. In fact, the case of multiple thresholds reduces to this case, since the behavior is non-trivial for only one threshold, namely $\gamma_{stat} = \sum_{q\in Goal} \delta_{stat}(q)$ (see [4] for details).

2.3 Trajectories and ultimate periodicity

We want to know whether the n^{th} distribution $M^n\delta$ of the trajectory starting in distribution $\delta \in Init$ is above the hyperplane defined by $\sum_{q\in Goal} x_q = \gamma$, i.e., whether $\sum_{q\in Goal}[M^n\delta](q) \geq \gamma$. We will write $\rho_\delta(n) = A$ (Above) for $\sum_{q\in Goal}[M^n\delta](q) \geq \gamma$, and $\rho_\delta(n) = B$ (Below) else.

▶ **Definition 6.** The *trajectory* $\rho_\delta = \rho_0\rho_1 \cdots \in \{A, B\}^\omega$ from a distribution δ is the infinite word with $\rho_n = \rho_\delta(n)$ for all $n \in \mathbb{N}$.

We write the eigenvalues of M as $p_0, \ldots, p_k$ with $||p_i|| \geq ||p_j||$ for all $i < j$. Notice that $k + 1 = |Q|$ the number of states (as the uPA is simple). It is a standard result that all eigenvalues of Markov chains have modulus at most 1, and at least one eigenvalue is 1. We fix $p_0 = 1$. Now, as M is simple, it is also diagonalizable. Thus, there exists $a_i(\delta) \in \mathbb{C}$ (see [4] for further details) such that:

$$\rho_\delta(n) = A \text{ iff } \sum_{i=0}^{k} a_i(\delta)p_i^n \geq \gamma \tag{1}$$

In the following, we denote $u_\delta(n) = \sum_{i=0}^{k} a_i(\delta)p_i^n$ for all $n \in \mathbb{N}$. If ρ_δ is (effectively) ultimately periodic (i.e, of the form uv^ω), every (omega) regular property, such as existence, positivity and ultimate positivity is decidable (and are in fact easy to check). Unfortunately, this is not always the case, even for small simple unary PAs.

▶ **Theorem 7.** *[2] There exists an initial distribution δ_0 and simple unary PA $\mathcal{A}$ with 3 states, and coefficients and threshold in $\mathbb{Q}$, such that ρ_{δ_0} is not ultimately periodic.*

Proof Sketch. The unary PA is given by: $Goal = \{1\}$ is the first state, $\gamma = \frac{1}{3}$ and the associated matrix M_0 and initial distribution δ_0 are:

$$M_0 = \begin{pmatrix} 0.6 & 0.1 & 0.3 \\ 0.3 & 0.6 & 0.1 \\ 0.1 & 0.3 & 0.6 \end{pmatrix} \text{ and } \delta_0 = \begin{pmatrix} \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{2} \end{pmatrix}$$

The reason the trajectory is not ultimately periodic follows from the fact that the eigenvalues of M_0 are 1, $r_0e^{i\theta_0}$ and $r_0e^{-i\theta_0}$ with $r_0 = \sqrt{19}/10$ and $\theta_0 = cos^{-1}(4/\sqrt{19})$. ◀

An easy way to obtain ultimately periodic trajectories is to restrict to eigenvalues v which are roots of real numbers, that is, there exists $n \in \mathbb{N} \setminus \{0\}$ with $v^n \in \mathbb{R}$.

▶ **Proposition 8.** *Let $\mathcal{A}$ be a simple unary PA with eigenvalues $(p_i)_{i\leq m}$ all roots of real numbers. Then ρ_δ is ultimately periodic for all distributions δ. The (ultimate) period of ρ_δ can be chosen as any $m \in \mathbb{N} \setminus \{0\}$ such that p_i^m is a positive real number for all $i \leq m$.*

Now, for a finite state (Büchi) automaton $\mathcal{B}$ over the alphabet $\{A, B\}$, the membership problem, of whether a given single trajectory $\rho_\delta \in \mathcal{L}(\mathcal{B})$, is decidable. As it is easy to obtain a (small) automaton $\mathcal{B}$ for each of the existence, positivity and ultimate positivity problem such that this problem is true iff $\rho_\delta \in \mathcal{L}(\mathcal{B})$, we obtain:

▶ **Proposition 9.** *Let $\mathcal{A}$ be a simple unary PA with eigenvalues all roots of real numbers. Let δ_0 be a distribution. Then the existence, positivity and ultimate positivity problems from initial distribution δ_0 are decidable.*

Note that Propositions 8 and 9 hold even when the matrix associated with the PA is diagonalizable, but not necessarily simple.

3 Language of a unary PA

Using automata-based methods allows us to consider more complex problems, where the initial distribution is not fixed. We define the set *Init* of initial distributions as a convex polytope, that is the convex hull of a finite number of distributions.

▶ **Definition 10.** The language of a unary PA $\mathcal{A}$ wrt. the set of initial distributions *Init* is $\mathcal{L}(\mathit{Init}, \mathcal{A}) = \{\rho_\delta \mid \delta \in \mathit{Init}\} \subseteq \{A, B\}^\omega$.

Note that A and B, and the language, depend on the threshold γ. As we assumed this threshold value to be fixed, the language only depends on $\mathcal{A}$ and *Init*. As $\mathcal{A}$ is often clear from the context, we will often write $\mathcal{L}(\mathit{Init})$ instead of $\mathcal{L}(\mathit{Init}, \mathcal{A})$. For the yeast example $M = M_{yeast}$, we have eigenvalues $1; 0.7; 0.6$:

$$M \cdot \begin{pmatrix} 5/12 \\ 1/3 \\ 1/4 \end{pmatrix} = 1 \begin{pmatrix} 5/12 \\ 1/3 \\ 1/4 \end{pmatrix}; \; M \cdot \begin{pmatrix} 5/12 \\ -5/12 \\ 0 \end{pmatrix} = 0.7 \begin{pmatrix} 5/12 \\ -5/12 \\ 0 \end{pmatrix}; \; M \cdot \begin{pmatrix} 5/12 \\ 0 \\ -5/12 \end{pmatrix} = 0.6 \begin{pmatrix} 5/12 \\ 0 \\ -5/12 \end{pmatrix}$$

We can decompose two initial distributions $\delta_1, \delta_2 \in \mathit{Init}_{yeast}$ on the eigenvector basis:

$$\begin{pmatrix} 1/3 \\ 1/4 \\ 5/12 \end{pmatrix} = \begin{pmatrix} 5/12 \\ 1/3 \\ 1/4 \end{pmatrix} + \frac{1}{5} \begin{pmatrix} 5/12 \\ -5/12 \\ 0 \end{pmatrix} - \frac{2}{5} \begin{pmatrix} 5/12 \\ 0 \\ -5/12 \end{pmatrix}; \quad \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \end{pmatrix} = \begin{pmatrix} 5/12 \\ 1/3 \\ 1/4 \end{pmatrix} - \frac{1}{5} \begin{pmatrix} 5/12 \\ 0 \\ -5/12 \end{pmatrix}$$

Projecting on the first component, we have $\rho_{\delta_1}(n) = A$ iff $\frac{1}{12}0.7^n - \frac{1}{6}0.6^n \geq 0$, that is $\rho_{\delta_1} = B^4A^\omega$. Also, $\rho_{\delta_2}(n) = A$ iff $-\frac{1}{12}0.6^n \geq 0$, that is $\rho_{\delta_2} = B^\omega$. With the techniques developed in the following, we can prove more generally that, for all $n \in \mathbb{N}$, we can find an ϵ s.t., $\delta = (1/3 \;\; 1/3-\epsilon \;\; 1/3+\epsilon)^T$ has trajectory $\rho_\delta = B^nA^\omega$, and that $\mathcal{L}(\mathit{Init}_{yeast}) = B^*A^\omega \cup B^\omega$. Thus, property (P_{yeast}), from Introduction, does not hold for every initial distribution.

In general, if $\mathcal{L}(\mathit{Init}, \mathcal{A})$ is regular, then any regular question will be decidable. For instance, if $\mathcal{L}(\mathit{Init}, \mathcal{A})$ is regular, then it is decidable whether there exists $\delta_0 \in \mathit{Init}$ such that the existence problem is true for $\mathcal{A}, \delta_0$. One can also ask whether for a given convex polytope Q, some property (such as positivity) expressed e.g. with $LTL_{\mathcal{I}}$ [1] is true. Taking δ in the interior of Q, this corresponds to checking the robustness of the property around δ.

Clearly, simple PA $\mathcal{A}$ does not ensure the regularity of $\mathcal{L}(\mathit{Init}, \mathcal{A})$ because of Theorem 7 (by choosing $\mathit{Init} = \{\delta_0\}$ which is a convex polytope). Surprisingly, restricting eigenvalues to

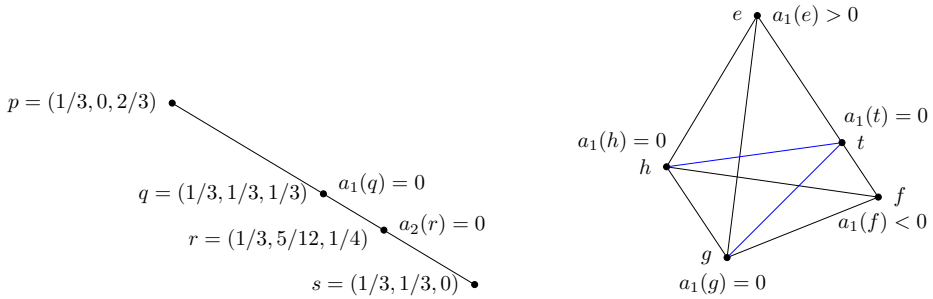


Figure 1 Breaking into convex polytopes with constant signs.

be *distinct* and *roots of real numbers* does not ensure regularity either (see Section 6). In the following, we thus take a stronger restriction: we assume that the eigenvalues of M are *distinct* and *positive real numbers*. That is, $p_0 = 1 > p_1 > \cdots > p_k \geq 0$ with $k+1 = |Q|$ the number of states. From Proposition 8, we obtain as corollary that for all δ_0, we have either $\rho_{\delta_0} = wA^\omega$ or $\rho_{\delta_0} = wB^\omega$ for w a finite word of $\{A, B\}^*$:

▶ **Corollary 11.** *Let M be a simple (or just diagonalizable) stochastic matrix with positive real eigenvalues. Then every trajectory ρ_{δ_0} is ultimately constant.*

However, the language $\mathcal{L}(Init_{yeast}, M_{yeast})$ shows that $\mathcal{L}(Init, \mathcal{A})$ is not always of the simple form $\bigcup_{w\in W_A} wA^\omega \cup \bigcup_{w\in W_B} wB^\omega$, for W_A, W_B two finite sets of finite words over $\{A, B\}^*$. Nevertheless, in the next two sections, we succeed in proving the regularity of $\mathcal{L}(Init, \mathcal{A})$, which is our main result:

▶ **Theorem 12.** *Let $\mathcal{A}$ be a unary PA with distinct positive real eigenvalues, and Init be a convex polytope of (initial) distributions. Then, $\mathcal{L}(Init, \mathcal{A})$ is effectively regular.*

Note that the hypotheses of Theorem 12 are decidable for $\mathcal{A}$ with rational coefficients. Indeed, it suffices to use linear algebra to compute the eigenvalues and vectors, and check whether their complex part is null. Further the proof carries through even when the matrix of $\mathcal{A}$ is diagonalizable (though we tackle just the simple case here). We also show that this result is tight, i.e., relaxing the hypothesis any further leads to non-regularity (see Section 6).

3.1 Partition of the set *Init* of initial distributions

Recall that we write $u_\delta(n) := \sum_{i=0}^{k} a_i(\delta) p_i^n$, where $a_i(\delta)$ are given by Equation (1) from the previous section. Because the eigenvalues are real numbers, $a_i(\delta)$ is a real number for every i and δ. Notice that a_i is a linear function in δ, that is, $a_i(\alpha\delta_1 + \beta\delta_2) = \alpha a_i(\delta_1) + \beta a_i(\delta_2)$. The trajectory ρ_δ depends crucially on the sign of $a_0(\delta)$, and if $a_0(\delta) = 0$, on the sign of $a_1(\delta)$, etc. First, let $L_i = \{\delta \mid a_0(\delta) = \cdots = a_i(\delta) = 0\}$. This is a vector space (i.e., it is in $\mathbb{R}^k$ and contains the space of distributions over Q), as for any $\nu_1, \nu_2 \in \mathbb{R}^k$, we have $\nu_1, \nu_2 \in L_i$ implies that any linear combination $\alpha\delta_1 + \beta\delta_2 \in L_i$ (since $a_i(\nu)$ is linear in ν, and the kernel of a linear function is a vector space).

We will divide the space of distributions into a finite set $\mathcal{H}$ of convex polytopes $H \in \mathcal{H}$ to keep the sign of each a_i constant on each polytope. Each $H \in \mathcal{H}$ satisfies that for all $e, f \in H$, for all $i \leq k$, we have $a_i(e), a_i(f)$ do not have different signs (either one is 0, or both are positive or both are negative). This can be done since $a_i(\nu)$ is continuous (as it is linear). This is pictorially represented in the left of Figure 1. For instance, we divide $Init_{yeast}$

into three polytopes: $\{(1/3, y, 2/3 - y) \mid y \leq 1/3\}$ and $\{(1/3, y, 2/3 - y) \mid 1/3 \leq y \leq 5/12\}$ and $\{(1/3, y, 2/3 - y) \mid y \geq 5/12\}$ as for $\delta = (1/3, 1/3, 1/3)$ we have $a_0(\delta) = 1$, $a_1(\delta) = 0$ (and $a_2(\delta) = -1/5$) and for $\delta = (1/3, 5/12, 1/4)$ we have $a_0(\delta) = 1$, $a_1(\delta) = -1/5$, $a_2(\delta) = 0$.

In general, we can assume that each of $H \in \mathcal{H}$ is the convex hull of $k + 2$ points (else we divide further: this can be done as the space has dimension $k + 1$). Consider the right part of Figure 1. Let *Init* be the convex hull of points e, f, g, h (in three dimensions) and $a_0(x) = 0$ and $a_2(x) > 0$ for all $x \in \{e, f, g, h, t\}$. Hence the sign of each trajectory ultimately depends upon $a_1(x)$. In the example, $a_1(g) = a_1(h) = 0$ while $a_1(e) > 0 > a_1(f)$. Then there is a point t between e and f for which $a_1(t) = 0$ (in fact, $t = |a_1(f)|/(|a_1(e)| + |a_1(f)|)e + |a_1(e)|/(|a_1(e)| + |a_1(f)|)f$). We have $L_1 \cap \mathit{Init}$ is the convex hull of h, g, t. We break *Init* into two convex polytopes, the convex hull of h, g, t, e and the convex hull of h, g, t, f.

Let $H \in \mathcal{H}$. We let P be the finite set of (at most $k + 2$) extremities of H. In particular, H is the convex hull of P. Now it suffices to show that the language $\mathcal{L}(H)$ (taking H as the initial set of distributions) of each of these convex polytopes H is regular to prove that the language $\mathcal{L}(\mathit{Init}) = \bigcup_{H \in \mathcal{H}} \mathcal{L}(H)$ is regular.

3.2 High level description of the proof

The proof of the regularity of the language $\mathcal{L}(H)$ starting from the convex polytope H is performed as follows. We first prove that there exists a N_{max} such that the ultimate language (after N_{max} steps) of H is effectively regular using analytical techniques.

▶ **Definition 13.** Given N_{max}, the *ultimate language* from a convex polytope H is defined as $\mathcal{L}_{ult}^{N_{max}}(H) = \{v \mid \exists w \in \{A, B\}^{N_{max}}, wv \in \mathcal{L}(H)\}$.

In the next section (Corollary 18), we show that this ultimate language $\mathcal{L}_{ult}^{N_{max}}(H)$ is regular, of the form $A^*B^* \cdots B^*A^\omega \cup A^*B^* \cdots A^*B^\omega$ with a bounded number of switches between A and B's. However, while for each prefix $w \in \{A, B\}^{N_{max}}$, the set H_w of initial distributions in H whose trajectory starts with w is a convex polytope; the language $\mathcal{L}(H_w)$ from H_w can be complex to represent. It is not in general $w\mathcal{L}_{ult}^{N_{max}}(H)$, but a strict subset.

In Section 5 (Lemma 21), we prove that the language $\mathcal{L}(H')$ associated with some carefully defined convex polytope $H' \subseteq H$ is a regular language, of the form $\bigcup_{w \in W} wA^iA^*B^* \cdots B^*A^\omega \cup wA^iA^*B^* \cdots A^*B^\omega$ for a finite set W. Further, removing H' from H gives rise to a finite number of convex polytopes with a smaller number of "sign-changes", as formally defined in the next section. Hence we can apply the arguments inductively (requiring potentially to change the N_{max} considered). Finally, the union of these languages gives the desired regularity characterization for $\mathcal{L}(H)$.

4 Ultimate Language

4.1 Limited number of switches

We first show that the ultimate language $\mathcal{L}_{ult}^{N_{max}}(H)$ is included into $A^*B^*A^* \cdots A^*B^\omega \cup A^*B^*A^* \cdots B^*A^\omega$ for some $N_{max} \in \mathbb{N}$, with a limited number of switches between A and B depending on properties of the set P of extremities of H.

We start by considering the generalisation of a sequence u_δ to a function over positive reals, and we will abuse the notation u_δ to denote both the sequence and the real function.

▶ **Definition 14.** A function of type $k \in \mathbb{N}$ is a function of the form $u : \mathbb{R}_{>0} \to \mathbb{R}$, with $u(x) = \sum_{j=0}^{k} \alpha_j p_j^x$, where $p_0 > \cdots > p_k > 0$.

Now, let $u : \mathbb{R}_{\geq 0} \to \mathbb{R}$ be a continuous function. We can associate with function u the (infinite) word $L(u) \in \{A, B\}^\omega, L(u) = (a_0 a_1 \ldots)$, where for all $n \in \mathbb{N}$, a_n is defined as $a_n = A$ if $u(n) \geq 0$ and $a_n = B$ otherwise. We have easily that $\rho_\delta = L(u_\delta)$. Knowing the zeros of u_δ and its sign before and after the zeros, defines uniquely the trajectory ρ_δ.

For example, let u be such that it has four zeros: $u(N-0.04) = u(N+10.3) = u(N+20) = u(N+35) = 0$ for some integer N. Assume that $u(0) < 0, u(N+1) > 0, u(N+11) < 0, u(N+30) < 0$ and $u(N+40) > 0$. Thus, by continuity of u, u is strictly negative on $[0, N-1]$, strictly positive on $[N, N+10]$, non-positive on $[N+11, N+34]$ and non-negative on $[N+35, \infty)$. Thus the associated trajectory $\rho_\delta = B^N A^{11} B^{24} A^\omega$.

Hence, it is important to analyze the zeros of functions u_δ. If the number of zeros is bounded, then the number of alternations between A's and B's in any trajectory ρ_δ from $\delta \in H$ will be bounded. In fact, it is a standard result (which we do not use hence do not reprove here) that every type k function u has at most k zeros. We now show a more precise bound on the number of zeros. Namely, for the convex hull H' of a finite set P' of distributions in H, the number of alternations between A's and B's in H' is limited by the number of alternations of the sign of the dominant coefficients of the distributions in P'.

Let $z \in \mathbb{N}$. For $i \in \{0, \ldots, z\}$, let $u^i(x) := a^i_0 p^x_0 + a^i_1 p^x_1 + \cdots + a^i_k p^x_k$, with $p_0 > p_1 > p_2 > \ldots > p_k > 0$, representing for instance the functions associated with the $z+1$ extremities of H'. We denote $dom(u^i)$ the dominant coefficient of u^i, that is the smallest integer j with $a^i_j \neq 0$. We reorder $(u^i)_{i \in \{0,\ldots,z\}}$ such that $dom(u^i) \leq dom(u^{i+1})$ for all $i < z$. We denote $sign\_dom(u^i) \in \{+1, -1\}$ as the sign of $dom(u^i)$. We will assume, as for H, that for all i, i', j, a^i_j and $a^{i'}_j$ have the same sign. We let $Z(u^0, \cdots, u^z) = |\{i \leq z-1 \mid sign\_dom(u^i) \neq sign\_dom(u^{i+1})\}|$. That is, $Z(u^0, \cdots, u^z)$ is the number of switches of sign between the dominant terms of u^i and u^{i+1}. We have $0 \leq Z(u^0, \cdots, u^z) \leq z$. Notice that as for $dom(u^i) = dom(u^j)$, we have $sign\_dom(u^i) = sign\_dom(u^j)$, $Z(u^0, \cdots, u^z)$ does not depend upon the choice in the ordering of $(u^i)_{i \in \{0,\ldots,z\}}$. We can now give a bound on the number of zeros of functions which are convex combinations of $u^0 \cdots u^z$.

▶ **Lemma 15.** *Let $u^0 \cdots u^z$ be $z+1$ type k functions. There exists a $N_{max} \in \mathbb{N}$ such that for all $\lambda_i \in [0, 1]$ with $\sum_i \lambda_i = 1$, denoting $u(x) = \sum_{i=0}^z \lambda_i u^i(x)$, $u(x)$ has at most $Z(u^0, \cdots, u^z)$ zeros after N_{max}. Further, if $u(x)$ has exactly $Z(u^0, \cdots, u^z)$ zeros after N_{max}, then its sign changes exactly $Z(u^0, \cdots, u^z)$ times (that is, no zero is a local maximum/minimum).*

In other words, we show that $u(x)$ behaves like a polynomial of degree $Z(u^0, \cdots, u^z)$ (as it has $Z(u^0, \cdots, u^z)$ dominating terms), although it has degree $k > Z(u^0, \cdots, u^z)$. In fact, we prove that for $\ell = dom(u^i)$, the coefficients $a^i_j p^x_j$ for all $j > \ell$ play a negligible role wrt. $a^i_\ell p^x_\ell$.

Let $H \in \mathcal{H}$, and P its finite set of extremal points. We can apply Lemma 15 to $u^0, \ldots, u^z$, the functions associated with the points of P (in decreasing order of dominating coefficient), and obtain a N_{max}. Now, since P is finite, the trajectories from P are ultimately constant, hence there exists N_y such that for all $i \leq y$, the trajectory of u^i is wA^ω or wB^ω for some $w \in \{A, B\}^{N_y}$. We define N_H to be the maximum of N_y and N_{max}. With this bound on the number of zeros, we deduce the following inclusion for the ultimate language $L^{N_H}_{ult}(H)$:

▶ **Corollary 16.** *Let $y = Z(u^0, \ldots, u^z)$. The ultimate language $L^{N_H}_{ult}(H) \subseteq C^*_1 \cdots C^*_{y-1} C^\omega_y \cup C^*_1 \cdots C^*_{y-1} C^\omega_{y-1}$ for $\{C_i, C_{i+1}\} = \{A, B\}$ for all $i < y$; and $C_y = A$ iff $sign\_dom(u^0)$ is positive.*

We can have 4 different sequences for $C^*_1 \cdots C^*_{y-1} C^\omega_y$ with $\{C_i, C_{i+1}\} = \{A, B\}$, depending on the first and last letters C_1, C_y (or equivalently, C_y and parity of y which determines C_1).

The proof of our main result on regularity of $\mathcal{L}(H)$ will proceed by induction over the *switching-dimension* $Z(H)$ of H which we define as $Z(H) = Z(u^0, \ldots, u^z)$. Notice that

we could define the switching dimension for any convex set (not necessarily a polytope) whenever the sign of $a_i(\delta)$ does not change within the convex set. Finally, we also define $sign\_dom(H) = sign\_dom(u^0)$.

4.2 Characterization of the Ultimate Language

We now show that the ultimate language of H is exactly $\mathcal{L}_{ult}^{N_H}(H) = A^*B^*A^* \cdots A^*B^\omega \cup A^*B^*A^* \cdots B^*A^\omega$, with at most $Z(H)$ switches of signs. We will state the associated technical Lemma 17 in the more general settings of "faces" as defined below, as it will be useful in the next section. Let P be the finite set of extremal points of a H. We call $(f^0, \ldots, f^y) \subseteq P$ a *face* of H if $Z(v^0, \ldots, v^y) = y = Z(H)$ for the functions $(v^0, \ldots, v^y)$ associated with the extremal points $(f^0, \ldots, f^y)$. Notice that denoting H' the convex hull of F, we can choose $N_{H'} = N_H$ (which is not the case for H' an arbitrary polytope included into H).

▶ **Lemma 17.** *Given a face $(f^0, \ldots, f^y) \subseteq P$ of H with associated functions v^i, we have, for all $n_1, n_2, \ldots, n_y \in \mathbb{N}$ there exist $\lambda_i \in [0,1]$ with $\sum_i \lambda_i = 1$, such that denoting $\widetilde{v}(x) = \sum_{i=1}^{y} \lambda_i v^i(x)$, $L(\widetilde{v}) = wA^{n_1}B^{n_2} \ldots B^{n_y}A^\omega$ (for y even) for some prefix $w \in \{A,B\}^{N_H}$.*

That is, for all $n_1, \cdots n_y$, one can find a prefix w of size N_H and a point δ in the convex hull of $e^1, \cdots e^y$, such that $\rho_\delta = wA^{n_1}B^{n_2} \cdots B^{n_y}A^\omega$ (assuming the correct parity of y). Let H' be the convex hull of $f^0, \ldots, f^y$. Hence $Z(H') = Z(H)$. Then, the ultimate language of H' (i.e., the language after prefixes of size N_H associated with y) contains $A^*B^* \ldots B^*A^\omega$ with y switches between A and B, which is the converse of Corollary 16. We can thus deduce the following about the ultimate language:

▶ **Corollary 18.** *$L_{ult}^{N_H}(H) = L_{ult}^{N_H}(H') = C_1^*C_2^* \ldots C_y^*A^\omega \cup C_1^*C_2^* \ldots C_{y-1}^*B^\omega$ with $\{C_i, C_{i+1}\} = \{A, B\}$.*

Proof. We first prove the result for $L_{ult}^{N_H}(H')$. We can apply lemma 17 to H' and lemma 15 to H'. We obtain the first part of the union. Now, let $H'' \subseteq H'$ be the convex hull of $e^1, \cdots, e^y$ (that is excluding e^0). Each point δ in $H' \setminus H''$ has a trajectory which ends with A^ω, as $dom(u_\delta) = dom(v^1)$, and thus $sign\_dom(u_\delta) = sign\_dom(v^1)$ by construction of H (and $H' \subseteq H$). Thus the points with trajectory ending with B^ω are in H'', and applying lemma 15, we know that their ultimate trajectory has at most $y-1$ switches. Applying Lemma 17 to H'', we obtain the second hand of the union. Now, $L_{ult}^{N_H}(H') \subseteq L_{ult}^{N_H}(H)$, and $L_{ult}^{N_H}(H) \subseteq C_1^*C_2^* \ldots C_y^*A^\omega \cup C_1^*C_2^* \ldots C_{y-1}^*B^\omega$ by Corollary 16. ◀

However, we cannot immediately conclude that $\mathcal{L}(H)$ is regular. Though N_H is finite, computable and there are a finite number of prefixes w of size N_H, we need to show that the subset of $\mathcal{L}_{ult}^{N_H}(H)$ appearing after a given $w \in \{A,B\}^{N_H}$ is (effectively) regular. This is what we do formally in the following section.

5 Regularity of the Language

Let $\{e^0, \cdots, e^z\} = P$ the extremal points of H. Let u^p the function associated with each $e^p \in P$. We denote $y = Z(H) = Z((u^p)_{p \leq z})$. We will show the regularity of $\mathcal{L}(H)$ using an induction on $Z(H)$.

For $Z(H) = 0$, the regularity of $\mathcal{L}(H)$ is trivial as all the dominant coefficients have the same sign. Thus, by Corollary 16, the ultimate language is $\mathcal{L}_{ult}^{N_H}(H) = A^\omega$ and then the language is $\mathcal{L}(H) = \bigcup_{w \in W} wA^\omega$; or the ultimate language is $\mathcal{L}_{ult}^{N_H}(H) = B^\omega$ and the language is $\mathcal{L}(H) = \bigcup_{w \in W} wB^\omega$, for a finite set of $W \subseteq \{A,B\}^{N_H}$.

For $w \in \{A,B\}^{N_H}$, consider $H_w = \{\delta \in H \mid \rho_\delta = wv\}$, i.e., the language of words which begin with the prefix w. It is easy to see that $H_w \subseteq H$ is a polytope. Hence $Z(H_w) \leq Z(H)$. Observe that $\mathcal{L}(H) = \bigcup_{w \in \{A,B\}^{N_H}} \mathcal{L}(H_w)$. To show the regularity of $\mathcal{L}(H)$, we show the regularity of $\mathcal{L}(H_w)$ for each of the finitely many $w \in \{A,B\}^{N_H}$. For each $w \in \{A,B\}^{N_H}$, we have two cases: either $Z(H_w) < Z(H)$; then we apply the induction hypothesis and we are done. Or else, $Z(H_w) = Z(H) = y$. In this case, the sketch of proof is as follows:

- We show that there exists J such that for all $i \leq y$ and all $j \geq J$, we have a point h^i_j in H_w with trajectory $wC_1^jC_2C_3\cdots C_{i-1}C_i^\omega$. This is shown by applying lemma 17 to each face $(f^0,\ldots,f^y)$ of H and then using convexity arguments and the fact that $Z(H_w) = Z(H)$.
- Subsequently, denoting H' the convex hull of $h^0_J \cdots h^y_J$, we will deduce that $\mathcal{L}(H')$ is a regular language of the form $wC_1^JC_1^*C_2^*C_3^*\cdots C_{i-1}^*C_i^\omega$,
- Partitioning $H_w \setminus H'$ into a finite set of polytopes, we obtain polytopes of lower switching-dimensions, which have regular languages by induction.
- We conclude since the finite union of these regular languages is a regular language, namely $\mathcal{L}(H_w)$.

We now formalize the above proof sketch in a sequence of lemmas, whose details can be found in [4]. For all faces F of H, applying Lemma 17 gives for all $j \in \mathbb{N}$, a point $g_j(F)$ of the convex hull of F with trajectory $w_jC_1^jC_2C_3\cdots C_y^\omega$, for some $w_j \in \{A,B\}^{N_H}$. We now prove that (g_j) converges towards f^y, the point of F with lowest dominant term.

▶ **Lemma 19.** *For every face $F = (f^0,\ldots,f^y)$ of H, $(g_j(F))_{j\in\mathbb{N}}$ converges towards f^y as j tends to infinity.*

For all j, we consider $F(y,j)$ the convex hull of $\{g_j(F) \mid F \text{ is a face of } H\}$. Every point of $F(y,j)$ has trajectory $w'C_1^jC_2C_3\cdots C_y^\omega$ for some $w' \in \{A,B\}^{N_H}$. We then show by convexity that H_2 intersects $F(y,j)$, i.e., it has a point with trajectory $w'C_1^jC_2C_3\ldots C_y^\omega$.

▶ **Lemma 20.** *For $w \in \{A,B\}^{N_H}$ with $Z(H_w) = Z(H)$, there exists J s.t. for all $j > J$, $F(y,j) \cap H_w \neq \emptyset$.*

Similarly, for all $i \leq y$ we can define a polytope $F(i,j)$. All the points in $F(i,j)$ have trajectory $w'C_1^jC_2C_3\cdots C_i^\omega$ for some $w' \in \{A,B\}^{N_H}$. We can find a J_i and a point $h^i_j \in H_w$ with trajectory $wC_1^jC_2C_3\cdots C_i^\omega$ for all $i \leq y$ and all $j > J_i$. Now, as the number of $i \leq y$ is bounded, one can find such a J uniform over all $i \leq y$ (by taking maximum over all i).

Consider $F(J)$ the convex hull of $F(0,J),\ldots,F(y,J)$. By convexity, all the points in $F(J)$ have their n-th letters of trajectory as C_1 for all $n \in [N_H+1\cdots N_H+J]$, since this is true for all points of $F(i,J)$. Hence, the language of $H_w \cap F(J)$ is included into $wC_1^JC_1^*C_2^*\cdots C_y^\omega \cup wC_1^JC_1^*C_2^*\cdots C_{y-1}^\omega$, because of the bound on the number of alternations after N_H of trajectories from points of H (Lemma 15). We show now that we have equality.

▶ **Lemma 21.** *The language of the convex hull of $\{h^0_J,\ldots,h^y_J\}$ is exactly $wC_1^JC_1^*C_2^*C_3^*\cdots C_{y-1}^*C_y^\omega \cup wC_1^JC_1^*C_2^*C_3^*\cdots C_{y-2}^*C_{y-1}^\omega$.*

Hence the language of $H_w \cap F(J)$ is $wC_1^iC_1^*\cdots C_y^\omega \cup wC_1^iC_1^*\cdots C_{y-1}^\omega$.

Next, we note that the set $H_w \setminus F(J)$ may not be convex. However, one can partition $H_w \setminus F(J)$ into a finite number of convex polytopes. Now, let G be a convex polytope in $H_w \setminus F(J)$. We want to show that $Z(G) < Z(H_w) = Z(H) = y$. Indeed, else, one could apply Lemma 20 to $G_w = G$ and for some J' obtain $F(i,j) \cap G \neq \emptyset$ for any $j > J'$, which contradicts G being a convex set in $H_w \setminus F(J)$.

Hence one can compute the language of every G inductively, and each of them is regular. Finally, this leads to the regularity of $\mathcal{L}(H_w)$ by finite union, and to the regularity of $\mathcal{L}(H)$, and again by finite union to the regularity of $\mathcal{L}(Init)$. This concludes our proof of the main regularity result, i.e., Theorem 12.

6 Non-regularity of the symbolic dynamics

In this section, we will prove that symbolic dynamics of uPA can produce non-regular languages even when eigenvalues of the transition matrix are distinct roots of real numbers. We prove this by constructing such a uPA and choosing the set of initial distributions carefully. Consider a uPA $\mathcal{A}_1$ with 7 states $q_1, \dots q_7$, $Goal = \{q_7\}$, and the following transition matrix:

$$M_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{1}{512} & \frac{8r+3}{512} & \frac{3+3r}{64} & \frac{13+16r}{128} & \frac{9+2r}{32} & \frac{1+4r}{16} & \frac{1-r}{2} \end{bmatrix}$$

where $r = cos(\pi/8) = \frac{\sqrt{\sqrt{2}+2}}{2}$. Eigenvalues of M_1 are 1, $\frac{1}{2}e^{\pm i\pi/2}$, $\frac{1}{2\sqrt{2}}e^{\pm i3\pi/4}$ and $\frac{1}{4}e^{\pm i7\pi/8}$, which are distinct roots of real numbers. We choose $\gamma = \sum_{q\in Goal}\delta_{stat}(q) = \delta_{stat}(q_7) = \frac{512}{65(17+8cos(\frac{\pi}{8}))}$ (for any other choice of γ, the language is regular).

Let δ be the initial distribution and $M_1^n\delta$ be the distribution after n steps of M_1. We consider a basis of eigenvectors such that the eigenvector corresponding to eigenvalue 1 is the stationary distribution and the remaining eigenvectors are normalized such that the 7^{th} component (corresponding to the *Goal* state) of each of them is 1. This is possible as the 7^{th} component of each eigenvector of M_1 is non-zero. Now, by eigenvalue decomposition:

$$M_1^n\delta(7) = \mu_0 + \frac{\mu_1}{2^n}(e^{ni\pi/2}+e^{-ni\pi/2}) + \frac{\mu_2}{(2\sqrt{2})^n}\left(e^{n3i\pi/4}+e^{-n3i\pi/4}\right) + \frac{\mu_3}{4^n}(e^{n7i\pi/8}+e^{-n7i\pi/8})$$

where $\mu_0 = \gamma$ and δ written in the eigenvector basis is $(1, \mu_1, \mu_1, \mu_2, \mu_2, \mu_3, \mu_3)$.

Consider the initial set of distributions *Init* to be the line segment $(P1, P2)$ where $P1 = (1, a, a, b, b, c, c)$ and $P2 = (1, 0, 0, b, b, c, c)$ in the eigenvector basis, where $a = \frac{cos(3\pi/8)}{2^{46}\sqrt{2}}$, $b = \frac{1}{2^{20}}$, $c = \frac{1}{2^{11}cos(3\pi/8)}$. These values are chosen so that μ_0 dominates over the other terms in the above equation, which ensures that $P1$ and $P2$ correspond to valid distributions in the standard basis. Note that *Init* is the set of convex combinations of distributions $P1$ and $P2$. Now, we can show our main theorem of this section.

▶ **Theorem 22.** *$\mathcal{L}(Init, \mathcal{A}_1)$ is not regular.*

Proof sketch. Let $L = \mathcal{L}(Init, \mathcal{A}_1)$. For $x, y, z, k \in \mathbb{N}$, we define $L^k_{x,y,z} = \{w \in \Sigma^\omega \mid \exists w' \in L,\ \forall i \in \mathbb{N}, w'_{k(i-1)+x} = w_{3(i-1)+1}, w'_{k(i-1)+y} = w_{3(i-1)+2}, w'_{k(i-1)+z} = w_{3(i-1)+3}\}$. That is, for every $a_1a_2a_3\dots \in L$, $a_xa_ya_za_{k+x}a_{k+y}a_{k+z}\dots \in L^k_{x,y,z}$ where $x, y, z \le k$. It is easy to see that if $L^k_{x,y,z}$ is non-regular, so is L. Now we can show that $L^{16}_{2,3,4} = \{(ABB)^2(AAB)^{y+g(\mu_1,\mu_2,\mu_3)}(BAB)^y(BAA)^w : y \ge 0\}$. As the range of y is $[1, \infty)$ and $g(\mu_1, \mu_2, \mu_3)$ is a bounded function, hence $L^{16}_{2,3,4}$ is not regular. Thus, L is not regular which completes the proof. ◀

7 Conclusion

Though unary Probabilistic Automata (or Markov Chains) are a simple formalism, there are still many basic problems, whose decidability is open and thought to be very hard. Indeed, it is surprising yet significant that even after assuming strong hypotheses, their behaviors cannot be described easily. In this paper, we proposed a class of unary probabilistic automata, for which all properties of some logic, e.g. $LTL_{\mathcal{I}}$ are decidable even considering an infinite set of initial distributions. This allows for instance to check for the robustness of the behavior wrt. a given property (e.g. positivity) for behaviors around a given initial distribution. Further, while we proved our results with respect to a single hyperplane (above is A, below is B), we can generalize these to more general settings as well. Finally, we showed that relaxing the assumptions immediately leads to non-regularity.

Acknowledgement. We would like to thank Manindra Agrawal and P. S. Thiagarajan for very fruitful discussions, and the anonymous reviewers for several useful suggestions. This work was partially supported by ANR projet STOCH-MC (ANR-13-BS02-0011-01) and DST-INSPIRE faculty award [IFA12-MA-17].

References

1 M. Agrawal, S. Akshay, B. Genest, and P. S. Thiagarajan. Approximate verification of the symbolic dynamics of Markov chains. In *LICS*, pages 55–64. IEEE Computer Society, 2012.

2 M. Agrawal, S. Akshay, B. Genest, and P. S. Thiagarajan. Approximate verification of the symbolic dynamics of Markov chains. *J.ACM*, 62(1):183–235, 2015.

3 S. Akshay, T. Antonopoulos, J. Ouaknine, and J. Worrell. Reachability problems for Markov chains. *Information Processing Letters*, 115(2):155–158, 2015.

4 S. Akshay, B. Genest, B. Karelovic, and N. Vyas. On regularity of unary probabilistic automata. Technical report available online at `http://perso.crans.org/~genest/AGKV16.pdf`, 2015.

5 Alberto Bertoni. The solution of problems relative to probabilistic automata in the frame of the formal languages theory. In *GI Jahrestagung*, pages 107–112, 1974.

6 R. Chadha, Dileep Kini, and M. Viswanathan. Decidable Problems for Unary PFAs. In *QEST*, pages 329–344. LNCS 8657, 2014.

7 K. Chatterjee and M. Tracol. Decidable Problems for Probabilistic Automata on Infinite Words. In *LICS*, pages 185–194. IEEE Computer Society, 2012.

8 N. Fijalkow, H. Gimbert, and Y. Ouahladj. Deciding the Value 1 Problem for Probabilistic Leaktight Automata. In *LICS*, pages 295–304. IEEE Computer Society, 2012.

9 H. Gimbert and Y. Ouahladj. Probabilistic Automata on Finite Words: Decidable and Undecidable Problems. In *ICALP*, pages 527–538. LNCS 6199, 2010.

10 V. Halava, T. Harju, and M. Hirvensalo. Positivity of second order linear recurrent sequences. *Discrete Applied Mathematics*, 154(3), 2006.

11 V. A. Korthikanti, M. Viswanathan, G. Agha, and Y. Kwon. Reasoning about MDPs as transformers of probability distributions. In *QEST, p. 199-208*. IEEE, 2010.

12 C. Lech. A note on recurring series. *Ark. Mat.*, 2, 1953.

13 O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147(1-2):5–34, 2003.

14 K. Mahler. Eine arithmetische Eigenschaft der Taylor-koeffizienten rationaler Funktionen. *Proc. Akad. Wet. Amsterdam*, 38, 1935.

15 L. Maruthi, I. Tkachev, A. Carta, E. Cinquemani, P. Hersen, G. Batt., and A. Abate. Towards real-time control of gene expression at the single cell level: a stochastic control approach. In *CMSB*, pages 155–172. LNCS/LNBI, 2014.

16 J. Ouaknine and J. Worrell. Decision problems for linear recurrence sequences. In *RP*, pages 21–28, 2012.

17 J. Ouaknine and J. Worrell. On the Positivity Problem for simple linear recurrence sequences. In *ICALP*, pages 318–329. LNCS 8573, 2014.

18 J. Ouaknine and J. Worrell. Positivity problems for low-order linear recurrence sequences. In *SODA*, pages 366–379. ACM-SIAM, 2014.

19 J. Ouaknine and J. Worrell. Ultimate Positivity is decidable for simple linear recurrence sequences. In *ICALP*, pages 330–341. LNCS 8573, 2014.

20 Azaria Paz. *Introduction to Probabilistic Automata*. Academic Press, 1971.

21 Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.

22 P. Turakainen. On Stochastic Languages. *Information and Control*, 12:304–313, 1968.

The Expanding Search Ratio of a Graph*

Spyros Angelopoulos[1], Christoph Dürr[2], and Thomas Lidbetter[3]

1 **Sorbonne Universités, Université Pierre et Marie Curie Paris 06, CNRS, LIP6, Paris, France**
spyros.angelopoulos@lip6.fr
2 **Sorbonne Universités, Université Pierre et Marie Curie Paris 06, CNRS, LIP6, Paris, France**
christoph.durr@lip6.fr
3 **Department of Mathematics, London School of Economics, UK**
t.r.lidbetter@lse.ac.uk

Abstract

We study the problem of searching for a hidden target in an environment that is modeled by an edge-weighted graph. Most of the previous work on this problem considers the *pathwise* cost formulation, in which the cost incurred by the searcher is the overall time to locate the target, assuming that the searcher moves at unit speed. More recent work introduced the setting of *expanding search* in which the searcher incurs cost only upon visiting previously unexplored areas of the graph. Such a paradigm is useful in modeling problems in which the cost of re-exploration is negligible (such as coal mining).

In our work we study algorithmic and computational issues of expanding search, for a variety of search environments including general graphs, trees and star-like graphs. In particular, we rely on the deterministic and randomized *search ratio* as the performance measures of search strategies, which were originally introduced by Koutsoupias and Papadimitriou [ICALP 1996] in the context of pathwise search. The search ratio is essentially the best competitive ratio among all possible strategies. Our main objective is to explore how the transition from pathwise to expanding search affects the competitive analysis, which has applications to optimization problems beyond the strict boundaries of search problems.

1998 ACM Subject Classification F.1.2 Modes of Computation, F.2.2 Nonnumerical Algorithms and Problems, I.2.8 Problem Solving, Control Methods and Search

Keywords and phrases Search games, randomized algorithms, competitive analysis, game theory

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.9

1 Introduction

Searching for a hidden target is a common task in everyday life and a significant computational problem with important applications. The game-theoretic approach to searching considers the search *environment* (e.g., a network represented by an undirected, edge-weighted graph), and defines a game between two players: the *Hider*, for whom a *pure strategy* is a hiding point in the environment; and the *Searcher*, for whom a pure strategy is some choice of how to navigate through the environment. We assume that the Searcher is initially located to a starting point called the *root*, that the Hider is static, and that the environment (e.g., the search graph) is fully known to the Searcher. Note that unlike the setting of pursuit-evasion

* This work was supported by project ANR-11-BS02-0015 "New Techniques in Online Computation" (NeTOC).


© Spyros Angelopoulos, Christoph Dürr, and Thomas Lidbetter;
licensed under Creative Commons License CC-BY

games, we assume that the Searcher has no knowledge of the Hider's position. In a zero-sum game formulation of the general search problem, given a pure strategy S of the Searcher and a pure strategy H of the Hider, we define the *cost function* $c(S, H)$ as the total effort incurred by the Searcher in locating the Hider. Since the game is zero-sum, this cost also represents the gain of the Hider. A strategy for the Searcher may be evaluated by calculating the highest cost $c(S) = \sup_H c(S, H)$ paid to locate a Hider at point H, and we define the *minimax value* of the game as the smallest value $\inf_S c(S)$ that this cost can attain. The extension to games where the players may use randomized (or *mixed*) strategies follows the standard game-theoretic framework.

This simple, yet inclusive formulation has provided the mathematical framework for a study of several variants of search games (the textbooks [22, 4, 2] provide a comprehensive summary of important developments in the field over the past three decades). This is due to the versatility in defining the cost (i.e., payoff) function, which makes the game-theoretic framework applicable to many settings. For instance, in much previous work, the cost function is defined as the *search time* $T(S, H)$, that is the time taken for the Searcher following a strategy S to find a target located at a point H. An alternative way of defining the cost function, which we will use in this paper, is the so-called *normalized search time*, defined formally as $\hat{T}(S, H) = T(S, H)/d(H)$, where $d(H)$ is the minimum time for the Searcher to reach H, assuming this point is known to him. This concept of the normalized search time is very useful when searching in unbounded domains, in which $T(S, H)$ can be arbitrarily large. Moreover, this cost formulation is equally applicable to bounded domains (e.g., finite graphs). In particular, it gives rise to the *competitive ratio* measure, also called *search ratio* in this context, which was first addressed by Koutsoupias and Papadimitriou [29]. The name itself is a reference to the well-known competitive analysis of online algorithms, since the position of the Hider is unknown to the Searcher and the normalization parameter $d(H)$ can be seen as the "optimal" cost for locating the Hider assuming full information.

Another aspect of the cost function is related to when the searcher incurs cost, which may vary according to the setting at hand. The usual search paradigm when seeking a target on a network is what we now call *pathwise search*, in which the Searcher follows a continuous, unit-speed path until the target is reached. Very recently, Alpern and Lidbetter [5] introduced a new search paradigm termed *expanding search*, in which, informally, the Searcher may restart the search at any time from any previously reached point. As a concrete application, [5] mentions the problem of mining for coal: here, digging into a new site is far more costly than moving through an area that has already been dug.

In [5] expanding search games were studied assuming non-normalized measures. In this paper, we study the competitive ratio of expanding search, which, following earlier work of Koutsoupias and Papadimitriou [29] on pathwise search we refer to as the search ratio; namely, we assume the normalized measure as defined earlier.

Related work

Following Gal's formalization of network search games [21] in the framework of pathwise search with un-normalized search time, the problem has had considerable attention, for example [35, 34, 23, 16, 1, 10, 11]. Expanding search was introduced in [5] in the setting in which the payoff is the total (un-normalized) cost of finding the Hider. Among other results, [5] solved the game in the case that the network is either a tree, or 2-edge-connected. This model was extended in [30] to a setting in which the Searcher must locate multiple hidden objects.

The competitive ratio of pathwise search was studied in [29], who showed that the problem of computing the optimal search ratio in a given undirected graph is NP-complete (and

MAX-SNP hard to approximate). They also gave a search strategy based on repeated executions of DFS traversals that achieves a constant approximation of the (deterministic) competitive ratio. Similar results can be obtained concerning the *randomized competitive ratio* (assuming that the Searcher randomizes over its strategy space). Connections between graph searching and other classic optimization problems such as the Traveling Salesman problem and the Minimum Latency problem were shown in [8]. The setting in which the search graph is revealed as the search progresses was studied in [18].

A specific search environment that has attracted considerable attention in the search literature is the *star-like* environment. More specifically, in the unbounded variant, the search domain consists of a set of infinite lines which have a common intersection point (the root of the Searcher); this problem is also known as *ray searching*. Ray searching is a natural generalization of the well-known *linear search* problem [13, 12] (informally called the "cow-path problem"). Optimal strategies were initially given by Gal [20] as well as by Baeza-Yates *et al.* [9] and Jaillet and Stafford [24]. Other related work includes the study of randomization [36], [27], multi-Searcher strategies [32], searching with turn cost [17], the variant in which some probabilistic information on target placement is known [24], [25], and the related problem of designing *hybrid algorithms* [26].

Bounded star search, namely the setting in which distance of the target from the root is bounded was studied in [31, 15]. New performance measures were introduced in [28, 33]. The problem of locating a certain number among the many Hiders was studied in [7].

It must be emphasized that star search has applications that are not confined to locating a target (which explains its significance and popularity). Some concrete applications include drilling for oil in a number of different locations [33], as well as the design of algorithms that return acceptable solutions even if interrupted during their execution [14, 6].

Contribution

In this work we study expanding search by means of competitive analysis, assuming a variety of search graphs such as stars, trees, and general edge-weighted, undirected graphs. Our main motivation is to explore how the transition from pathwise to expanding search affects the deterministic and the randomized search ratios. As in [5], we address both the *discrete* and the *continuous* settings. In the discrete setting, the Hider can hide only on the vertices of the graph. In contrast, in the continuous setting the search space is a network with arcs, and the Hider can hide anywhere across an arc.

We begin in Section 2 with the definitions of the (expanding) search ratio and randomized search ratio, both in the continuous and discrete settings. In Section 3 we give a simple optimal algorithm for the deterministic search ratio in the continuous setting, and show that this is also a 2-approximation of the randomized search ratio. In the discrete setting, we show that the problem of finding the optimal (deterministic) search ratio is NP-hard (using a substantially more complicated reduction than for pathwise search in [29]). Applying well-known iterative deepening techniques, we obtain a $4\ln(4) \approx 5.55$ approximation.

Our main technical results, presented in Section 4, apply to the discrete setting where the search graph is a star. Here, it is easy to show that an optimal deterministic search strategy searches the edges of the star in non-decreasing order of length. This strategy is also a 2-approximation of the randomized search ratio. We thus turn our attention to obtaining a better *randomized* search strategy. More precisely, we give a randomized strategy that approximates the randomized search ratio within a factor of $5/4$, representing a significant improvement over the afore-mentioned 2-approximation. Improved approximations via randomization are usually not easy to achieve (see, e.g. [29]). Our result confirms the

intuitive expectation that randomization has significant benefits. Moreover, using game-theoretic techniques we show a tight bound on the randomized search ratio of any n-edge star graph; namely, we show it is at most $(n+1)/2$, with equality if and only if all the edges have the same length. We accomplish this by analyzing an explicit randomized strategy.

As argued earlier, star-search problems have applications that transcend searching. This is indeed the case in expanding search. Consider the following problem: we are given a collection of n boxes, among which only one contains a prize. We can open a box i at cost d_i. We seek a (randomized) strategy for locating the prize, and the randomized search ratio of the strategy is the total expected cost of all opened boxes, divided by the cost of the box that holds the prize. This problem is equivalent to the problem of finding the (randomized) search ratio of a star graph.

Lastly, we show in Section 5 that the principle of searching vertices in non-decreasing order of their distance from the root extends from stars to trees and unweighted graphs, and gives the optimal deterministic search ratio and a 2-approximation for the optimal randomized search ratio.

Since our main objective is to study the algorithmic and computational impact of *re-exploration* due to the transition from pathwise search to expanding search, it is important to compare our results to the best-known bounds in the context of pathwise search (and, specifically, in the discrete model). More precisely, for unweighted graphs, [29] gives asymptotic approximations of the deterministic and randomized search ratios equal to 6 and 8.98, respectively, but its techniques appear to be applicable also to general graphs, at the expense of somewhat larger, but constant approximations. Furthermore, [29] notes that the problems of computing the search ratios of trees are "surprisingly hard". In contrast, for expanding search of unweighted graphs and (weighted) trees we obtain optimal algorithms and a 2-approximation of the deterministic and randomized search ratios, respectively. Going beyond the threshold of 2 requires more sophisticated strategies even for simple environments such as a star. For general graphs, we note that our 5.55 approximation is strict, and not asymptotic. As a last observation, we note that the pathwise and expanding search algorithms appear to depend crucially on the approximability of TSP and the Steiner Tree problem, respectively.

2 Preliminaries

Continuous setting

We begin by defining an *expanding search* on a connected network Q with root O, as introduced in [5]. The network Q consists of nodes and edges, and Q is endowed with Lebesgue measure corresponding to length. The measure of a subset A of Q is denoted by $\lambda(A)$. Let $\mu = \lambda(Q)$ be the total measure of Q.

▶ **Definition 1.** An expanding search on a network Q with root O is a family of connected subsets $S(t) \subset Q$ (for $0 \le t \le \mu$) satisfying: (i) $S(0) = O$; (ii) $S(t) \subset S(t')$ for all $t \le t'$; and (iii) $\lambda(S(t)) = t$ for all t.

Since we will only consider expanding searches in this paper, we refer to a given expanding search as a *search strategy*. For a point $H \in Q$ we write $d(H)$ for the length of the shortest path from O to H. For a given expanding search S of Q and a point $H \in Q$, let $T(S,H) = \min\{t : H \in S(t)\}$ be the *search time* of H under S. For $H \neq O$, let $\hat{T}(S,H)$ be the ratio $T(S,H)/d(H)$ of the search time of H to the distance of H from the root. We refer to $\hat{T}(S,H)$ as the *normalized search time*.

▶ **Definition 2.** The (deterministic) search ratio $\sigma_S = \sigma_S(Q)$ of a search strategy S for a network Q is given by $\sigma_S(Q) = \sup_{H \in Q - \{O\}} \hat{T}(S, H)$. The (deterministic) search ratio, $\sigma = \sigma(Q)$ of Q is given by $\sigma(Q) = \inf_S \sigma_S(Q)$, where the infimum is taken over all search strategies S. If $\sigma_S = \sigma$ we say that S is optimal.

We will also consider randomized search strategies: that is, search strategies that are chosen according to some probability distribution. We denote randomized strategies by lower case letters, and for such a strategy s and point $H \in Q$ we denote the *expected search time* by $T(s, H)$ and the *expected normalized search time* by $\hat{T}(s, H) = T(s, H)/d(H)$.

▶ **Definition 3.** The randomized search ratio $\rho_s = \rho_s(Q)$ of a randomized search strategy s for a network Q is given by $\rho_s(Q) = \sup_{H \in Q - \{O\}} \hat{T}(s, H)$. The randomized search ratio, $\rho = \rho(Q)$ of Q is given by $\rho(Q) = \inf_s \rho_s(Q)$,, where the infimum is taken over all possible randomized search strategies s. If $\rho_s = \rho$ we say that s is optimal.

We can view the randomized search ratio of a network as the value of the following zero-sum game $\Gamma(Q, O)$. A strategy S for the Searcher is a search strategy as described above and a strategy H for the Hider is a point on Q. The payoff of the game is the normalized search time $\hat{T}(S, H)$. For *mixed* (randomized) strategies s and h of the Searcher and Hider, respectively, the expected payoff is denoted by $\hat{T}(s, h)$.

In [5] the authors considered a similar zero-sum game in which the player's strategy sets are the same but the payoff is the unnormalized search time $T(S, H)$. They showed that the strategy sets are compact with respect to the uniform Hausdorff metric and that $T(S, H)$ is lower semicontinuous in S for fixed H. Since $d(H)$ is a constant for fixed H, it follows that $\hat{T}(S, H) = T(S, H)/d(H)$ is also lower semicontinuous in S for fixed H, and by the Minimax Theorem of Alpern and Gal [3], we have the following theorem.

▶ **Theorem 4.** *Let Q be a network with root O. The game $\Gamma(Q, O)$ has a value V, which is equal to the randomized search ratio $\rho(Q)$. The Searcher has an optimal mixed strategy (with search ratio $\rho(Q)$) and the Hider has ϵ-optimal mixed strategies.*

Theorem 4 allows us to find lower bounds for the randomized search ratio, since for any mixed Hider strategy h, we have $\rho(Q) \geq \inf_S \hat{T}(S, h)$.

Discrete setting

In the discrete setting the search environment consists of an undirected, edge-weighted graph $G = (\mathcal{E}, \mathcal{V})$, with $|\mathcal{V}| = n$, and a distinguished root vertex $O \in \mathcal{V}$; moreover, the Hider is always located on some vertex of G. The weight or *length* of edge e, denoted by $\lambda(e)$, represents the time required to search that edge (we assume, via normalization, that $\lambda(e) \geq 1$)). We will call a graph of unit edge weights *unweighted*.

A *search strategy* on G is a sequence of edges, starting from the root, chosen so that the set of edges that have been searched is a connected, increasing set. More precisely:

▶ **Definition 5.** An expanding search S on a graph G is a sequence of edges $e_1, \ldots, e_{n-1}$ such that every *prefix* $\{e_1, \ldots, e_k\}$, $k = 1, \ldots, n-1$ is a subtree of G rooted at O.

For a given vertex $v \in \mathcal{V}$ and a given search strategy $S = (e_1, \ldots, e_{n-1})$, denote by S_v the first prefix $\{e_1, \ldots, e_k\}$ that covers v. The *search time*, $T(S, v)$ of v is the total time $\sum_{e \in S_v} \lambda(e)$ taken to search all the edges before v is discovered. Let $d(v)$ denote the length of the shortest path from O to v, which is the minimum time for the Searcher to discover v. For

$v \neq O$ the *normalized search time* is denoted by $\hat{T}(S, v) = T(S, v)/d(v)$. The deterministic and randomized search ratios are then defined along the lines of Definitions 2 and 3.

We will view the randomized search ratio ρ through the lens of a zero-sum game between a Searcher and a Hider. Unlike the continuous setting, the game is finite. The Searcher's pure strategy set, $\mathcal{S}$ is the set of search strategies and the Hider's pure strategy set is the set $\mathcal{V} - O$ of non-root vertices of G. For a Hider strategy $v \in \mathcal{V}$ and a Searcher strategy $S \in \mathcal{S}$, the payoff of the game is $\hat{T}(S, v)$, which the Hider wants to maximize and the Searcher wants to minimize. By the standard minimax theorem for zero-sum games, the value of the game is equal to the randomized search ratio and an optimal randomized search strategy is an optimal strategy for the Searcher in the game. A mixed strategy for the Hider is a probability distribution h over the vertices of G, and for mixed strategies h and s of the Hider and Searcher respectively, we write $T(s, h)$ and $\hat{T}(s, h)$ for the corresponding expected search time and expected normalized search time.

3 General graphs

3.1 Continuous setting

The optimal deterministic search ratio is simple to compute in the continuous case, using a "uniformly-expanding", BFS strategy. For any $r \geq 0$, we denote the closed disc of radius r around O by $D(r) = \{x \in Q : d(x) \leq r\}$. Consider the real function $f : \mathbb{R}^+ \to \mathbb{R}$ given by $f(r) = \lambda(D(r))$, so $f(r)$ is the measure of the set of points at distance no more than r from the root. The function f is strictly increasing so has an inverse g. The interpretation is that $g(t)$ is the unique radius r for which $D(r)$ has measure t.

For a network Q with root O, consider the expanding search S^* defined by $S^*(t) = D(g(t))$. Thus, $S^*(t)$ is an expanding disc of radius $g(t)$. It is easy to verify that S^* is indeed an expanding search. First we note that $S^*(t)$ is connected, since $D(r)$ is always connected. It also trivially satisfies (i) and (ii) from Definition 1, and (iii) is also satisfied since $\lambda(S^*(t)) = \lambda(D(g(t))) = f(g(t)) = t$.

It is very easy to see that S^* can be implemented in polynomial time. We will show that S^* is optimal. First note that the search time of a point $H \in Q$ under S^* is the unique time t such that $S^*(t) = D(d(H))$, so $T(S^*, H) = \lambda(D(d(H))) = f(d(H))$. Hence

$$\sigma_{S^*} = \sup_{H \in Q - \{O\}} \frac{f(d(H))}{d(H)} = \sup_{r>0} \frac{f(r)}{r} = \frac{1}{\inf_{t>0} \frac{g(t)}{t}}. \tag{1}$$

▶ **Theorem 6.** *The search ratio σ of a network Q with root O is given by $\sigma = \sup_{r>0} \frac{f(r)}{r}.$, for f defined as above. Therefore, the expanding search S^* is optimal.*

We further show that the randomized search ratio is always at least half of the search ratio, which implies that S^* is a 2-approximation of the optimal randomized search strategy.

▶ **Proposition 7.** *For a network Q with root O, the randomized search ratio ρ satisfies $\sigma/2 \leq \rho \leq \sigma$. Furthermore, the bounds are tight.*

3.2 Discrete setting

In this section we show that the problem of computing the (deterministic) search ratio is NP-hard. We also give a search strategy that achieves a $4\ln(4) \approx 5.55$ approximation ratio.

For a given graph G, suppose that S is a search strategy which searches the edges in the order $e_1, \ldots, e_{n-1}$. For a subgraph H of G, denote by $\lambda(H)$ the sum of all the lengths $\lambda(e)$ of edges e in H.

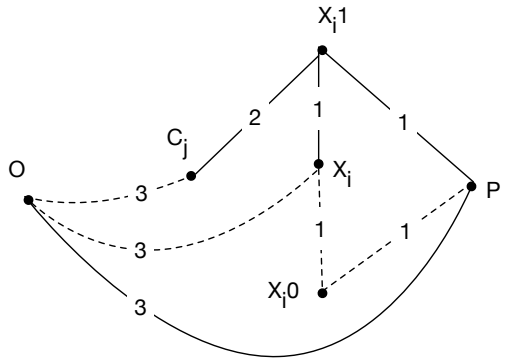


Figure 1 A schematic view of the graph G used in the reduction of Theorem 8.

.

▶ **Theorem 8.** *Given a graph G with root O and a constant $R \geq 0$, it is NP-Complete to decide whether $\sigma(G) \leq R$.*

Proof. The proof is based on a reduction from 3-SAT. Given a 3-SAT instance consisting of n variables and m clauses with $m \geq n$, we construct an instance of our problem.

We construct the graph G consisting of vertices O, P, a vertex C_j for every clause (the *clause vertices*), vertices X_i (the *variable vertices*) and vertices X_i^0, X_i^1 (the *literal vertices*) for every variable. For every $i = 1, \ldots, n$ there are unit length edges of the form $(X_i, X_i^0), (X_i, X_i^1), (P, X_i^0), (P, X_i^1)$. For every variable x_i appearing positively in the j-th clause there is an edge (C_j, X_i^1) of length 2 and for every variable x_i appearing negatively in the j-th clause there is an edge (C_j, X_i^0) of length 2. For every $j = 1, \ldots, m$ there is an edge (O, C_j) of length 3 and for every $i = 1, \ldots, n$ there is an edge (O, X_i) of length 3. Finally, there is an edge (O, P) of length 3. We fix $R = 1 + \frac{2}{3}(n + m)$. The construction is shown in Figure 1.

Note that the vertices can be partitioned according to their distance from O. In particular, vertex P, as well as variable and clause vertices have distances 3, whereas literal vertices have distance 4.

We must show that there exists a boolean assignment to the variables satisfying all clauses if and only if the search ratio of G is at most R.

For the easy direction of the proof, consider a boolean assignment $b \in \{0, 1\}^n$ to the variables satisfying all clauses. We will show that there is a search strategy with search ratio at most R. First we construct a tree H covering all distance 3 vertices with total length $3R$. The tree consists of the edge (O, P), the edges $(P, X_i^{b_i}), (X_i, X_i^{b_i})$ for every $i = 1, \ldots, n$, and for every clause C_j an edge from C_j to the literal vertex corresponding to a literal satisfying the clause. We denote the tree constructed from b by H^b. The total length of H^b is $3 + 2n + 2m$ which is exactly $3R$ by the choice of R. To turn the tree into a search strategy S we order the edges from H by increasing distance from 0. This sequence S is completed in arbitrary order with the remaining edges of the form (X_i, X_i^0) and (X_i, X_i^1). We have $\rho_S(P) = 1, \rho_S(C_j) \leq 3R/3, \rho_S(X_i) \leq 3R/3$ and $\rho(X_i^x) \leq (3R + n)/4 \leq R$ for every i, j, which shows that the search ratio of G is at most R.

For the hard direction, assume that there is a search strategy with search ratio at most R. Let H be its shortest prefix covering all distance 3 vertices. By the definition of the search ratio we know that $\lambda(H) \leq 3R$. Through a sequence of transformations we turn H into a tree of the form H^b with $\lambda(H^b) \leq \lambda(H)$. This will show that b is a satisfying assignment for the formula and complete the proof of the theorem.

- If (O, P) does not belong to H we add it. This must create a cycle, containing an edge of the form (O, v) with $v \neq P$. Now we remove this edge, and obtain a tree of the same length.
- If there is an edge (O, C_j) in H for some j, then we replace this edge by the edges $(C_j, v), (v, P)$, where v is a vertex corresponding to a literal from the j-th clause. Some of the added edges might already have been present. The result is a tree of no greater length.
- If there is an edge of the form (O, X_i) in H for some i, then we replace this edge by the edges $(X_i, X_i^0), (X_i^0, P)$. Again, the result is a tree with of no greater length.
- At this stage we know that O is only connected to P in the tree.
- If there is a vertex C_j connected to several vertices $v_1, \ldots, v_k$ for $k \geq 2$, then we remove the edges $(C_j, v_1), \ldots, (C_j, v_k)$. Hence, the tree now contains k components, each containing some distinct vertex v_i, and only one of them also containing P. Without loss of generality suppose that v_1 and P are in the same component. Then we add (C_j, v_1) back to H and add for each vertex v_i $(i = 2, \ldots, k)$, a length 2 path to P, going through any literal vertex to which v_i is connected. This way we maintain a tree, and do not increase its length (it might even decrease if some of the added edges were already present).
- At this stage we know that every C_j vertex is adjacent to exact one length 2 edge. Also for every $i = 1, \ldots, n$, among the vertices $\{X_i, X_i^0, X_i^1, P\}$ there are at least two edges, one adjacent to X_i and one adjacent to P. The last edge is necessary since otherwise there would be no connection from the vertices $\{X_i, X_i^0, X_i^1\}$ to P, since by the previous point we know that such a path could not go through a clause vertex. Let k be the total number of additional edges that could exist among the vertex sets $\{X_i, X_i^0, X_i^1, P\}$ over all $i = 1, \ldots, n$. Then the total length of H is $3 + 2m + 2n + k$, which by assumption is at most $3R$. By the choice of R we have equality and thus $k = 0$. This shows that H is a tree of the form H^b for some $b \in \{0, 1\}^n$, which a satisfying assignment. ◀

Using an approach similar to the doubling heuristic of [29], we obtain a constant-approximation algorithm. It is worth pointing out that the algorithm doubles the radius, and explores the resulting graph by computing a Steiner tree of the corresponding vertex set (in contrast to pathwise search, in which the resulting graph is simply explored depth-first).

▶ **Theorem 9.** *There is a polynomial-time search algorithm that approximates $\sigma(G)$ within a factor of $4\ln(4) + \epsilon < 5.55$.*

4 Star search

In this section we consider problems related to the search ratio and randomized search ratio of a star graph in the discrete setting. Note that, unlike the generalizations of the cow-path problem [19], our star environment in finite and discrete. Suppose that G is a star graph consisting of n edges $e_1, \ldots, e_n$ of lengths $d_1, \ldots, d_n$ with each e_i incident to the root, O and to a vertex v_i. We assume without loss of generality that $1 = d_1 \leq d_2 \cdots \leq d_n$. An expanding search of such a graph corresponds simply to a permutation of edges (or vertices).

Computing the optimal deterministic search ratio is relatively simple; in particular, it suffices to search the edges in non-decreasing order of length.

▶ **Proposition 10.** *The deterministic search ratio of a star graph G is*

$$\sigma(G) = \max_{j \leq n} \frac{\sum_{i \leq j} d_i}{d_j}. \tag{2}$$

In addition, searching the vertices in the order $v_1, v_2, \ldots, v_n$ is an optimal strategy.

4.1 Randomized approximation of the randomized search ratio

We now turn to the randomized search ratio, $\rho = \rho(G)$ of the star G. Here, the Hider's pure strategy set is the set $\{v_1, \ldots, v_n\}$ of n leaves and the Searcher's pure strategy set is the set of orderings of the edges. We start with a lower bound for ρ which helps us obtain optimal randomized stategies for 2-edge stars (but is inefficient for general stars). For $j = 1, \ldots, n$, let $\mu_j = \sum_{i=1}^{j} d_i$ be the total length of the first j edges and let $D_j = \sum_{i=1}^{j} d_i^2$ be the sum of the squares of the lengths of the first j edges.

▶ **Lemma 11.** *For each $k = 1, \ldots, n$, consider the Hider strategy which chooses a vertex v_j, where $j \leq k$ with probability $p_j = d_j^2/D_k$. Then for any pure Searcher strategy S, we have*

$$\rho_S \geq \hat{T}(S, p) = \frac{1}{2}\left(1 + \frac{\mu_k^2}{D_k}\right). \tag{3}$$

Let $\pi_k = \frac{1}{2}\left(1 + \frac{\mu_k^2}{D_k}\right)$ be the right-hand side of (3). A natural question is whether the randomized search ratio for star graphs is exactly π_n. For instance, it is easy to see that if the star only has two edges, this is the indeed the case and the bound helps us obtain optimal randomized strategies. However, it is not true in general that the randomized search ratio of an n-edge star is π_n, even for $n = 3$.

An immediate consequence of (3) is that the pure search strategy that simply searches all the edges in the order $e_1, \ldots, e_n$ is a 2-approximation of the optimal strategy. However, this relies on a search strategy that is deterministic, so a reasonable question is whether it is possible to obtain a better approximation using a randomized search strategy. This is indeed the case, as we shall give a search strategy that has approximation ratio $5/4$. The idea behind the strategy is to randomize between edges of similar size; moreover, this partition is also determined at random.

For the purpose of the analysis, let t be the smallest integer such that the longest edge has length less than 2^t. Consider the partition of the edges into subsets $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_t$ where the set $\mathcal{A}_i$ consists of all edges of lengths d with $2^{i-1} \leq d < 2^i$, $i \in \mathbb{N}^+$.

Before we define formally our strategy, we observe that the strategy s that randomizes uniformly between all edges in each of the $\mathcal{A}_i$ successively is not efficient. For example, consider the star with edges lengths $d_1 = 1, d_2 = \cdots = d_{n-1} = 2 - \epsilon, d_n = 2$. We have $\hat{T}(s, v_n) \approx n$, whereas the strategy that searches v_1 first before randomizing uniformly between the remaining edges has a randomized search ratio of approximately $n/2$.

▶ **Definition 12** (Randomized deepening strategy)**.** For each $i = 1, \ldots, t$ choose some x_i uniformly at random between 2^{i-1} and 2^i and let $x_0 = 1$ and $x_{t+1} = 2^t$. For $i = 0, \ldots, t$, let $\mathcal{B}_i$ be the set of edges with length in the interval $[x_i, x_{i+1})$. The **randomized deepening strategy**, s randomizes uniformly between all the edges in each $\mathcal{B}_i$ in the order $\mathcal{B}_0, \ldots, \mathcal{B}_t$.

Let α_i be the measure of the edges in $\cup_{j \leq i} \mathcal{A}_j$ and let $\hat{\alpha}_i$ be the measure of the edges in $\mathcal{A}_i$. Let k_i be the sum of the squares of the lengths of the edges in $\cup_{j \leq i} \mathcal{A}_j$ and let $\hat{k}_i$ be the sum of the squares of all the lengths of the edges in $\mathcal{A}_i$. We start with two simple lemmas which will help us bound the expected search time of a vertex.

▶ **Lemma 13.** *The expected measure of the edges in $\mathcal{A}_i$ with length less than x_i is $2\hat{\alpha}_i - \hat{k}_i/2^{i-1}$.*

▶ **Claim 14.** *For any $i = 1, \ldots, k$ we have $2^{i-1}\hat{\alpha}_i \leq \hat{k}_i \leq 2^i \hat{\alpha}_i$.*

▶ **Theorem 15.** *The approximation ratio of the randomized deepening strategy s, as defined in Definition 12 is $5/4$. Namely, $\rho_s \leq (5/4)\rho$.*

Proof. Consider a vertex v at distance d from O. We calculate the expected search time $T(s,v)$ of v. To simplify the analysis, we scale the length of all the edges in the star so that $2 \le d < 4$ and $e \in \mathcal{A}_2$. This means that the shortest edge v_1 of the star may now have length less than 1, and we write $\mathcal{A}_0$ for the set of edges with length less than 1.

Let T_1, T_2, T_3 be the expected time spent searching edges in $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$, respectively, before reaching v. So $T(s,v) = \alpha_0 + T_1 + T_2 + T_3$.

To calculate the expected time T_1, observe that if x_2 is less than d (which happens with probability $(d-2)/2$) then the time is $T_1 = \hat{\alpha_1}$. If x_2 is at least d (which happens with probability $(4-d)/2$) then the expected time spent searching $\mathcal{A}_1$ is the sum of the expected measure of the edges in $\mathcal{A}_1$ with length less than x_1 and half the expected measure of the edges in $\mathcal{A}_1$ with length at least x_1. By Lemma 13, this is $(2\hat{\alpha_1} - \hat{k_1}) + (1/2)(\hat{k_1} - \hat{\alpha_1}) = 3\hat{\alpha_1}/2 - \hat{k_1}/2$. Hence $T_1 = \left(\frac{d-2}{2}\right)\hat{\alpha_1} + \left(\frac{4-d}{2}\right)(3\hat{\alpha_1}/2 - \hat{k_1}/2) = \left(2 - \frac{d}{4}\right)\hat{\alpha_1} - \left(1 - \frac{d}{4}\right)\hat{k_1}$.

To calculate T_3, observe that if x_2 is greater than d then no arcs in $\mathcal{A}_3$ are searched before e. Otherwise, if x_2 is no greater than d (which happens with probability $(d-2)/2$), the expected time spent searching $\mathcal{A}_3$ is half of the expected measure of the edges in $\mathcal{A}_3$ with length less than x_3. So by Lemma 13, $T_3 = \left(\frac{d-2}{2}\right)(\hat{\alpha_3} - \hat{k_3}/8) = \left(\frac{d}{2} - 1\right)\hat{\alpha_3} - \left(\frac{d}{16} - \frac{1}{8}\right)\hat{k_3}$.

Let $f(x)$ be the measure of edges in $\mathcal{A}_2$ with length no greater than x. Then $T_2 = \frac{d}{2} + \int_{x=2}^{d}(f(x) + \frac{1}{2}(\hat{\alpha_2} - f(x))) \cdot \frac{1}{2}dx + \int_{x=d}^{4}\frac{1}{2}f(x) \cdot \frac{1}{2}dx = \frac{d}{2} + \frac{1}{4}(d-2)\hat{\alpha_2} + \int_2^4 \frac{1}{4}f(x)dx = \frac{d}{2} + \left(\frac{d}{4} + \frac{1}{2}\right)\hat{\alpha_2} - \left(\frac{1}{4}\right)\hat{k_2}$. Combining these and rearranging we get: $T(s,v) = \frac{d}{2} + \left(1 - \frac{d}{4}\right)(k_0 - \alpha_0) + \left(\frac{3}{4} - \frac{d}{4}\right)(2\alpha_1 - k_1) + \left(\frac{3}{8} - \frac{d}{16}\right)(4\alpha_2 - k_2) + \left(\frac{d}{16} - \frac{1}{8}\right)(8\alpha_3 - k_3)$. The second of these five terms is negative since $d \le 4$ and $k_0 \le \alpha_0$ (by Claim 14).

Dividing by d, we obtain an expression for the randomized search ratio ρ_s of s:

$$\rho_s \le \frac{1}{2} + \left(\frac{3}{4d} - \frac{1}{4}\right)(2\alpha_1 - k_1) + \left(\frac{3}{8d} - \frac{1}{16}\right)(4\alpha_2 - k_2) + \left(\frac{1}{16} - \frac{1}{8d}\right)(8\alpha_3 - k_3).$$

Let $\rho^{(1)}, \rho^{(2)}, \rho^{(3)}$ denote the first, second and third terms in the above expression, respectively. The following lemma bounds these terms with respect to ρ.

▶ **Lemma 16.** *For $\rho^{(1)}, \rho^{(2)}, \rho^{(3)}$ defined as above we have $\frac{1/2+\rho^{(3)}}{\rho} \le 2 - 4/d$, and $\frac{\rho^{(2)}}{\rho} \le 3/d - 1/2$, and $\frac{\rho^{(1)}}{\rho} \le 3/(2d) - 1/2$.*

Using Lemma 16 we obtain $\frac{\rho_s}{\rho} \le \frac{\rho^{(1)}}{\rho} + \frac{\rho^{(2)}}{\rho} + \frac{1/2+\rho^{(3)}}{\rho} \le (2 - 4/d) + (3/d - 1/2) + (3/(2d) - 1/2) = 1 + 1/(2d) \le 5/4$ (with equality holding when $d = 2$). ◀

4.2 Tight bounds for the randomized search ratio of a star graph

We now consider how large the deterministic and randomized search ratios can be for a star graph G with n edges. From (2), we have $\sigma(G) = \max_{j \le n} \frac{\sum_{i \le j} d_i}{d_j} \le \max_{j \le n} \frac{j d_j}{d_j} = n$. This upper bound is tight if and only if all edges have the same length. In this case it is very easy to see that the randomized search ratio ρ is $(n+1)/2$, and the optimal search strategy is to search the vertices in a uniformly random order. In contrast, it is not so easy to see that $(n+1)/2$ is the largest value that the randomized search ratio can attain for *any* star graph with n edges. We will show that this is indeed the case by inductively defining a particular randomized search strategy whose randomized search ratio is bounded above by $(n+1)/2$. We thus prove that the bound is tight.

For a given star graph G we inductively define a randomized search strategy s_k on the star graph G_k consisting of only the edges $e_1, \ldots, e_k$ with total length μ_k. Having defined the strategy s_k, we will define s_{k+1} as a randomized mix of two strategies, s_{k+1}^+ and s_{k+1}^-.

Table 1 Maximum value of $\hat{T}(s, v)$.

Search strategy, s	Vertex, v	
	v_i for some $i \leq k$	v_{k+1}
s^+_{k+1}	ρ_{s_k}	$\mu_k/d_{k+1} + 1$
s^-_{k+1}	$\rho_{s_k}(1 + d_{k+1}/\mu_k)$	$\mu_k/(2d_{k+1}) + 1 - D_k/(2\mu_k d_{k+1})$

▶ **Definition 17.** Suppose s_k has been defined for some $k = 1, \ldots, n-1$. Let s^+_{k+1} and s^-_{k+1} be randomized search strategies on G_{k+1} defined by:

(i) s^+_{k+1}: follow the strategy s_k on G_k and then search edge e_{k+1}.

(ii) s^-_{k+1}: choose a time t uniformly at random in $[0, \mu_k]$ and denote the edge that is being searched at time t by e. Follow the strategy s_k, but search edge e_{k+1} immediately before searching e.

Before giving the precise definition of s_k, we evaluate the normalized expected search times $\hat{T}(s^+_{k+1}, v_i)$ and $\hat{T}(s^-_{k+1}, v_i)$ in terms of ρ_{s_k} for the vertices v_i with $i = 1, \ldots, k+1$.

First suppose $i \leq k$. Then clearly $\hat{T}(s^+_{k+1}, v_i) \leq \rho_{s_k}$ (with equality for some $i \leq k$). Under s^-_{k+1}, with probability $T(s_k, v_i)/\mu_k$ edge e_{k+1} is searched before e_i, so the expected search time of v_i is $T(s_k, v_i) + (T(s_k, v_i)/\mu_k)d_{k+1}$. Hence $\hat{T}(s^-_{k+1}, v_i) = \frac{T(s_k,v_i)+(T(s_k,v_i)/\mu_k)d_{k+1}}{d_i} = \hat{T}(s_k, v_i)(1 + d_{k+1}/\mu_k) \leq \rho_{s_k}(1 + d_{k+1}/\mu_k)$. Now suppose $i = k+1$. Under s^+_{k+1}, the time taken to find the Hider is $\mu_k + d_{k+1}$, so $\hat{T}(s^+_{k+1}, v_{k+1}) = \mu_k/d_{k+1} + 1$. Under s^-_{k+1}, the expected search time is $\mu_k/2 + d_{k+1}$ minus a random correction error which depends upon which edge e is being searched under s_k at the random time t chosen uniformly in $[0, \mu_k]$. The edge e is e_i with probability d_i/μ_k, and in this case the expected value of the correction error is $d_i/2$. Hence the expected value of this correction error is $\sum_{i=1}^{k}(d_i/\mu_k) \cdot (d_i/2) = D_k/(2\mu_k)$. So we have $\hat{T}(s^-_{k+1}, v_{k+1}) = \frac{\mu_k/2+d_{k+1}-D_k/(2\mu_k)}{d_{k+1}} = \mu_k/(2d_{k+1}) + 1 - D_k/(2\mu_k d_{k+1})$.

To sum up, the expected search ratio for each combination of strategies can be bounded above by the payoffs in Table 1. We can then proceed to define s_n.

▶ **Definition 18.** Let s_1 be the only strategy available on G_1. Suppose s_k has already been defined on G_k for some $k = 1, \ldots, n-1$. The strategy s_{k+1} is an optimal mixture of s^+_{k+1} and s^-_{k+1} in the zero-sum game with payoff matrix given by Table 1.

The search ratio of s_n can be calculated iteratively, since the search ratio $\rho_{s_{k+1}}$ of s_{k+1} is the value of the game with payoff matrix given by Table 1, for each $k = 1, \ldots, n-1$. We use this to show that $\rho_{s_n} \leq (n+1)/2$.

▶ **Theorem 19.** *The randomized search ratio ρ of star network G with n edges is at most $(n+1)/2$, with equality if and only if all the edges have the same length.*

Proof. We have already pointed out that $\rho = (n+1)/2$ for the star whose edges all have the same length. To show that $\rho \leq (n+1)/2$ we use induction on the number of edges to show that $\rho_{s_n} \leq (n+1)/2$. It is clear that for $k = 1$, we have $\rho_{s_k} = 1 = (k+1)/2$, so assume that $\rho(s_k) \leq (k+1)/2$ for some $k > 1$ and we will show that $\rho_{k+1} \leq (k+2)/2 = k/2 + 1$.

First observe that if $d_{k+1} \geq 2\mu_k/k$ then the Searcher can ensure a payoff of no more than $k/2 + 1$ in the game in Table 1 just by using strategy s^+_{k+1}. This is because the payoff ρ_{s_k} against a vertex v_i with $i \leq k$ is no more than $(k+1)/2$ by the induction hypothesis and the payoff against v_{k+1} is $\mu_k/d_{k+1} + 1 \leq k/2$.

So assume that $d_{k+1} \leq 2\mu_k/k$, and note also that $d_{k+1} \geq \mu_k/k$, since the lengths of the edges are non-decreasing and d_{k+1} must be at least the average length of edges $e_1, \ldots, e_k$.

Table 2 Upper bounds for $\hat{T}(s, v)$.

Search strategy, s	Vertex, v	
	v_i for some $i \leq k$	v_{k+1}
s^+_{k+1}	$(k+1)/2$	$\mu_k/d_{k+1} + 1$
s^-_{k+1}	$(k+1)(1 + d_{k+1}/\mu_k)/2$	$\mu_k/(2d_{k+1}) + 1 - \mu_k/(2kd_{k+1})$

By the induction hypothesis, $\rho_{s_k} \leq (k+1)/2$, so the value of the game with payoff matrix given by Table 1 cannot decrease if we replace ρ_{s_k} with $(k+1)/2$ in the table. The value also does not decrease if we replace $-D_k$ by the maximum value it can take, which is $-\mu^2/k$ (that is, its value when $d_1, \ldots, d_k$ are all equal). In summary, $\rho_{s_{k+1}}$ is no more than the value of the game given in Table 2.

By assumption, against strategy s^+_{k+1}, the best response of the Hider (that is, the highest payoff) is given by choosing vertex v_{k+1}. We show that against strategy s^-_{k+1}, the Hider's best response is to choose a vertex v_i with $i \leq k$. This follows from writing the difference, δ between the payoffs in entries $(2,1)$ and $(2,2)$ of Table 2 as $\delta = (k-1)\frac{\mu_k}{2d_{k+1}}\left(\left(\frac{k+1}{k-1}\right)\left(\frac{d_{k+1}}{\mu_k}\right)^2 + \frac{d_{k+1}}{\mu_k} - 1/k\right)$. The quadratic in (d_{k+1}/μ_k) inside the parentheses is increasing for positive values of d_{k+1}/μ_k, and when $d_{k+1}/\mu_k = 1/k$ the quadratic is positive. Since $d_{k+1}/\mu_k \geq 1/k$, we must have $\delta \geq 0$.

Hence the Hider does not have a dominating strategy in the game in Table 2. It is also clear that the Searcher does not have a dominating strategy, since it is better to search e_{k+1} last if and only if the Hider is at some v_i with $i \leq k$. Therefore the game in Table 2 has a unique equilibrium in proper mixed strategies (that is, the players both play each of their strategies with positive probability). The search ratio $\rho_{s_{k+1}}$ of s_{k+1} is bounded above by the value V of the game, which is easily verified to be $V = k/2 + 1 - \frac{\frac{k}{2}(d_{k+1}/\mu_k - 1/k)^2}{(d_{k+1}/\mu_k)^2 + 1/k}$. This is clearly at most $k/2 + 1$, with equality if and only if $d_{k+1}/\mu_k = k$. Equality is only possible if $d_1 = d_2 = \ldots = d_{k+1} = \mu_k/k$. ◀

5 Trees and unweighted graphs

We conclude with the cases in which the graph is either a tree or an unweighted graph (in the discrete setting). If G is a graph with root O, for any $r > 0$ let G_r be the sub-graph of G with vertex set $\mathcal{V}_r$ consisting of all the vertices in G of distance no more than r from the root and with edge set $\mathcal{E}_r$ consisting of all the edges in $\mathcal{E}$ adjacent to some vertex in $\mathcal{V}_r$. The following proposition generalizes Proposition 10.

▶ **Proposition 20.** *Let G be a rooted graph and suppose that G is a tree or an unweighted graph. Then the search ratio σ is given by $\sigma = \sup_{r>0} \frac{\lambda(G_r)}{r}$. An optimal search strategy is to search the vertices in non-decreasing order of their distance from the root.*

We also generalize Proposition 7 and make it applicable to the discrete setting:

▶ **Proposition 21.** *The randomized search ratio of a tree or unweighted graph satisfies $\sigma/2 \leq \rho \leq \sigma$.*

References

1 S. Alpern, V. Baston, and S. Gal. Network search games with immobile hider, without a designated searcher starting point. *International Journal of Game Theory*, 37(2):281–302, 2008.

2 S. Alpern, R. Fokkink, L. Gąsieniec, R. Lindelauf, and V. S. Subrahmanian, editors. *Search theory. A game theoretic perspective.* New York, NY: Springer, 2013.

3 S. Alpern and S. Gal. A mixed strategy minimax theorem without compactness. *SIAM Journal on Control and Optimization*, 26(6):1357–1361, 1988.

4 S. Alpern and S. Gal. *The theory of search games and rendezvous.* Kluwer Academic Publishers, 2003.

5 S. Alpern and T. Lidbetter. Mining coal or finding terrorists: The expanding search paradigm. *Operations Research*, 61(2):265–279, 2013.

6 S. Angelopoulos. Further connections between contract-scheduling and ray-searching problems. In *Proc. of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1516–1522, 2015.

7 S. Angelopoulos, A. López-Ortiz, and K. Panagiotou. Multi-target ray searching problems. *Theoretical Computer Science*, 540:2–12, 2014.

8 G. Ausiello, S. Leonardi, and A. Marchetti-Spaccamela. On salesmen, repairmen, spiders, and other traveling agents. In *Algorithms and Complexity, 4th Italian Conference, CIAC 2000, Rome, Italy, March 2000, Proceedings*, pages 1–16, 2000. URL: `http://dx.doi.org/10.1007/3-540-46521-9_1`, `doi:10.1007/3-540-46521-9_1`.

9 R. Baeza-Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Information and Cmputation*, 106:234–244, 1993.

10 V. Baston and K. Kikuta. Search games on networks with travelling and search costs and with arbitrary searcher starting points. *Networks*, 62(1):72–79, 2013.

11 V. Baston and K. Kikuta. Search games on a network with travelling and search costs. *International Journal of Game Theory*, 44(2):347–365, 2015.

12 A. Beck. On the linear search problem. *Naval Research Logistics*, 2:221–228, 1964.

13 R. Bellman. An optimal search problem. *SIAM Review*, 5:274, 1963.

14 D.S. Bernstein, T. J. Perkins, S. Zilberstein, and L. Finkelstein. Scheduling contract algorithms on multiple processors. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*, pages 702–706, 2002.

15 P. Bose, J. De Carufel, and S. Durocher. Searching on a line: A complete characterization of the optimal solution. *Theoretical Computer Science*, 569:24–42, 2015.

16 A. Dagan and S. Gal. Network search games, with arbitrary searcher starting point. *Networks*, 52(3):156–161, 2008.

17 E.D. Demaine, S.P. Fekete, and S. Gal. Online searching with turn cost. *Theoretical Computer Science*, 361:342–355, 2006.

18 R. Fleischer, T. Kamphans, R. Klein, E. Langetepe, and G. Trippen. Competitive online approximation of the optimal search ratio. *SIAM Journal on Computing*, 38(3):881–898, 2008.

19 S. Gal. A general search game. *Israel Journal of Mathematics*, 12:32–45, 1972.

20 S. Gal. Minimax solutions for linear search problems. *SIAM J. on Applied Math.*, 27:17–30, 1974.

21 S. Gal. Search games with mobile and immobile hider. *SIAM Journal on Control and Optimization*, 17(1):99–122, 1979.

22 S. Gal. *Search Games.* Academic Press, 1980.

23 S. Gal. On the optimality of a simple strategy for searching graphs. *International Journal of Game Theory*, 29(4):533–542, 2001.

24 P. Jaillet and M. Stafford. Online searching. *Operations Research*, 49:234–244, 1993.

25 M-Y. Kao and M.L. Littman. Algorithms for informed cows. In *Proceedings of the AAAI 1997 Workshop on Online Search*, 1997.

26 M-Y. Kao, Y. Ma, M. Sipser, and Y.L. Yin. Optimal constructions of hybrid algorithms. *Journal of Algorithms*, 29(1):142–164, 1998.

27 M-Y. Kao, J.H. Reif, and S.R. Tate. Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem. *Inform. and Comp.*, 131(1):63–80, 1996.

28 D. G. Kirkpatrick. Hyperbolic dovetailing. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA)*, pages 616–627, 2009.

29 E. Koutsoupias, C.H. Papadimitriou, and M. Yannakakis. Searching a fixed graph. In *Proc. of the 23rd Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 280–289, 1996.

30 T. Lidbetter. Search games with multiple hidden objects. *SIAM Journal on Control and Optimization*, 51(4):3056–3074, 2013.

31 A. López-Ortiz and S. Schuierer. The ultimate strategy to search on m rays? *Theoretical Computer Science*, 261(2):267–295, 2001.

32 A. López-Ortiz and S. Schuierer. On-line parallel heuristics, processor scheduling and robot searching under the competitive framework. *Theor. Comp. Sci.*, 310(1–3):527–537, 2004.

33 A. McGregor, K. Onak, and R. Panigrahy. The oil searching problem. In *Proc. of the 17th European Symposium on Algorithms (ESA)*, pages 504–515, 2009.

34 L. Pavlovic. A search game on the union of graphs with immobile hider. *Naval Research Logistics*, 42(8):1177–1199, 1995.

35 J. Reijnierse and J. Potters. Search games with immobile hider. *International Journal of Game Theory*, 21:385–394, 1993.

36 S. Schuierer. A lower bound for randomized searching on m rays. In *Computer Science in Perspective*, pages 264–277, 2003.

Derandomizing Isolation Lemma for $K_{3,3}$-free and K_5-free Bipartite Graphs

Rahul Arora[1], Ashu Gupta[2], Rohit Gurjar*[3], and Raghunath Tewari[4]

1 **University of Toronto, Canada; and**
Indian Institute of Technology Kanpur, India
arorar@cs.toronto.edu, arorar@iitk.ac.in
2 **University of Illinois at Urbana-Champaign, USA**
agupta80@illinois.edu
3 **HTW Aalen, Germany**
rgurjar@iitk.ac.in
4 **Indian Institute of Technology Kanpur, India**
rtewari@iitk.ac.in

Abstract

The perfect matching problem has a randomized NC algorithm, using the celebrated Isolation Lemma of Mulmuley, Vazirani and Vazirani. The Isolation Lemma states that giving a random weight assignment to the edges of a graph ensures that it has a unique minimum weight perfect matching, with a good probability. We derandomize this lemma for $K_{3,3}$-free and K_5-free bipartite graphs. That is, we give a deterministic log-space construction of such a weight assignment for these graphs. Such a construction was known previously for planar bipartite graphs. Our result implies that the perfect matching problem for $K_{3,3}$-free and K_5-free bipartite graphs is in SPL. It also gives an alternate proof for an already known result – reachability for $K_{3,3}$-free and K_5-free graphs is in UL.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, G.2.2 Graph Theory

Keywords and phrases bipartite matching, derandomization, isolation lemma, SPL, minor-free graph

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.10

1 Introduction

The perfect matching problem is one of the most extensively studied problem in combinatorics, algorithms and complexity. In complexity theory, the problem plays a crucial role in the study of parallelization and derandomization. In a graph $G(V, E)$, a *matching* is a set of disjoint edges and a matching is called *perfect* if it covers all the vertices of the graph. Edmonds [12] gave the first polynomial time algorithm for the matching problem. Since then, there have been improvements in its sequential complexity [24], but an NC (efficient parallel) algorithm for it is not known. The perfect matching problem has various versions:

- Decision-PM: Decide if there exists a perfect matching in the given graph.
- Search-PM: Construct a perfect matching in the given graph, if it exists.

* Supported by TCS Research Fellowship.

© Rahul Arora, Ashu Gupta, Rohit Gurjar, and Raghunath Tewari;
licensed under Creative Commons License CC-BY

A randomized NC (RNC) algorithm for Decision-PM was given by [23]. Subsequently, Search-PM was also shown to be in RNC [18, 25]. The solution of Mulmuley et al. [25] was based on the powerful idea of *Isolation Lemma*. They defined a notion of an isolating weight assignment on the edges of a graph. Given a weight assignment on the edges, weight of a matching is defined to be the sum of the weights of all the edges in it.

▶ **Definition 1** ([25]). For a graph $G(V, E)$, a weight assignment $\mathbf{w}\colon E \to \mathbb{N}$ is isolating if G either has a unique minimum weight perfect matching according to w or has no perfect matchings.

The Isolation Lemma states that a random integer weight assignment (polynomially bounded) is isolating with a good probability. Other parts of the algorithm in [25] are deterministic. They showed that if we are given an isolating weight assignment (with polynomially bounded weights) for a graph G, then a perfect matching in G can be constructed in NC^2. Later, Allender et al. [2] showed that the Decision-PM would be in SPL, which is in NC^2, if an isolating weight assignment can be constructed in L (see also [9]). A language L is in the class SPL if its characteristic function $\chi_L\colon \Sigma^* \to \{0, 1\}$ can be (log-space) reduced to computing determinant of an integer matrix.

Derandomizing the Isolation Lemma remains a challenging open question. A general version of Isolation Lemma has also been studied, where one has to ensure a unique minimum weight set in a (non-explicitly) given family of sets (or multisets). Arvind and Mukhopadhyay [4] have shown that derandomizing this version of Isolation Lemma would imply circuit size lower bounds. While Reinhardt and Allender [26] have shown that derandomizing Isolation Lemma for some specific families of paths in a graph would imply $\mathsf{NL} = \mathsf{UL}$.

With regard to matchings, Isolation Lemma has been derandomized for some special classes of graphs: planar bipartite graphs [9, 29], constant genus bipartite graphs [10], graphs with small number of matchings [14, 1] and graphs with small number of nice cycles [15]. In a result subsequent to this work, Fenner et al. [13] achieved an almost complete derandomization of the isolation lemma for bipartite graphs. They gave a deterministic construction but with quasi-polynomially large weights. A graph G is bipartite if its vertex set can be partitioned into two parts V_1, V_2 such that any edge is only between a vertex in V_1 and a vertex in V_2. A graph is planar if it can be drawn on a plane without any edge crossings.

It is well known that a graph is planar if and only if it is both $K_{3,3}$-free and K_5-free [33]. For a graph H, G is an H-free graph if H is not a minor of G. $K_{3,3}$ is the complete bipartite graph with $(3, 3)$ nodes and K_5 is the complete graph with 5 nodes. A natural generalization of planar bipartite graphs would be $K_{3,3}$-free bipartite graphs or K_5-free bipartite graphs. We make a further step towards the derandomization of Isolation Lemma by derandomizing it for these two graph classes. Note that these graphs are not captured by the classes of graphs mentioned above. In particular, a $K_{3,3}$-free or K_5-free graph can have arbitrarily high genus, exponentially many matchings or exponentially many nice cycles.

▶ **Theorem 2.** *Given a $K_{3,3}$-free or K_5-free bipartite graph, an isolating weight assignment (polynomially bounded) for it can be constructed in log-space.*

Another motivation to study these graphs came from the fact that Count-PM (counting the number of perfect matchings) is in NC^2 for $K_{3,3}$-free graphs [31] and in TC^2 ($\subseteq \mathsf{NC}^3$) for K_5-free graphs [28]. These were the best known results for Decision-PM too. The counting results, together with the known NC-reduction from Search-PM to Count-PM (for bipartite graphs) [20], implied an NC algorithm for Search-PM. Thus, a natural

question was to find a direct algorithm for Search-PM via isolation, which we do here. One limitation of the earlier approach is that Count-PM is $\#\mathcal{P}$-hard for general bipartite graphs. Thus, there is no hope of generalizing this approach to work for all graphs. While the isolation approach can potentially lead to a solution for general/bipartite graphs.

Theorem 2 together with the results of Allender et al. [2] and Datta et al. [9] gives us the following results about matching.

▶ **Corollary 3.** *For a $K_{3,3}$-free or K_5-free bipartite graph,*

- Decision-PM *is in* SPL.
- Search-PM *is in* $\mathsf{FL}^{\mathsf{SPL}}$.
- Min-Weight-PM *is in* $\mathsf{FL}^{\mathsf{SPL}}$.

$\mathsf{FL}^{\mathsf{SPL}}$ is the set of function problems which can be solved by a log-space Turing machine with access to an SPL oracle. Like SPL, $\mathsf{FL}^{\mathsf{SPL}}$ also lies in NC^2. The problem Min-Weight-PM asks to construct a minimum weight perfect matching in a given graph with polynomially bounded weights on its edges.

The crucial property of these graphs, which we use, is that their 4-connected components are either planar or small sized. This property has been used to reduce various other problems on $K_{3,3}$-free or K_5-free graphs to their planar version, e.g. graph isomorphism [11], reachability [30]. However, their techniques do not directly work for the matching problem. There has been an extensive study on more general minor-free graphs by Robertson and Seymour. In a long series of works, they gave similar decomposition properties for these graphs [27]. Our approach for matching can possibly be generalized to H-free graphs for a larger/general graph H.

Our techniques. We start with the idea of Datta et al. [9] which showed that a skew-symmetric weight function on the edges ($w(u,v) = -w(v,u)$) such that every cycle has a nonzero circulation (weight in a fixed orientation) implies isolation of a perfect matching in bipartite graphs. To achieve nonzero circulation in a $K_{3,3}$-free or K_5-free graph, we work with its 3-connected or 4-connected component decomposition given by [33, 5], which can be constructed in log-space [30, 28]. The components are either planar or constant-sized and share a pair/triplet of vertices. These components form a *tree structure*, when each component is viewed as a node and there is an edge between two components if they share a pair/triplet. For any cycle C in the graph, we break it into its fragments contained within each of these components, which we call *projections* of C. Any such projection can be made into a cycle by adding virtual edges for separating pairs/triplets in the corresponding component.

Circulation of any cycle can be seen as a sum of circulations of its projections. The projections of a cycle can have circulations with opposite signs and thus, can cancel each other. To avoid this cancellation, we observe that the components, where a cycle has a non-empty projection form a subtree of the component tree. The idea is to assign edge weights using a different scale for each level of nodes in the tree. This ensures that for any subtree, its root node will contribute a weight higher than the total weight from all its other nodes. To avoid any cancellations within a component, weights in a component are given by modifying some known techniques for planar graphs [9, 19] and constant sized graphs.

This idea would work only if the component tree has a small depth, which might not be true in general. Thus, we create an $O(\log n)$-depth *working tree* by finding 'centers' for the component tree and its subtrees recursively. The construction of such a balanced working tree has been studied in context of evaluating arithmetic expressions [7]. In the literature,

this construction is also known as 'centroid decomposition' or 'recursive balanced separators'. Its log-space implementation is more involved.

As the working tree has $O(\log n)$ depth, the straightforward way of using a different scale for each level will lead to edge weights being $n^{O(\log n)}$. So instead, in a component node, we assign weights to only those edges which surround a separating pair/triplet. The weighting scheme ensures that the total weight grows only by a constant multiple, when we move one step higher in the working tree.

Achieving non-zero circulation in log-space also puts directed reachability in UL [26, 6, 29]. Thus, we get an alternate proof for the result – directed reachability for $K_{3,3}$-free and K_5-free graphs is in UL [30].

In Section 2, we introduce the concepts of nonzero circulation, clique-sum, graph decomposition and the corresponding component tree. In Section 3, we give a log-space construction of a weight assignment with nonzero circulation for every cycle, for a class of graphs defined via clique-sum operations on planar and constant-sized graphs. This class contains $K_{3,3}$-free and K_5-free graphs (a proof can be found in the full version of this paper [3]).

2 Preliminaries

Let us first define a skew-symmetric weight function on the edges of a graph. For this, we consider the edges of the graph directed in both directions. We call this directed set of edges $\vec{E}$. A weight function $w\colon \vec{E} \to \mathbb{Z}$ is called skew-symmetric if for any edge (u, v), $w(u, v) = -w(v, u)$.

▶ **Definition 4** (Circulation). For a cycle C, whose edges are given by $\{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k), (v_k, v_1)\}$, its circulation is defined to be $w(v_1, v_2) + w(v_2, v_3) + \cdots + w(v_k, v_1)$.

Clearly, as our weight function is skew-symmetric, changing the orientation of the cycle only changes the sign of the circulation. The following lemma [29, Theorem 6] gives the connection between nonzero circulations and isolation of a matching. For a bipartite (undirected) graph $G(V_1, V_2, E)$, a skew-symmetric weight function $w\colon \vec{E} \to \mathbb{Z}$ on its edges has a natural interpretation on the undirected edges as $\mathbf{w}\colon E \to \mathbb{Z}$ such that $\mathbf{w}(u, v) = w(u, v)$, where $u \in V_1$ and $v \in V_2$.

▶ **Lemma 5** ([29]). *Let $w\colon \vec{E} \to \mathbb{Z}$ be a skew-symmetric weight function on the edges of a bipartite graph G such that every cycle has a non-zero circulation. Then, $\mathbf{w}\colon E \to \mathbb{Z}$ is an isolating weight assignment for G.*

The bipartiteness assumption is needed only in the above lemma. We will construct a skew-symmetric weight function that guarantees nonzero circulation for every cycle, for a given $K_{3,3}$-free or K_5-free graph, i.e. without assuming bipartiteness.

2.1 Clique-sum

First, we will construct a nonzero circulation weight assignment for a special class of graphs, defined via a graph operation called *clique-sum*.

▶ **Definition 6** (Clique-sum). Let G_1 and G_2 be two graphs each containing a clique (of the same size). A clique-sum of graphs G_1 and G_2 is obtained from their disjoint union by identifying pairs of vertices in these two cliques to form a single shared clique, and by possibly deleting some of the edges in the clique. It is called a k-clique-sum if the cliques involved have at most k vertices.

One can form clique-sums of more than two graphs by a repeated application of clique-sum operation on two graphs. Using this, we define a new class of graphs. Let $\mathcal{P}_c$ be the class of all planar graphs together with all graphs of size at most c, where c is a constant. Define $\langle \mathcal{P}_c \rangle_k$ to be the class of graphs constructed by repeatedly taking k-clique-sums, starting from the graphs which belong to the class $\mathcal{P}_c$. In other words, it is the closure of $\mathcal{P}_c$ under k-clique sums. The starting graphs are called the component graphs. We will construct a nonzero circulation weight assignment for the graphs which belong to the class $\langle \mathcal{P}_c \rangle_3$.

Taking 1-clique-sum of two graphs will result in a graph which is not biconnected. For the perfect matching problem, one can assume without loss of generality that the given graph is biconnected (see the full version [3] for details). Thus, we assume that every clique-sum operation involves either 2-cliques or 3-cliques. A 2-clique which is involved in a clique-sum operation is called a separating pair. Similarly, a 3-clique is called a separating triplet. In general, they are called separating sets. Note that deletion of any separating pair/triplet will make the graph disconnected. We emphasize here that there can be other pairs/triplets in the graph which are not involved in a clique-sum operation, but whose deletion will make the graph disconnected. In this work, the term separating pair/triplet does not refer to such pairs/triplets.

2.2 Component Tree

In general, clique-sum operation can be performed many times using the same separating set. In other words, many components can share a separating set. One can modify a graph in class $\langle \mathcal{P}_c \rangle_3$ via some matching preserving operations such that on decomposition, any separating set is shared by only two components (see the full version [3] for details). Henceforth, in this section we assume this property.

Using this assumption, we can define a component graph for any graph $G \in \langle \mathcal{P}_c \rangle_3$ as follows: each component is represented by a node and two such nodes are connected by an edge if the corresponding components share a separating set. Observe that this component graph is actually a tree. This is because when we take repeated clique-sums, a new component can be attached with only one of the already existing components, as a clique will be contained within one component. In literature [16, 30], the component tree also contains a node for each separating set and it is connected by all the components which share this separating set. But, here we can ignore this node as we have only two sharers for each separating set.

In the component tree, each component is shown with all the separating sets it shares with other components. Thus, a copy of a separating set is present in both its sharer components. Moreover, in each component, a separating set is shown with a virtual clique, i.e., a virtual edge for a separating pair and a virtual triangle for a separating triplet. These virtual cliques represent the paths between the nodes via other components If any two vertices in a separating set have a real edge in G, then that real edge is drawn in one of the sharing components, parallel to the virtual edge. Note that while a vertex can have its copy in two components, any real edge is present in exactly one component.

3 Nonzero Circulation

In this section, we construct a nonzero circulation weight assignment for a given graph in the class $\langle \mathcal{P}_c \rangle_3$, provided that the component tree and the planar embeddings of the planar components are given. Moreover, to construct this weight assignment we will make some assumptions about the given graph and its component tree.

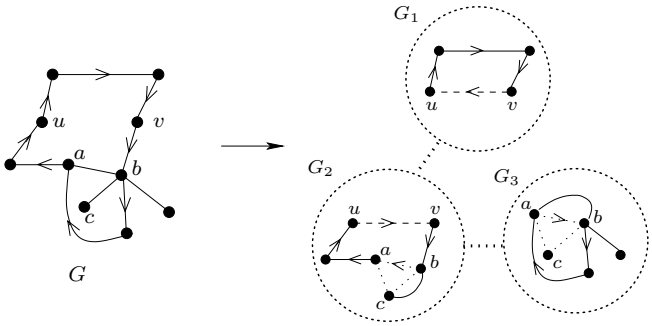


Figure 1 Breaking a cycle into its component cycles (projections) in the component tree. Notice that the original cycle and its components share the same set of *real* edges.

1. In any component, a vertex is a part of at most one separating set.
2. Each separating set is shared by at most two components.
3. Any virtual triangle in a planar component is always a face.

Given a $K_{3,3}$-free or K_5-free graph, a component tree can be constructed which has these properties (see the full version [3] for details). The third property comes naturally, as the inside and outside parts of any virtual triangle can be considered as different components sharing this separating triplet. All these constructions are in log-space.

3.1 Components of a cycle

We look at a cycle in the graph as *sum* of many cycles, one from each component the cycle passes through. Intuitively, the original cycle is *broken* at the separating set vertices which were part of the cycle, thereby generating fragments of the cycle in various nodes of the component tree. In all the component nodes containing these fragments, we include the virtual edges of the separating sets in question to complete the fragment into a cycle, thus resulting in component cycles in the component nodes (see Figure 1).

Consider a directed cycle $C = \{(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_0)\}$ in a graph $G = (V, E)$. Without loss of generality, consider that G is separated into two components G_1 and G_2 via a separating pair (v_i, v_0) or a separating triplet (v_i, v_0, u), where $1 \leq i < k$ and $u \in V$. Then, one of the components, say G_1, will contain the vertices $v_i, v_{i+1 \bmod k}, \ldots, v_{k-1}, v_0$, and the other ($G_2$) will contain the vertices $v_0, v_1, \ldots, v_{i-1}, v_i$. Then the cycles $C_1 = \{(v_i, v_{i+1 \bmod k}), \ldots, (v_{k-1}, v_0), (v_0, v_i)\}$ and $C_2 = \{(v_0, v_1), \ldots, (v_{i-1}, v_i), (v_i, v_0)\}$ in G_1 and G_2 respectively are the component cycles of C, and we say that C is the sum of C_1 and C_2. Observe that the edges (v_i, v_0) and (v_0, v_i) are virtual.

Repeat the processes recursively for C_1 and C_2 until no separating set breaks a cycle component, and we get the component cycles of the cycle C. Note that any edge in a cycle C is contained in exactly one of its component cycles. Moreover for any component cycle, all its edges, other than the virtual edges, are contained in C.

Observe that for any separating set in a component, a cycle can use one of its vertices to go out of the component and another vertex to come in (this transition is represented by a virtual edge in the component). As any separating set has size at most 3, a cycle can visit a node of the component tree only once. In other words, a cycle can have only one component cycle in any component tree node (this would not be true if we had separating sets of size 4). Also, a component cycle can take only one edge of any virtual triangle.

▶ **Definition 7** (Projection of a cycle)**.** For a given component node N in the component tree, the component cycle of a cycle C in N is called the projection of C on N. If there is no component cycle of C in N, then C is said to have an empty projection on N.

It is easy to see that for any cycle C, the components on which C has a non-empty projection, form a subtree of the component tree. To construct the weight assignment (Section 3.2), we will work with the component nodes of the component tree. Within any component, weight of a virtual edge will always be set to zero. Along with the fact that each cycle has the same set of real edges as the union of the edges in all its projections, this leads to the following lemma.

▶ **Lemma 8.** *The circulation of a cycle is the sum of circulations of its component cycles.*

Note that for a cycle, its component cycles can have circulations with different signs (positive or negative) as they can have different orientations (clockwise or anti-clockwise) in the planar components. Hence the total circulation can potentially be zero. Our idea is to ensure that one of the component cycles get a circulation greater than all the other component cycles put together. This will imply a nonzero circulation.

3.2 Weighting Scheme

The actual weight function we employ is a combination of two weight functions w_0 and w_1. They are combined with an appropriate scaling so that they do not interfere with each other. w_1 ensures that all the cycles which are within one component have a non-zero circulation and w_0 ensures that all the cycles which project on at least two components have a non-zero circulation. We first describe the construction of w_0.

Working Tree: The given component tree can have arbitrary depth, while our weight construction would need the tree-depth to be $O(\log n)$. Thus, we re-balance the tree to construct a new *working tree*. It is a rooted tree which has the same nodes as the component tree, but the edge relations are different. The working tree, in some sense, 'preserves' the subtree structure of the original tree.

For a tree S, its working tree $\mathsf{wt}(S)$ is constructed as follows: Find a 'center' node $c(S)$ in the tree S and mark it as the root of the working tree, $r(\mathsf{wt}(S))$. Deleting the node $c(S)$ from the tree S would give a set of disjoint trees, say $\{S_1, S_2, \ldots, S_k\}$. Apply this procedure recursively on these trees to construct their working trees $\mathsf{wt}(S_1), \mathsf{wt}(S_2), \ldots, \mathsf{wt}(S_k)$. Connect each $\mathsf{wt}(S_i)$ to the root $r(\mathsf{wt}(S))$, as a subtree. In other words, $r(\mathsf{wt}(S_i))$ is a child of $r(\mathsf{wt}(S))$. For the base case, when the tree is a node, its working tree is the node itself. This completes the construction. If the component $c(S)$ shares the separating set τ_i with S_i, then the subtree $\mathsf{wt}(S_i)$ is said to be attached to the root $r(\mathsf{wt}(S))$ at τ_i.

The 'center' nodes are chosen in a balanced way so that the working tree depth is $O(\log n)$. Von Braunmühl and Verbeek [32], and later Limaye et al. [21], gave a log-space construction of such a balanced tree, but in terms of well-matched strings (also see [8]). In Section 3.3, we present the log-space construction in terms of a tree, along with a precise definition of a 'center' node.

Note that for any two nodes $v_1 \in S_i$ and $v_2 \in S_j$ such that $i \neq j$, $\mathrm{path}(v_1, v_2)$ in S passes through the node $c(S) = r(\mathsf{wt}(S))$. Thus, we get the following property for the working tree.

▶ **Claim 9.** *For any two nodes $u, v \in S$, let their least common ancestor in the working tree $\mathsf{wt}(S)$ be the node a. Then $\mathrm{path}(u, v)$ in the tree S passes through a.*

The root $r(\mathsf{wt}(S))$ of the working tree $\mathsf{wt}(S)$ is said to be at depth 0. For any other node in $\mathsf{wt}(S)$, its depth is defined to be one more than the depth of its parent. Henceforth, depth of a node will always mean its depth in the working tree. From Claim 9, we get the following.

▶ **Claim 10.** *Let S' be an arbitrary subtree of S, with its set of nodes being $\{v_1, v_2, \ldots, v_k\}$. There exists $i^* \in \{1, 2, \ldots, k\}$ such that for any $j \in [k]$ with $j \neq i^*$, v_j is a descendant of v_{i^*} in the working tree $\mathsf{wt}(S)$.*

Proof. Let d^* be the minimum depth of any node in S', and let v_{i^*} be a node in S' with depth d^*. We claim that every other node in S' is a descendant of v_{i^*} in the working tree $\mathsf{wt}(S)$. For the sake of contradiction, let there be a node $v_j \in S'$ which is not a descendant of v_{i^*}. Then, the least common ancestor of v_j and v_{i^*} in $\mathsf{wt}(S)$ must have depth strictly smaller than d^*. By Claim 9, this least common ancestor must be present in the tree S'. But, we assumed d^* is the minimum depth value in S'. Thus, we get a contradiction. ◀

This claim plays a crucial role in our weight assignment construction, as for any cycle C the components with a non-empty projection of C form a subtree of the component tree. To assign weights in the graph, we work with the working tree of its component tree. Let the working tree be $\mathcal{T}$. We start by assigning weight to the nodes having the largest depth, and move up till we reach depth 0, that is, the root node $r(\mathcal{T})$. The idea is that for any cycle C, its unique least-depth projection should get a circulation higher than the total circulation of all its other projections.

Complementary to the depth, we also define *height* of every node in the working tree. Let the maximum depth of any node in the working tree be D. Then, the height of a node is defined to be the difference between its depth and $D + 1$.

Circulations of cycles spanning multiple components. For any subtree T of the working tree $\mathcal{T}$, the weights to the edges inside the component $r(T)$ will be given by two different schemes depending on whether the corresponding graph is planar or constant sized.

Let the maximum possible number of edges in a constant sized component be m. Then, let K be a constant such that $K > \max(2^{m+2}, 7)$. Also, suppose that the height of a node N is given by the function $h(N)$, and the number of leaves in subtree T is given by $l(T)$. Lastly, suppose the set of subtrees attached at $r(T)$ is $\{T_1, T_2, \ldots, T_k\}$.

Constant sized graph: Let the set of (real) edges of the graph be $\{e_1, e_2, \ldots, e_m\}$. The edge e_j will be given weight $2^j \times K^{h(r(T))-1} \times l(T)$ for an arbitrarily fixed direction. The intuition behind this scheme is that powers of 2 ensure that sum of weights for any non-empty subset of edges remain nonzero even when they contribute with different signs.

Planar graph: We work with a given planar embedding of the graph. For any weight assignment $w : \vec{E} \to \mathbb{Z}$ on the edges of the graph, we define the *circulation of a face* as the circulation of the corresponding cycle in the clockwise direction, i.e., traverse the boundary edges of the face in the clockwise direction and take the sum of their weights. Instead of directly assigning edge weights, we will fix circulations for the inner faces of the graph. As we will see later, fixing positive circulations for all inner faces will avoid any cancellations. Lemma 14 describes how to assign weights to the edges of a planar graph to get the desired circulation for each of the inner faces.

Assigning circulations to the faces: Here, only those inner faces are assigned nonzero circulations which are adjacent to some separating pair/triplet shared with a subtree. This is a crucial idea. As we will see in Lemma 11, this ensures that the maximum possible circulation of a cycle grows only by a constant multiple as we move one level higher up in the working tree.

If T is a singleton, i.e., there are no subtrees attached at T, we give a zero circulation to all the faces (and thus zero weight to all the edges) of $r(T)$. Otherwise, consider a separating pair $\{a, b\}$ where a subtree T_i is attached to $r(T)$. The two faces adjacent to the virtual edge (a, b) will be assigned circulation $2 \times K^{h(r(T_i))} \times l(T_i)$. Similarly, consider a triplet $\{a, b, c\}$ where a subtree T_j is attached. Then all the faces (at most 3) adjacent to the virtual triangle $\{a, b, c\}$ get circulation $2 \times K^{h(r(T_j))} \times l(T_j)$. Repeat this procedure for all the faces adjacent to any pairs and/or triplets where subtrees are attached. If a face is adjacent to more than one virtual edge/triangle, then we just take the sum of different circulations due to each virtual edge/triangle.

Recall that by definition, each face has a positive circulation in the clockwise direction. The intuition behind this scheme is the following: circulation of any cycle in the planar component is just the sum of circulations of the faces inside it (Claim 12). As all of them have the same sign, they cannot cancel each other. Moreover, it will be ensured that the contribution to the circulation from this planar component is higher than the total contribution from all its subtrees, and thus, cannot be canceled.

Now, we formally show that this weighting scheme ensures that all the cycles spanning multiple components in the tree get non-zero circulation.

Nonzero Circulation of a cycle. First, we derive an upper bound on the circulation of any cycle completely contained in a subtree T of the working tree.

▶ **Lemma 11.** *The upper bound on the circulation of any cycle contained in a subtree T of the working tree $\mathcal{T}$ is $U_T = K^{h(r(T))} \times l(T)$.*

Proof. We prove this using induction on the height of $r(T)$.

Base case: The height of $r(T)$ is 1. Notice that this means that $r(T)$ has the maximum depth amongst all the nodes in $\mathcal{T}$, and therefore, $r(T)$ is a leaf node, and T is a singleton. Consider the two cases: i) when $r(T)$ is a planar graph, ii) when it is a constant sized graph.

By our weight assignment, if $r(T)$ is planar, the total weight of all the edges is zero. On the other hand, if $r(T)$ is a constant sized graph, the maximum circulation of a cycle is the sum of weights of its edges, that is, $\sum_{i=1}^{m}(K^0 \times 1 \times 2^i) < 2^{m+1} \leq K$. Thus, the circulation is upper bounded by $K^{h(r(T))} \times l(T)$ (as $l(T) = 1$).

Induction hypothesis: For any tree T' with $h(r(T')) \leq j - 1$, the upper bound is $U_{T'} = K^{h(r(T'))} \times l(T')$.

Induction step: We will prove that for any tree T with $h(r(T)) = j$, the upper bound is $U_T = K^{h(r(T))} \times l(T)$.

Let the subtrees attached at $r(T)$ be $\{T_1, T_2, \ldots, T_k\}$. For any cycle in T, sum of the circulations of its projections on the subtrees $T_1, T_2, \ldots, T_k$ can be at most $\sum_{i=1}^{k} U_{T_i}$.

First, we handle the case when $r(T)$ is planar. For any subtree T_i, the total circulation of faces in $r(T)$ due to connection to T_i can be $6 \times K^{h(r(T_i))} \times l(T_i)$. This is because the circulation of each face adjacent to the separating set connecting with T_i is $2 \times K^{h(r(T_i))} \times l(T_i)$, and there can be at most 3 such faces. Thus,

$$U_T = \sum_{i=1}^{k} U_{T_i} + \sum_{i=1}^{k} \left(6 \times K^{h(r(T_i))} \times l(T_i)\right)$$

$$
\begin{aligned}
&= \sum_{i=1}^{k} \left(K^{h(r(T_i))} \times l(T_i) \right) + \sum_{i=1}^{k} \left(6 \times K^{h(r(T_i))} \times l(T_i) \right) \\
&= 7 \times K^{h(r(T))-1} \times \sum_{i=1}^{k} l(T_i) \qquad (\because \forall i,\ h(r(T_i)) = h(r(T)) - 1) \\
&< K^{h(r(T))} \times \sum_{i=1}^{k} l(T_i) \qquad (\because K > 7) \\
&= K^{h(r(T))} \times l(T)
\end{aligned}
$$

Now, consider the case when $r(T)$ is a small non-planar graph. The maximum possible contribution from edges of $r(T)$ to the circulation of a cycle in T is less than $2^{m+1} \times K^{h(r(T))-1} \times l(T)$. Similar to the case when $r(T)$ is planar, contribution from all subtrees is at most $K^{h(r(T))-1} \times l(T)$. The total circulation of a cycle in T can be at most the sum of these two bounds, and is thus bounded above by $(2^{m+1}+1) \times K^{h(r(T))-1} \times l(T)$. Since, $K > 2^{m+2}$, the total possible circulation is less than $K^{h(r(T))} \times l(T)$.

Therefore, the upper bound $U_T = K^{h(r(T))} \times l(T)$. ◀

To see that each cycle gets a nonzero circulation, recall Lemma 8, which says that the circulation of the cycle is the sum of circulations of its projections on different components. Consider a cycle C. Recall that components with a non-empty projection of C form a subtree S_C in the component tree. From Claim 10, we can find a node $v^* \in S_C$ such that all other nodes in S_C are its descendants in the working tree $\mathcal{T}$. Thus, v^* is the unique minimum depth component on which C has a non-empty projection. Now, we show two things: (i) the contribution to the circulation from this component is nonzero, and (ii) it is larger than sum of all the circulation contributions from all its subtrees in the working tree.

Let v^* be the root of a subtree T in the working tree. Let the subtrees attached at $r(T)$ $(= v^*)$ be $\{T_1, T_2, \ldots, T_k\}$ and the separating sets in $r(T)$ at which they are attached be $\{\tau_1, \tau_2, \ldots, \tau_k\}$ respectively.

Case 1: when $r(T)$ is a constant-sized component. It is easy to see that the circulation of any cycle in this component will be nonzero as long as it takes a real edge, because the weights given are powers of 2. Also, the minimum weight of any edge in $r(T)$ is $2 \times \sum_{i=1}^{k} U_{T_i}$. Thus, when a cycle takes a real edge, contribution to its circulation from $r(T)$ is larger than the contribution from higher depth components (components in the subtrees attached at $r(T)$). And any cycle has to take a real edge, as the virtual edges and triangles all have disjoint set of vertices. (Here, the virtual triangle does not count as a cycle).

Case 2: when $r(T)$ is a planar component. The crucial observation here is that in a planar graph, all the faces inside a cycle contribute to its circulation in the same orientation.

▶ **Claim 12** ([6]). *In a planar graph, circulation of a cycle in clockwise orientation is the sum of circulations of the faces inside it (a proof can be found in [3]).*

Since C passes through at least one of the subtrees attached at $r(T)$, say T_i, it must go through the separating set τ_i. Hence, the projection of C in $r(T)$, say C', must use the virtual edge (or one of the edges in the virtual triangle) corresponding to τ_i. This would imply that at least one of the faces adjacent to τ_i is inside C'. This is true for any subtree T_i which C passes through. As the faces adjacent to separating sets have nonzero circulations and each face has a positive circulation in clockwise direction, the circulation of C' is nonzero.

Recall that circulation of any face adjacent to τ_i is $2U_{T_i}$, where U_{T_i} is the upper bound on circulation contribution from T_i. This implies that the circulation of C' will surpass the

total circulation from all the subtrees which C passes through. Thus, we can conclude the following.

▶ **Lemma 13.** *Circulation of any cycle which passes through at least two components is nonzero.*

Face circulations using edge weights: Now, we come back to the question of assigning weights to the edges in a planar component such that the faces get the desired circulations. Lemma 14 describes this procedure for any planar graph.

▶ **Lemma 14** ([19]). *Let $G(V, E)$ be a planar graph with F being its set of inner faces in some planar embedding. For any given function on the inner faces $w' : F \to \mathbb{Z}$, a skew-symmetric weight function $w \colon \vec{E} \to \mathbb{Z}$ can be constructed in log-space such that each face $f \in F$ has a circulation $w'(f)$ (a proof can be found in the full version [3]).*

This scheme can assign weight to any edge in the given graph, while we are not allowed to give weights to virtual edges/triangles. So, we first collapse all the virtual triangles to one node and all the virtual edges to one node. As no two virtual triangles/edges are adjacent, after this operation, every face remains a non-trivial face (except the virtual triangle face). Now, we apply the procedure from Lemma 14. After undoing the collapse, the circulations of the faces will not change and we will have the desired circulations.

Circulation of cycles contained within a single component: To construct w_1 for planar components, we assign $+1$ circulation to every face using Lemma 14 (similar to the case of multiple components). This would ensure nonzero circulation for every cycle within the planar component. This construction has been used in [19] for bipartite planar graphs. [29] also gives a log-space construction which ensures nonzero circulation for all cycles in a planar graph, using Green's theorem.

For the non-planar components, w_0 already ensures that each cycle has non-zero circulation. Therefore, we set $w_1 = 0$. Use a linear combination of w_0 and w_1 such that they do not interfere with each other. Such a combination is easy to achieve by multiplying w_1 by n^2 or a higher power of n since since w_0 is $O(n)$. This together with Lemma 13 gives us the following.

▶ **Lemma 15.** *Circulation of any cycle is non-zero.*

Complexity: The weights given by this scheme are polynomially bounded and the weight-construction procedure can be done in log-space (see the full version [3] for details).

3.3 Construction of the Working Tree

Now, we describe a log-space construction of the working tree. The idea is obtained from the construction of [21, Lemma 6], where they create a $O(\log n)$-depth tree of well-matched substrings of a given well-matched string. Recall that for a tree S, the working tree $\mathsf{wt}(S)$ is constructed by first choosing a center node $c(S)$ of S and marking it as the root of $\mathsf{wt}(S)$, and then recursively finding the working trees for each component obtained by removing the node $c(S)$ from S and connecting them to the root of $\mathsf{wt}(S)$, as subtrees.

First, consider the following possible definition of the center: for any tree S with n nodes, one can define its center to be a node whose removal would give disjoint components of size $\leq 1/2|S|$. Finding such a center is an easy task and can be done in log-space. Clearly, the

depth of the working tree would be $O(\log n)$. It is not clear if the recursive procedure of finding centers for each resulting component can be done in log-space. Therefore, we give a more involved way of defining centers, so that the whole recursive procedure can be done in log-space.

First, we make the tree S rooted at an arbitrary node r. To find the child-parent relations of the rooted tree, one can do the standard log-space traversal of a tree.

Tree traversal [22] : for every node, give its edges an arbitrary cyclic ordering. Start traversing from the root r by taking an arbitrary edge. If you arrive at a node u using its edge e then leave node u using the right neighbor of e. This traversal ends at r with every edge being traversed exactly twice.

For any node v, let S_v denote the subtree of S, rooted at v. For any node v and one of its descendant nodes v' in S, let $S_{v,v'}$ denote the tree $S_v \setminus S_{v'}$. Moreover $S_{v,\epsilon}$ would just mean S_v, for any v. With our new definition of the center, at any stage of the recursive procedure, the component under consideration will always be of the form $S_{v,v'}$, for some nodes $v, v' \in S$. Now, we give a definition of the center for a rooted tree of the form $S_{v,v'}$.

Center $c(S_{v,v'})$:

Case (i) When $v' = \epsilon$, i.e. the given tree is S_v. Let c be a node in S_v, such that its removal gives components of size $\leq 1/2|S_v|$. If there are more than one such nodes then choose the lexicographically smallest one (there is at least one such center [17]). Define c as the center of $S_{v,v'}$.

Let the children of c in S_v be $\{c_1, c_2, \ldots, c_k\}$. Clearly, after removing c from S_v, the components we get are $S_{c_1}, S_{c_2}, \ldots, S_{c_k}$ and $S_{v,c}$. Thus, they are all of the form described above, and have size $\leq 1/2|S_v|$.

Case (ii) When v' is an actual node in S_v. Let the node sequence on the path connecting v and v' be $(u_0, u_1, \ldots, u_p)$, with $u_0 = v$ and $u_p = v'$. Let $0 \leq i < p$ be the least index such that $|S_{u_{i+1},v'}| \leq 1/2|S_{v,v'}|$. This index exists because $|S_{u_p,v'}| = 0$. Define u_i as the center of $S_{v,v'}$.

Let the children of u_i, apart from u_{i+1}, be $\{c_1, c_2, \ldots, c_k\}$. After removal of u_i from $S_{v,v'}$, the components we get are $S_{c_1}, S_{c_2}, \ldots, S_{c_k}$, $S_{u_{i+1},v'}$ and S_{v,u_i}. By the choice of i, $|S_{u_i,v'}| > 1/2|S_{v,v'}|$. Thus, $|S_{v,u_i}| \leq 1/2|S_{v,v'}|$. So, the only components for which we do not have a guarantee on their sizes, are $S_{c_1}, S_{c_2}, \ldots, S_{c_k}$. Observe that when we find a center for the tree $S_{c_j,\epsilon}$ in the next recursive call, it will fall into case (i) and the components we get will have their sizes reduced by a factor of $1/2$.

Thus, we can conclude that in the recursive procedure for constructing the working tree, we reduce the size of the component by half in at most two recursive calls. Hence, the depth of working tree is $O(\log n)$.

Now, we describe a log-space procedure for the working tree.

▶ **Lemma 16.** *For any tree S, its working tree $\mathsf{wt}(S)$ can be constructed in log-space.*

Proof. We just describe a log-space procedure for finding the parent of a given node x in the working tree. Running this procedure for every node will give us the working tree.

Find the center of the tree S. Removing the center would give many components. Find the component S_1, to which the node x belongs. Apply the same procedure recursively on S_1. Keep going to smaller components which contain x, till x becomes the center of some component. The center of the previous component in the recursion will be the parent of x in the working tree.

In this recursive procedure, to store the current component $S_{v,v'}$, we just need to store two nodes v and v'. Apart from these, we need to store center of the previous component and size of the current component.

To find the center of a given component $S_{v,v'}$, go over all possibilities of the center, depending on whether v' is ϵ or a node. For any candidate center c, find the sizes of the components generated if c is removed. Check if the sizes satisfy the specified requirements. Any of these components is also of the form $S_{u,u'}$ and thus can be stored with two nodes.

By the standard log-space traversal of a tree, for any given tree $S_{v,v'}$, one can count the number of nodes in it and test membership of a given node. Thus, the whole procedure works in log-space. ◀

4 Discussion

One of the open problems is to construct a polynomially bounded isolating weight assignment for a more general class of graphs, in particular, for all bipartite graphs. Our approach does not directly extend to more general minor-free graphs, because their decomposition can involve separating sets of size more than 3. For example, when we have a separating set of size 4, a cycle can have two different projections in a component, i.e., it enters the component twice and leaves the component twice. These two projections can contribute to the total circulation with opposite signs and can cancel each other.

The isolation question is also open for general planar graphs and small genus bipartite graphs.

Acknowledgements. We thank Arpita Korwar for various helpful discussions.

References

1 Manindra Agrawal, Thanh Minh Hoang, and Thomas Thierauf. The polynomially bounded perfect matching problem is in NC^2. In Wolfgang Thomas and Pascal Weil, editors, *STACS 2007*, volume 4393 of *Lecture Notes in Computer Science*, pages 489–499. Springer Berlin Heidelberg, 2007.

2 Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting uniform and nonuniform upper bounds. *J. Comput. Syst. Sci.*, 59(2):164–181, 1999.

3 Rahul Arora, Ashu Gupta, Rohit Gurjar, and Raghunath Tewari. Derandomizing isolation lemma for $K_{3,3}$-free and K_5-free bipartite graphs. Technical Report TR14-161, Electronic Colloquium on Computational Complexity (ECCC), 2014.

4 V. Arvind and Partha Mukhopadhyay. Derandomizing the isolation lemma and lower bounds for circuit size. In Ashish Goel, Klaus Jansen, JoséD.P. Rolim, and Ronitt Rubinfeld, editors, *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, volume 5171 of *Lecture Notes in Computer Science*, pages 276–289. Springer Berlin Heidelberg, 2008.

5 Takao Asano. An approach to the subgraph homeomorphism problem. *Theoretical Computer Science*, 38(0):249 – 267, 1985.

6 Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Trans. Comput. Theory*, 1(1):4:1–4:17, February 2009.

7 Richard P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, April 1974.

8 Bireswar Das, Samir Datta, and Prajakta Nimbhorkar. Log-space algorithms for paths and matchings in k-trees. *Theory of Computing Systems*, 53(4):669–689, 2013.

9 Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory of Computing Systems*, 47:737–757, 2010.

10 Samir Datta, Raghav Kulkarni, Raghunath Tewari, and N. V. Vinodchandran. Space complexity of perfect matching in bounded genus bipartite graphs. *J. Comput. Syst. Sci.*, 78(3):765–779, 2012.

11 Samir Datta, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Graph isomorphism for $K_{3,3}$-free and K_5-free graphs is in log-space. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, pages 145–156, 2009.

12 Jack Edmonds. Path, trees, and flowers. *Canadian J. Math.*, 17:449–467, 1965.

13 Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-nc. Technical Report TR15-177, Electronic Colloquium on Computational Complexity (ECCC), 2015.

14 Dima Grigoriev and Marek Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in NC (extended abstract). In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 166–172, 1987.

15 Thanh Minh Hoang. On the matching problem for special graph classes. In *IEEE Conference on Computational Complexity*, pages 139–150. IEEE Computer Society, 2010.

16 John E. Hopcroft and Robert Endre Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973.

17 Camille Jordan. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathematik*, 70:185–190, 1869.

18 Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

19 Arpita Korwar. Matching in planar graphs. Master's thesis, Indian Institute of Technology Kanpur, 2009.

20 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.

21 Nutan Limaye, Meena Mahajan, and B.V.Raghavendra Rao. Arithmetizing classes around NC_1 and l. In *STACS 2007*, volume 4393 of *Lecture Notes in Computer Science*, pages 477–488. Springer Berlin Heidelberg, 2007.

22 Steven Lindell. A logspace algorithm for tree canonization (extended abstract). In *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 400–404, New York, NY, USA, 1992. ACM.

23 László Lovász. On determinants, matchings, and random algorithms. In *FCT*, pages 565–574, 1979.

24 Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{V}E)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, SFCS '80, pages 17–27, Washington, DC, USA, 1980. IEEE Computer Society.

25 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987.

26 Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, 2000.

27 Neil Robertson and P.D Seymour. Graph minors. xvi. excluding a non-planar graph. *Journal of Combinatorial Theory, Series B*, 89(1):43 – 76, 2003.

28 Simon Straub, Thomas Thierauf, and Fabian Wagner. Counting the number of perfect matchings in K_5-free graphs. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 66–77, 2014.

29 Raghunath Tewari and N. V. Vinodchandran. Green's theorem and isolation in planar graphs. *Inf. Comput.*, 215:1–7, 2012.

30 Thomas Thierauf and Fabian Wagner. Reachability in $K_{3,3}$-free and K_5-free graphs is in unambiguous logspace. *Chicago J. Theor. Comput. Sci.*, 2014, 2014.

31 Vijay V. Vazirani. NC algorithms for computing the number of perfect matchings in $K_{3,3}$-free graphs and related problems. *Information and Computing*, 80(2):152–164, 1989.

32 Burchard von Braunmühl and Rutger Verbeek. Input-driven languages are recognized in log n space. In Marek Karpinski, editor, *Foundations of Computation Theory*, volume 158 of *Lecture Notes in Computer Science*, pages 40–51. Springer Berlin Heidelberg, 1983.

33 Klaus Wagner. Über eine Eigenschaft der ebenen Komplexe. *Math. Ann.*, 114, 1937.

Entropy Games and Matrix Multiplication Games*

Eugene Asarin[1], Julien Cervelle[2], Aldric Degorre[3], Cătălin Dima[4], Florian Horn[5], and Victor Kozyakin[6]

1 **IRIF, University Paris Diderot and CNRS, France**
2 **LACL, University Paris-Est Créteil, France**
3 **IRIF, University Paris Diderot and CNRS, France**
4 **LACL, University Paris-Est Créteil, France**
5 **IRIF, University Paris Diderot and CNRS, France**
6 **IITP, Russian Academy of Science, Russia**

Abstract

Two intimately related new classes of games are introduced and studied: entropy games (EGs) and matrix multiplication games (MMGs). An EG is played on a finite arena by two-and-a-half players: Despot, Tribune and the non-deterministic People. Despot wants to make the set of possible People's behaviors as small as possible, while Tribune wants to make it as large as possible. An MMG is played by two players that alternately write matrices from some predefined finite sets. One wants to maximize the growth rate of the product, and the other to minimize it. We show that in general MMGs are undecidable in quite a strong sense. On the positive side, EGs correspond to a subclass of MMGs, and we prove that such MMGs and EGs are determined, and that the optimal strategies are simple. The complexity of solving such games is in $\mathsf{NP} \cap \mathsf{coNP}$.

1998 ACM Subject Classification F.1.1 Models of Computation, F.2.1 Numerical Algorithms and Problems

Keywords and phrases game theory, entropy, joint spectral radius

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.11

1 Introduction

In recent years, some of us have been working on a new non-probabilistic quantitative approach to classical models in computer science based on the notion of language entropy (growth rate). This approach has produced new insights about timed automata and languages [1] as well as temporal logics [2]. In this article, we apply it to game theory and obtain a new natural class of games that we call *entropy games* (EGs). Such a game is played on a finite arena in a turn-based way, in infinite time, by two-and-a-half[1] players: *Despot*, *Tribune* and the non-deterministic *People*. Whenever Despot and Tribune decide on their strategies σ and τ, it leaves a set $L(\sigma, \tau)$ (an ω-language) of possible behaviors of People. Despot wants $L(\sigma, \tau)$ to be as small as possible, while Tribune wants to make this language as large as possible. Formally the payoff of the game is the entropy of $L(\sigma, \tau)$, with Despot minimizing and Tribune maximizing this value.

* The support of Agence Nationale de la Recherche under the project EQINOCS (ANR-11-BS02-004) is gratefully acknowledged. The results of Section 4 were obtained at the Institute for Information Transmission Problems, Russian Academy of Science, by V. Kozyakin at the expense of the Russian Science Foundation (project 14-50-00150).

[1] Although this term is mostly used for stochastic games, it is also an appropriate description of EGs.

© Eugene Asarin, Julien Cervelle, Aldric Degorre, Cătălin Dima, Florian Horn, and Victor Kozyakin;
licensed under Creative Commons License CC-BY

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 11; pp. 11:1–11:14

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

Potentially these games can be used to model hidden channel capacity problems in computer security, where the aim of the security policy (Despot) is to minimize the information flow whatever the environment (Tribune) does. EGs can also be rephrased in terms of population dynamics, where one player aims to maximize the population growth rate, while the other minimizes it; applications of this setting to medicine, ecology, and computer security (virus propagation) are still to be explored. On the theoretical side, well-known mean-payoff games on finite graphs can be seen as a subclass of our EGs. However the purpose of this paper is to explore the theoretical setting of EGs, we therefore leave applications and identification of relevant subclasses of EGs for further work.

The second class of objects studied is that of *matrix multiplication games* (MMGs), which came naturally when analyzing EGs and is, in our opinion, novel and interesting on its own. In such a game, two players, Adam and Eve, each possess a set of matrices, $\mathscr{A}$ and $\mathscr{E}$, respectively. The game is played in a turn-based way, in infinite time. At every turn, the player writes a matrix from his or her set. Adam wants the norm of the product of matrices $A_1E_1A_2E_2\dots$ obtained to be as small as possible (in the limit), while Eve wants it to be as large as possible. Formally, the payoff is the growth rate of the norm of the product.

The main interest of MMGs comes from the observation that, in the case when one of the two players is trivial (i.e. his or her set contains only the identity matrix), the game turns into the classical, important, and difficult, problem of computing the joint spectral radius or the joint spectral subradius of a set of matrices, see [22, 15]. Thus, MMGs is a game (or alternating) generalization of this problem. It is thus unsurprising that, in the general case, MMGs are even more difficult to analyze. We prove that several natural problems for MMGs are undecidable, in particular it is impossible to distinguish between games with value 0 and 1 (and thus it is impossible to approximate the value of an MMG).

Fortunately, MMGs have tractable subclasses. We reduce EGs to a particular subclass of MMGs (referred to as IMMGs), when the sets $\mathscr{A}$ and $\mathscr{E}$ are so-called *independent row uncertainty sets* of non-negative matrices [5], and show that for this class the game can be solved: it is determined, and for each player the optimal strategy is to write one and the same matrix at every turn. This result is based on a new, quite technical, *minimax* theorem on the spectral radius of products of the type AB where both A and B belong to sets of matrices with independent row uncertainties. We deduce that EGs are determined, and that the optimal strategies for Despot and Tribune are positional. A careful complexity analysis of the games considered (EGs and IMMGs) allows to prove that comparing their value to a rational constant can be done with complexity $\mathsf{NP} \cap \mathsf{coNP}$.

The article is structured as follows. In Sect. 2 we recall useful notions from linear algebra and language theory. In Sect. 3 we formally define the two games and show how they are related, we also prove undecidability of general MMGs. In Sect. 4 we prove the key technical minimax theorem for matrices. In Sect. 5 we prove the main properties of EGs and IMMGs: determinacy, existence of simple strategies and complexity bounds. In Sect. 6 we relate the EGs studied here to classical mean-payoff games and a new kind of population games. We conclude with a discussion on the perspectives. Proofs of all lemmas can be found in [3].

2 Preliminaries

2.1 Some Linear Algebra

Given two vectors $x, y \in \mathbb{R}^N$, we write $x \geqslant y$, if $x_i \geqslant y_i$ for each $1 \leqslant i \leqslant N$. Similar notation will be applied to matrices. We denote by $\|\cdot\|$ the 1-norm of vectors and matrices. Note that, for non-negative vectors and matrices, $\|x\| = \sum_i x_i$.

Let A be an $(N \times N)$-matrix. Its *spectral radius* is defined as the maximal modulus of its eigenvalues and denoted by $\rho(A)$. It characterizes the growth rate of A^n for $n \to \infty$: according to Gelfand's formula $\rho(A) = \lim_{n\to\infty} \|A^n\|^{1/n}$. The spectral radius depends continuously on the matrix, and is monotone for non-negative matrices [14, Cor. 8.1.19]: $\rho(A) \leqslant \rho(B)$ when $0 \leqslant A \leqslant B$. If $A > 0$, i.e. all the elements of A are positive, then by the Perron-Frobenius theorem, the number $\rho(A)$ is a simple eigenvalue of the matrix A, and all the other eigenvalues of A are strictly less than $\rho(A)$ in modulus. The eigenvector $v = (v_1, v_2, \ldots, v_N)^\intercal$ corresponding to the eigenvalue $\rho(A)$ (normalized, for example, by the equation $\sum v_i = 1$) is uniquely determined and positive.

Following [5], given N sets of M-dimensional rows $\mathscr{A}_i$ we define the *IRU-set* (independent row uncertainty set) $\mathscr{A}$ of $(N \times M)$-matrices that consists of all matrices of the form $A = (a_{ij})_{\substack{1\leqslant i\leqslant N\\ 1\leqslant j\leqslant M}}$ wherein each of the rows $a_i = [a_{i1}, a_{i2}, \ldots, a_{iM}]$ belongs to the respective $\mathscr{A}_i$. We will need several simple properties of IRU-sets.

▶ **Lemma 1.** *For an IRU-set $\mathscr{A}$ formed by sets of rows $\mathscr{A}_1, \mathscr{A}_2, \ldots, \mathscr{A}_N$ the following holds:*

(i) *for any matrix B the set $\mathscr{A}B = \{AB \mid A \in \mathscr{A}\}$ is IRU as well;*
(ii) *the convex hull $\mathtt{conv}(\mathscr{A})$ is the IRU-set formed by the row sets $\mathtt{conv}(\mathscr{A}_1), \ldots, \mathtt{conv}(\mathscr{A}_N)$;*
(iii) *the set $\mathscr{A}$ is compact if and only if so are all the row sets $\mathscr{A}_1, \mathscr{A}_2, \ldots, \mathscr{A}_N$.*

2.2 Joint Spectral Radius and Subradius

The *joint spectral radius* [19, 9, 10] of a bounded set $\mathscr{A}$ of $(N \times N)$-matrices characterizes the maximal growth rate of products of n matrices from the set and admits the following equivalent definitions (where the identity between the upper and the lower formulas constitutes the famous Berger-Wang Theorem [4]):

$$\begin{aligned}\hat\rho(\mathscr{A}) &= \lim_{n\to\infty} \sup\left\{\|A_1\cdots A_n\|^{1/n} \;\middle|\; A_i \in \mathscr{A}\right\} = \inf_{n\geqslant 1} \sup\left\{\|A_1\cdots A_n\|^{1/n} \;\middle|\; A_i \in \mathscr{A}\right\}\\ &= \lim_{n\to\infty} \sup\left\{\rho(A_1\cdots A_n)^{1/n} \;\middle|\; A_i \in \mathscr{A}\right\} = \sup_{n\geqslant 1} \sup\left\{\rho(A_1\cdots A_n)^{1/n} \;\middle|\; A_i \in \mathscr{A}\right\}. \quad (1)\end{aligned}$$

For a compact (closed and bounded) set $\mathscr{A}$, the suprema in (1) may be replaced by maxima.

The *joint spectral subradius* [13], or *lower spectral radius*, corresponds to the minimal growth rate of products of matrices:

$$\begin{aligned}\check\rho(\mathscr{A}) &= \lim_{n\to\infty} \inf\left\{\|A_1\cdots A_n\|^{1/n} \;\middle|\; A_i \in \mathscr{A}\right\} = \inf_{n\geqslant 1} \inf\left\{\|A_1\cdots A_n\|^{1/n} \;\middle|\; A_i \in \mathscr{A}\right\}\\ &= \lim_{n\to\infty} \inf\left\{\rho(A_1\cdots A_n)^{1/n} \;\middle|\; A_i \in \mathscr{A}\right\} = \inf_{n\geqslant 1} \inf\left\{\rho(A_1\cdots A_n)^{1/n} \;\middle|\; A_i \in \mathscr{A}\right\}.\end{aligned}$$

The equivalence of the characterizations based on norms and on spectral radii is established in [13, Thm B1] for finite sets $\mathscr{A}$, and in [21, Lemma 1.12] and [8, Thm 1] for arbitrary sets $\mathscr{A}$. Calculating the joint and lower spectral radii is a challenging problem, and only in exceptional cases these characteristics may be found explicitly, see, e.g., [15, 16] and the bibliography therein. The case of compact IRU-sets of non-negative matrices is such an exception, for which $\hat\rho$ and $\check\rho$ admit a simple characterization: as stated in [17, Thm 2], for such a set $\mathscr{A}$ the following equalities hold:

$$\hat\rho(\mathscr{A}) = \max_{A\in\mathscr{A}} \rho(A), \quad \check\rho(\mathscr{A}) = \min_{A\in\mathscr{A}} \rho(A). \quad (2)$$

Compact IRU-sets of non-negative matrices and their convex hulls have another useful property: as is shown in [17, Cor. 1],

$$\max_{A\in\mathscr{A}}\rho(A)=\max_{A\in\mathtt{conv}(\mathscr{A})}\rho(A),\quad \min_{A\in\mathscr{A}}\rho(A)=\min_{A\in\mathtt{conv}(\mathscr{A})}\rho(A), \tag{3}$$

and hence $\hat{\rho}(\mathscr{A})=\hat{\rho}(\mathtt{conv}(\mathscr{A}))$, $\check{\rho}(\mathscr{A})=\check{\rho}(\mathtt{conv}(\mathscr{A}))$.

2.3 Entropy of an ω-Language

The notion of entropy of a language and methods for computing it in the case of regular languages were introduced in [7] for finite words and in [20] for infinite ones. We will use the latter definition. The entropy of an ω-language $L\subseteq\Sigma^\omega$ is defined as

$$H(L)=\limsup_{n\to\infty}\frac{\log|\mathtt{pref}_n(L)|}{n}$$

(all the logarithms here are in base 2), where $\mathtt{pref}_n(L)$ is the set of prefixes of length n of infinite words in L. Intuitively, $H(L)$ is the information content ("bandwidth"), measured in bits per symbol, in typical words of the language. In particular, $H(\Sigma^\omega)=\log|\Sigma|$.

For a regular $L\subseteq\Sigma^\omega$ accepted by a given Büchi automaton, its entropy can be effectively computed as follows: compute the (finite) automaton recognizing $\mathtt{pref}(L)$, determinize it, and compute the entropy as the logarithm of the spectral radius of the adjacency matrix of the automaton obtained.

3 The Two Games

3.1 Entropy Games

Consider the arena (D,T,Σ,Δ) where D and T are disjoint finite sets of vertices (of two players), Σ a finite alphabet of actions and $\Delta\subseteq T\times\Sigma\times D\cup D\times\Sigma\times T$ is a transition relation. Given such an arena, we define a game with two-and-a-half players: Despot, Tribune and People. The latter plays non-deterministically and counts for half a player. People chooses the initial state in D. When the game is in a state d of D, Despot plays an action $a\in\Sigma$ and the game changes to some $t\in T$ (chosen by People) such that $(d,a,t)\in\Delta$. Then, Tribune plays an action $b\in\Sigma$ and the game changes its state to $d'\in D$, again chosen by People and such that $(t,b,d')\in\Delta$. It is again Despot's turn. The players must not block the game: they always choose an action that has a corresponding transition $(d,a,\cdot)\in\Delta$, or $(t,b,\cdot)\in\Delta$, respectively. We assume that the arena is non-blocking: at every state there is at least one such transition. Figure 1 shows an example of such an arena, which we will use as a running example in this paper.

A *play* of the EG is a finite or infinite sequence $\pi\in(D\cdot\Sigma\cdot T\cdot\Sigma)^\infty$ compatible with the transition relation Δ. Note that four letters in a row correspond to one turn of the game. A *strategy* σ for Despot is a function $(D\cdot\Sigma\cdot T\cdot\Sigma)^*\cdot D\to\Sigma$ that, given any finite play ending in a D state, outputs an action taken by Despot. The strategy is positional if it only depends on the current state of the game, i.e. it can be expressed just as $\sigma(d)$. A *strategy* τ for Tribune is a function $(D\cdot\Sigma\cdot T\cdot\Sigma)^*\cdot D\cdot\Sigma\cdot T\to\Sigma$ which, given any finite play ending in a T state, outputs the action taken by Tribune. The strategy is positional if it only depends on the current state of the game. In a natural way we define plays compatible with a Despot's strategy σ, or with a Tribune's strategy τ. Then, given σ and τ, we have an ω-language $L(\sigma,\tau)$ containing all the plays compatible with σ and τ. In other words,

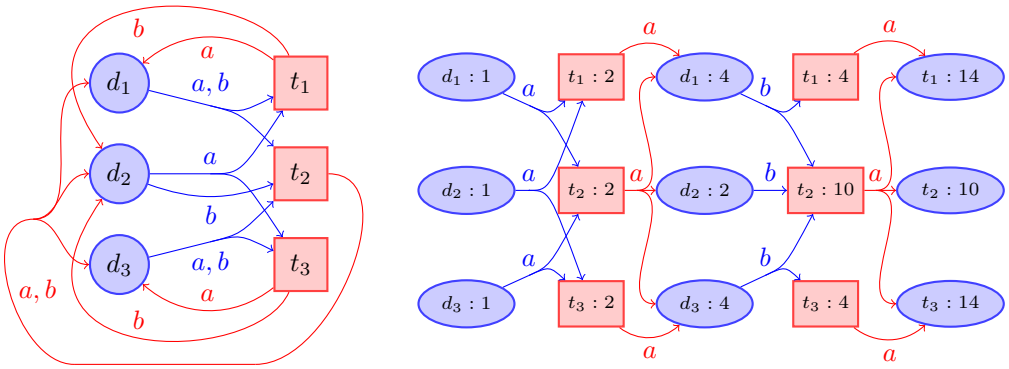


Figure 1 *Left.* Arena of our running example of an entropy game. Circles are states of the Despot while squares are states of the Tribune. At each move, the player has to choose between actions a and b, the outcome of which may sometimes be non-deterministic (e.g. when Despot plays a in state d_2, the next state may non-deterministically be either t_1 or t_3). *Right.* A finite play on this arena. Despot plays ab (whatever his opponent does) while Tribune plays aa. We only give, for each step, the number of words that end up in each state controlled by the active player.

$L(\sigma, \tau)$ is the set of runs that People can choose if Despot and Tribune commit themselves to σ and τ. What makes EGs different from other games (parity/mean-payoff etc.) is that the payoff does not depend on a single run of the game, but on the whole set of possible runs. More precisely, the payoff (the amount that Despot pays to Tribune) is defined as $P(\sigma, \tau) = \limsup_{n\to\infty} |\mathtt{pref}_{4n}(L(\sigma, \tau))|^{1/n}$, that is the growth rate (w.r.t. the number of turns) of the number of plays available to the People under the strategies σ and τ. Note that the payoff is a monotone function of the entropy of $L(\sigma, \tau)$, indeed $P(\sigma, \tau) = 2^{4H(L(\sigma,\tau))}$, i.e. Despot tries to diminish the entropy while Tribune aims to augment it.

3.2 Matrix Multiplication Games

Let $\mathscr{A}$ be a set of $M \times N$-matrices and $\mathscr{E}$ of $N \times M$-matrices. The MMG between two players, Adam and Eve, is played as follows: in turn, for every $i \in \mathbb{N}$, Adam writes a matrix $A_i \in \mathscr{A}$ and then Eve writes a matrix $E_i \in \mathscr{E}$. Formally, we define a *play* as an infinite sequence $A_1E_1A_2E_2 \dots A_iE_i \dots$ with $A_i \in \mathscr{A}$ and $E_i \in \mathscr{E}$. A strategy for Adam is a function $\sigma : (\mathscr{A} \cdot \mathscr{E})^* \to \mathscr{A}$ that maps any finite history (which is a sequence of matrices) to Adam's next move. Similarly, a strategy for Eve is a mapping $\tau : (\mathscr{A} \cdot \mathscr{E})^* \cdot \mathscr{A} \to \mathscr{E}$. A strategy is called *constant* if it does not depend on the history, i.e. is given by just one matrix: $\sigma = A \in \mathscr{A}$ or $\tau = E \in \mathscr{E}$. We define a play compatible with a strategy σ (or τ) in a natural way. Note that, given a strategy σ for Adam and a strategy τ for Eve, there exists a unique play $\pi(\sigma, \tau)$ compatible with both of them. The payoff of a play $\pi = A_1E_1A_2E_2 \dots A_iE_i \dots$ (that is, the amount that Adam pays to Eve) is the growth rate of the norm of the infinite product of matrices: $P(\pi) = P(\sigma, \tau) = \limsup_{k\to\infty} \left\| \prod_{i=1}^{k} A_iE_i \right\|^{1/k}$.

3.3 General Matrix Multiplication Games are Undecidable

The difficulty of general MMGs should be compared with results on the difficulty of JSR (joint spectral radius) computation. Thus, as proved in [6, Thm 2], given a finite set $\mathscr{E}$ of non-negative matrices with rational elements, it is undecidable whether $\hat{\rho}(\mathscr{E}) \leqslant 1$. The

decidability status of the problem $\hat{\rho}(\mathscr{E}) < 1$ is unknown. Finally, it is immediate from the characterization (1) that, given a precision $\varepsilon > 0$, it is possible to compute ε-approximation of $\hat{\rho}(\mathscr{E})$ (in other words $\hat{\rho}(\mathscr{E})$ is computable as function of $\mathscr{E}$ in the sense of computable analysis, see [24]).

▶ **Theorem 2.** *Given a determined MMG with finite sets of non-negative matrices with rational elements and $\alpha \in \mathbb{Q}_+$, the decision problem for its value $V \leqslant \alpha$ is undecidable.*

Proof. Let $\mathscr{A} = \{Id\}$ (Adam is trivial) and $\mathscr{E}$ be a finite set of non-negative matrices with rational elements. The corresponding MMG is determined with value $V = \hat{\rho}(\mathscr{E})$ and thus the decision problem $V \leqslant 1$ is undecidable due to [6, Thm 2], cited above. ◀

To prove stronger undecidability results for MMGs without direct counterparts for the JSR, we need a couple of simulation lemmas: for arbitrary matrices and for non-negative ones.

▶ **Lemma 3.** *Given a two-counter machine M, one can construct two finite sets of* integer *matrices $\mathscr{A}$ and $\mathscr{E}$ such that the corresponding MMG is determined and its value V satisfies:* "if M halts (starting with counters containing 0) then $V = 0$, else $V = 1$."

▶ **Lemma 4.** *Given a two-counter machine M, one can construct two finite sets of non-negative integer matrices $\mathscr{A}$ and $\mathscr{E}$ such that the corresponding MMG satisfies:*
"if M halts then Adam can ensure payoff < 2, otherwise Eve can ensure payoff $\geqslant 2$."

In both cases the construction, inspired by [11], follows the same principle: Eve tries to simulate the machine M; if she cheats, then Adam detects this and "resets" the product. Since the halting problem is undecidable, we obtain immediately the following two theorems.

▶ **Theorem 5.** *Given a determined MMG with finite sets of matrices with integer elements*
- *its value V is not computable from the matrices;*
- *it is not computable even knowing a priori that $V \in \{0, 1\}$.*

Hence the MMG value cannot be approximated and is not computable (as function of $\mathscr{A}$ and $\mathscr{E}$) in the sense of computable analysis.

▶ **Theorem 6.** *Given an MMG with finite sets of non-negative matrices with integer elements, it is undecidable whether the maximal payoff that Eve can ensure is < 2.*

3.4 Relations Between the Two Kinds of Games

Fortunately, as will be shown below, the subclass of MMGs with IRU-sets of non-negative matrices is much easier to solve. In this section, we relate EGs to such MMGs.

Let $\mathbf{A} = (D, T, \Sigma, \Delta)$ be an arena with $D = \{d_1, \ldots, d_M\}$ and $T = \{t_1, \ldots, t_N\}$. We define matrix sets $\mathscr{A}, \mathscr{E}$ as follows. For each Despot's vertex $d_i \in D$, and action $a \in \Sigma$ we define the row $c_{ia} = [c_{ia,1}, \ldots, c_{ia,N}]$ where $c_{ia,j} = 1$ if $(d_i, a, t_j) \in \Delta$ and $c_{ia,j} = 0$ otherwise. Next we define the row set $\mathscr{A}_i = \{c_{ia} \neq 0 \mid a \in \Sigma\}$ (non-zero rows correspond to non-blocking actions). Row sets $\mathscr{A}_1, \ldots, \mathscr{A}_M$ determine an IRU-set of matrices $\mathscr{A}$. The IRU-set $\mathscr{E}$ corresponding to Tribune's actions is defined similarly. In the running example in Figure 1, for instance, the row sets are the following: $\mathscr{A}_1 = \{[1,1,0]\}, \mathscr{A}_2 = \{[0,1,0], [1,0,1]\}, \mathscr{A}_3 = \{[0,1,1]\}, \mathscr{E}_1 = \{[0,1,0], [1,0,0]\}, \mathscr{E}_2 = \{[1,1,1]\}, \mathscr{E}_3 = \{[0,1,0], [0,0,1]\}$.

Note first that there is a natural bijection between the positional strategies of Despot and the set $\mathscr{A}$: any positional strategy $\sigma : D \to \Sigma$ corresponds to the matrix $A_\sigma \in \mathscr{A}$ with i-th row $c_{i,\sigma(d_i)}$ for Adam. Similarly, a positional strategy of Tribune τ corresponds to Eve's matrix $E_\tau \in \mathscr{E}$. The following lemma generalizes this observation to any type of strategies:

▶ **Lemma 7.** *Let* $\mathbf{A}$ *be an arena and* $\mathscr{A}, \mathscr{E}$ *the corresponding IRU matrix sets. Then for every pair of strategies* (σ, τ) *of Despot and Tribune in the EG on* $\mathbf{A}$ *there exists a pair of strategies* (ς, θ) *of Adam and Eve in the MMG* $(\mathtt{conv}(\mathscr{A}), \mathtt{conv}(\mathscr{E}))$ *with exactly the same payoff. Moreover, if* σ *is positional, then* ς *is constant and permanently chooses* A_σ*. The case of positional* τ *is similar.*

Note that Lemma 7 provides a rather weak relation between two games and does not mean, by itself, that the two games have the same value. However, we will show later (cf. Lemma 15) that optimal **constant** strategies in the MMG that belong to $\mathscr{A}$ and $\mathscr{E}$ are in bijection with optimal positional strategies in the EG.

4 Minimax Theorem for IRU-Sets of Matrices

In this section, we prove the key theorem of this article.

▶ **Theorem 8.** *Let* $\mathscr{A}$ *be a compact IRU-set of non-negative* $(N \times M)$*-matrices and* $\mathscr{B}$ *be a compact IRU-set of non-negative* $(M \times N)$*-matrices. Then*

$$\min_{A\in\mathscr{A}} \max_{B\in\mathscr{B}} \rho(AB) = \max_{B\in\mathscr{B}} \min_{A\in\mathscr{A}} \rho(AB). \tag{4}$$

In the rest of the article we will denote this minimax by $\mathtt{mm}(\mathscr{A}, \mathscr{B})$. The study of minimax relations will be based on the following well-known fact:

▶ **Lemma 9** (see [23, Sect. 13.4])**.** *Let* $f(x, y)$ *be a continuous function on the product of compact spaces* $X \times Y$*. Then* $\min_x \max_y f(x, y) \geqslant \max_y \min_x f(x, y)$*. The exact equality holds if and only if there exists a saddle point, i.e. a point* (x_0, y_0) *satisfying the inequalities* $f(x_0, y) \leqslant f(x_0, y_0) \leqslant f(x, y_0)$ *for all* $x \in X$*,* $y \in Y$*.*

We will also use two lemmas on matrices. The first one provides spectral radius bounds and is quite standard in Perron-Frobenius theory; as usual in this theory it relates global characteristics of a non-negative matrix (such as spectral radius) with its behavior on one non-negative vector.

▶ **Lemma 10.** *Let* A *be a non-negative* $(N \times N)$*-matrix; then the following properties hold:*

(i) *if* $Au \leqslant \rho u$ *for some vector* $u > 0$*, then* $\rho \geqslant 0$ *and* $\rho(A) \leqslant \rho$*;*
(ii) *if furthermore* $A > 0$ *and* $Au \neq \rho u$*, then* $\rho(A) < \rho$*;*
(iii) *if* $Au \geqslant \rho u$ *for some non-zero vector* $u \geqslant 0$ *and some number* $\rho \geqslant 0$*, then* $\rho(A) \geqslant \rho$*;*
(iv) *if furthermore* $Au \neq \rho u$*, then* $\rho(A) > \rho$*.*

The next lemma concerning IRU-sets of matrices is new and can be explained as follows. For an IRU-set of matrices and two vectors u and v we imagine that the sets $B_l = \{x : x \leqslant v\}$ and $B_u = \{x : v \leqslant x\}$ form the lower and upper bulbs of an hourglass with the neck at the point v. The lemma asserts that either all the grains Au (for all matrices A in the set) fill one of the bulbs, or there remains at least one grain in the other bulb. Clearly this alternative does not hold for general sets of matrices.

▶ **Lemma 11** (hourglass alternative)**.** *Let* $\mathscr{A}$ *be an IRU-set of* $(N \times M)$*-matrices and let* $\tilde{A}u = v$ *for some matrix* $\tilde{A} \in \mathscr{A}$ *and vectors* u, v*. Then the following holds:*

(i) *either* $Au \geqslant v$ *for all* $A \in \mathscr{A}$ *or exists a matrix* $\bar{A} \in \mathscr{A}$ *such that* $\bar{A}u \leqslant v$ *and* $\bar{A}u \neq v$*;*
(ii) *either* $Au \leqslant v$ *for all* $A \in \mathscr{A}$ *or exists a matrix* $\bar{A} \in \mathscr{A}$ *such that* $\bar{A}u \geqslant v$ *and* $\bar{A}u \neq v$*.*

We are ready to prove the minimax theorem.

Proof of Thm 8. According to Lemma 9, the minimax equality (4) may occur if and only if some matrices $\tilde{A} \in \mathscr{A}$ and $\tilde{B} \in \mathscr{B}$ satisfy the inequalities

$$\rho(\tilde{A}B) \leqslant \rho(\tilde{A}\tilde{B}) \quad \text{for all } B \in \mathscr{B}; \tag{5}$$

$$\rho(\tilde{A}\tilde{B}) \leqslant \rho(A\tilde{B}) \quad \text{for all } A \in \mathscr{A}. \tag{6}$$

Consider first the case when all the matrices in $\mathscr{A}$ and $\mathscr{B}$ are **positive**. To construct the matrices $\tilde{A} \in \mathscr{A}$ and $\tilde{B} \in \mathscr{B}$ we proceed as follows. For each $B \in \mathscr{B}$ let $A_B \in \mathscr{A}$ be a matrix that minimizes (in A) the quantity $\rho(AB)$. Such a matrix A_B exists due to compactness of the set $\mathscr{A}$ and continuity of the function $\rho(AB)$ in A and B. Then, for each matrix $B \in \mathscr{B}$, the relations $\rho(A_B B) = \min_{A\in\mathscr{A}} \rho(AB) \leqslant \rho(AB)$ hold for all $A \in \mathscr{A}$. Let $\tilde{B}$ be the matrix maximizing $\min_{A\in\mathscr{A}} \rho(AB)$ over the set $\mathscr{B}$, and let $\tilde{A} = A_{\tilde{B}}$. In this case

$$\max_{B\in\mathscr{B}} \rho(A_B B) = \max_{B\in\mathscr{B}} \min_{A\in\mathscr{A}} \rho(AB) = \min_{A\in\mathscr{A}} \rho(A\tilde{B}) = \rho(A_{\tilde{B}}\tilde{B}) = \rho(\tilde{A}\tilde{B}), \tag{7}$$

which implies inequality (6) for all $A \in \mathscr{A}$, and it remains to prove (5) for all $B \in \mathscr{B}$.

Let $v = (v_1, v_2, \ldots, v_N)^{\mathsf{T}}$ be the positive eigenvector of the $(N \times N)$-matrix $\tilde{A}\tilde{B}$ corresponding to the eigenvalue $\tilde{\rho} = \rho(\tilde{A}\tilde{B})$. By denoting $w = \tilde{B}v \in \mathbb{R}^M$ we obtain that $\tilde{\rho}v = \tilde{A}w$. Let us show that in this case

$$\tilde{\rho}v \leqslant Aw \quad \text{for all } A \in \mathscr{A}. \tag{8}$$

Otherwise, by Lemma 11(i) there would exist a matrix $\bar{A} \in \mathscr{A}$ such that $\tilde{\rho}v \geqslant \bar{A}w$ and $\tilde{\rho}v \neq \bar{A}w$, which implies, by the definition of the vector w, that $\tilde{\rho}v \geqslant \bar{A}\tilde{B}v$ and $\tilde{\rho}v \neq \bar{A}\tilde{B}v$. Then by Lemma 10 $\rho(\bar{A}\tilde{B}) < \tilde{\rho} = \rho(\tilde{A}\tilde{B})$, which contradicts (6). This contradiction completes the proof of inequality (8). Similarly, now we show that

$$w \geqslant Bv \quad \text{for all } B \in \mathscr{B}. \tag{9}$$

Again, assuming the contrary, by Lemma 11(ii) there exists a matrix $\bar{B} \in \mathscr{B}$ such that $w \leqslant \bar{B}v$ and $w \neq \bar{B}v$. This last inequality, together with (8) applied to the matrix $A_{\bar{B}}$, yields $\tilde{\rho}v \leqslant A_{\bar{B}}\bar{B}v$ and $\tilde{\rho}v \neq A_{\bar{B}}\bar{B}v$. Then by Lemma 10 $\tilde{\rho} < \rho(A_{\bar{B}}\bar{B})$, which contradicts (7) asserting that $\tilde{\rho} = \rho(\tilde{A}\tilde{B})$ is the maximum value of the function $\rho(A_B B)$ over all $B \in \mathscr{B}$. This contradiction completes the proof of inequality (9).

From $\tilde{\rho}v = \tilde{A}w$ and (9) we obtain the inequality $\tilde{\rho}v \geqslant \tilde{A}Bv$ valid for all $B \in \mathscr{B}$, which by Lemma 10 implies the relations $\rho(\tilde{A}\tilde{B}) = \tilde{\rho} \geqslant \rho(\tilde{A}B)$ valid for all $B \in \mathscr{B}$, or, which is the same, inequality (5). The theorem is proved for positive matrices.

Consider now the general case of compact IRU-sets of **non-negative matrices** $\mathscr{A}$ and $\mathscr{B}$. If the set $\mathscr{A}$ is determined by some sets of M-rows $\mathscr{A}_i$, $i = 1, 2, \ldots, N$, then choose an arbitrary $\varepsilon > 0$ and consider the sets of rows $\mathscr{A}_i^{(\varepsilon)} = \{a^{(\varepsilon)} \mid a^{(\varepsilon)} = a + \varepsilon[1, 1, \ldots, 1],\ a \in \mathscr{A}_i\}$, where $i = 1, 2, \ldots, N$. In this case the IRU-set of matrices $\mathscr{A}^{(\varepsilon)}$ consists of positive matrices $A + \varepsilon\mathbf{1}$, where $A \in \mathscr{A}$ and $\mathbf{1}$ is the matrix with all elements equal to 1. Define similarly the IRU-set of matrices $\mathscr{B}^{(\varepsilon)}$.

By the result just proved, for each $\varepsilon > 0$ the minimax equality holds for positive matrices: $\min_{A\in\mathscr{A}^{(\varepsilon)}} \max_{B\in\mathscr{B}^{(\varepsilon)}} \rho(AB) = \max_{B\in\mathscr{B}^{(\varepsilon)}} \min_{A\in\mathscr{A}^{(\varepsilon)}} \rho(AB)$, which by Lemma 9 is equivalent to the existence of $\tilde{A}_\varepsilon \in \mathscr{A}$ and $\tilde{B}_\varepsilon \in \mathscr{B}$ such that

$$\rho((\tilde{A}_\varepsilon + \varepsilon\mathbf{1})(B + \varepsilon\mathbf{1})) \leqslant \rho((\tilde{A}_\varepsilon + \varepsilon\mathbf{1})(\tilde{B}_\varepsilon + \varepsilon\mathbf{1})) \leqslant \rho((A + \varepsilon\mathbf{1})(\tilde{B}_\varepsilon + \varepsilon\mathbf{1}))$$

for all $A \in \mathscr{A}$ and $B \in \mathscr{B}$. Taking here $\varepsilon = \varepsilon_n$, where $\{\varepsilon_n\}$ is an arbitrary sequence of positive numbers converging to zero, we get

$$\rho((\tilde{A}_{\varepsilon_n} + \varepsilon_n \mathbf{1})(B + \varepsilon_n \mathbf{1})) \leqslant \rho((\tilde{A}_{\varepsilon_n} + \varepsilon_n \mathbf{1})(\tilde{B}_{\varepsilon_n} + \varepsilon_n \mathbf{1})) \leqslant \rho((A + \varepsilon_n \mathbf{1})(\tilde{B}_{\varepsilon_n} + \varepsilon_n \mathbf{1})) \quad (10)$$

for all $A \in \mathscr{A}$ and $B \in \mathscr{B}$. Without loss of generality, in view of the compactness of the sets $\mathscr{A}$ and $\mathscr{B}$, we may assume the existence of matrices $\tilde{A}$ and $\tilde{B}$ such that $\tilde{A}_{\varepsilon_n} \to \tilde{A} \in \mathscr{A}$ and $\tilde{B}_{\varepsilon_n} \to \tilde{B} \in \mathscr{B}$ as $n \to \infty$. Then turning to the limit in (10), we obtain the inequalities $\rho(\tilde{A}B) \leqslant \rho(\tilde{A}\tilde{B}) \leqslant \rho(A\tilde{B})$ for all $A \in \mathscr{A}$ and $B \in \mathscr{B}$, which are equivalent to (5) and (6). This concludes the proof. ◀

▶ **Corollary 12.** *For IRU-sets $\mathscr{A}$ and $\mathscr{B}$ of non-negative matrices it holds that*

$$\mathtt{mm}(\mathtt{conv}(\mathscr{A}), \mathtt{conv}(\mathscr{B})) = \mathtt{mm}(\mathscr{A}, \mathscr{B}).$$

5 Solving the Games

5.1 Solving Matrix Multiplication Games for IRU-Sets

▶ **Theorem 13.** *Let $\mathscr{A}$ and $\mathscr{E}$ be compact IRU-sets of non-negative matrices. Then the corresponding MMG is determined, and moreover Adam and Eve possess constant optimal strategies.*

Proof. Let us apply Thm 8 to matrix sets $\mathscr{A}$ and $\mathscr{E}$. Define V, E_0 and A_0 such that

$$\min_{E\in\mathscr{E}} \rho(EA_0) = \max_{A\in\mathscr{A}} \min_{E\in\mathscr{E}} \rho(EA) = \min_{E\in\mathscr{E}} \max_{A\in\mathscr{A}} \rho(EA) = \max_{A\in\mathscr{A}} \rho(E_0A) = V. \quad (11)$$

Let Adam only play A_0. Take any compatible play $\pi = A_0E_1A_0E_2\cdots$ and put $C_i = A_0E_i$. Denote $\mathscr{C} = \{EA_0 | E \in \mathscr{E}\}$; it is an IRU-set by Lemma 1. The payoff P for π yields

$$\begin{aligned} P = \limsup_{n\to\infty} \|A_0C_1\cdots C_{n-1}E_n\|^{1/n} &\leqslant \limsup_{n\to\infty} (\|A_0\| \cdot \|C_1\cdots C_{n-1}\| \cdot \|E_n\|)^{1/n} \\ &\leqslant \lim_{n\to\infty} K^{\frac{2}{n}} \limsup_{n\to\infty} \|C_1\cdots C_{n-1}\|^{\frac{1}{n-1}} \leqslant \hat{\rho}(\mathscr{C}) \overset{1}{=} \max_{C\in\mathscr{C}} \rho(C) = \max_{E\in\mathscr{E}} \rho(EA_0) \overset{2}{=} V, \end{aligned}$$

where the constant K is an upper bound for the norms of the matrices in $\mathscr{A}$ and $\mathscr{E}$, equality 1 comes from the first equality (2) and equality 2 comes from (11).

Let Eve only play E_0. Take any compatible play $\pi' = A_1E_0A_2E_0\cdots$. Let us write $D_i = A_iE_0$. Denote $\mathscr{D} = \{AE_0, A \in \mathscr{A}\}$; it is an IRU-set. The payoff P' for π' is such that

$$\begin{aligned} P' = \limsup_{n\to\infty} \|C_1\cdots C_n\|^{1/n} \geqslant \liminf_{n\to\infty} \|C_1\cdots C_n\|^{1/n} \\ \geqslant \check{\rho}(\mathscr{D}) \overset{1}{=} \min_{D\in\mathscr{D}} \rho(D) = \min_{A\in\mathscr{A}} \rho(AE_0) \overset{2}{=} V, \end{aligned}$$

where equality 1 comes from the second equality (2) and equality 2 from (11) using the equality $\rho(EA_0) = \rho(A_0E)$.

We have proved that Adam (by constantly playing A_0) can ensure payoff $\leqslant V$ whatever Eve plays; and that Eve (by constantly playing E_0) can ensure payoff $\geqslant V$ whatever Adam plays. This concludes the proof. ◀

▶ **Corollary 14.** *Let $\mathscr{A}$ and $\mathscr{E}$ be compact IRU-sets of non-negative matrices. In the MMG on $\mathtt{conv}(\mathscr{A}), \mathtt{conv}(\mathscr{E})$, the constant optimal strategies can be chosen from sets $\mathscr{A}$ and $\mathscr{E}$.*

This follows immediately from the proof of the theorem and Cor. 12.

5.2 Solving Entropy Games

In this section, we consider an EG on an arena $\mathbf{A}$ and the corresponding matrix sets $\mathscr{A}$ and $\mathscr{E}$, as defined in Sect. 3.4.

▶ **Lemma 15.** *Let (σ, τ) be two positional strategies in the EG. Then, if corresponding constant strategies A_σ and E_τ are optimal for their respective players in the MMG with matrix sets $\mathtt{conv}(\mathscr{A})$ and $\mathtt{conv}(\mathscr{E})$, then so are σ and τ.*

▶ **Theorem 16.** *Every EG is determined, and Despot and Tribune possess positional optimal strategies.*

Proof. From Thm 13, we know that for the MMG $(\mathtt{conv}(\mathscr{A}), \mathtt{conv}(\mathscr{E}))$ both Adam and Eve possess optimal strategies, which consist in constantly playing some matrices A and E. From Cor. 14, the matrices A and E can be chosen from sets $\mathscr{A}$ and $\mathscr{E}$, respectively. Then, there exist positional strategies σ and τ on $\mathbf{A}$ such that $A = A_\sigma$ and $E = E_\tau$. By Lemma 15, strategies σ and τ are optimal in the EG. ◀

Back to the running example. Here a quick exploration of the combinations of rows shows that the matrices realizing the minimax over the two IRU-sets defined by row sets $\mathscr{A}_1, \mathscr{A}_2, \mathscr{A}_3$ and $\mathscr{E}_1, \mathscr{E}_2, \mathscr{E}_3$ are $A = \left[\begin{smallmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{smallmatrix}\right]$ for Adam/Despot and $E = \left[\begin{smallmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{smallmatrix}\right]$ for Eve/Tribune. These matrices describe both the optimal constant strategy of the MMG and the optimal positional strategy of the EG induced by this arena. The value of both games is the spectral radius $\rho(AE) = \rho\left(\left[\begin{smallmatrix} 2 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 2 \end{smallmatrix}\right]\right) = \left(\sqrt{17} + 3\right)/2 \simeq 3.562$.

5.3 Complexity Issues

We will analyze the complexity of solving matrix multiplication (and hence entropy) game. We start with necessary and sufficient conditions for inequalities on joint spectral radii and subradii of IRU-sets (recall also (2) relating them to maximal and minimal spectral radii).

▶ **Lemma 17.** *For any compact IRU-set of positive matrices $\mathscr{A}$ and $\alpha \in \mathbb{Q}_+$ the following equivalences hold:*

$$\hat{\rho}(\mathscr{A}) < \alpha \Leftrightarrow \exists v > 0 \, \forall A \in \mathscr{A}(Av < \alpha v); \tag{12}$$

$$\hat{\rho}(\mathscr{A}) \leqslant \alpha \Leftrightarrow \exists v > 0 \, \forall A \in \mathscr{A}(Av \leqslant \alpha v); \tag{13}$$

$$\check{\rho}(\mathscr{A}) > \alpha \Leftrightarrow \exists v > 0 \, \forall A \in \mathscr{A}(Av > \alpha v); \tag{14}$$

$$\check{\rho}(\mathscr{A}) \geqslant \alpha \Leftrightarrow \exists v > 0 \, \forall A \in \mathscr{A}(Av \geqslant \alpha v). \tag{15}$$

If the matrices are only non-negative, the equivalences (12) *above and* (16) *below hold:*

$$\check{\rho}(\mathscr{A}) \geqslant \alpha \Leftrightarrow \exists (v \geqslant 0, v \neq 0) \, \forall A \in \mathscr{A}(Av \geqslant \alpha v). \tag{16}$$

The computational aspects of calculating the values $\hat{\rho}(\mathscr{A})$ and $\check{\rho}(\mathscr{A})$ for IRU-sets of non-negative matrices, based on relations (2), are discussed in [5, 17, 18]. These articles provide polynomial algorithms for approximation of the minimal and maximal spectral radii, as well as a variant of the simplex method for these problems. In the next theorem we prove a complexity result in a form suitable for game analysis.

▶ **Theorem 18.** *Given a finite IRU-set of nonnegative matrices $\mathscr{A}$ with rational elements (represented by row sets $\mathscr{A}_1, \mathscr{A}_2, \ldots, \mathscr{A}_N$), and a number $\alpha \in \mathbb{Q}_+$, the decision problems whether $\hat{\rho}(\mathscr{A}) < \alpha$ and whether $\check{\rho}(\mathscr{A}) \geqslant \alpha$ belong to the complexity class* P*. Moreover, if the matrices are positive, then the decision problems $\hat{\rho}(\mathscr{A}) \leqslant \alpha$ and $\check{\rho}(\mathscr{A}) > \alpha$ are also in* P*.*

Proof. The polynomial algorithms are based on the previous lemma. Consider the problem of deciding $\hat{\rho}(\mathscr{A}) < \alpha$, which can be rewritten using (12) as $\exists v > 0 \forall A \in \mathscr{A}(Av < \alpha v)$. We will not test all the matrices $A \in \mathscr{A}$ (there are exponentially many of them); instead, we will treat each row separately. The condition $\forall A \in \mathscr{A}(Av < \alpha v)$ can be rewritten as a system of linear inequalities: for each i and for each row $[c_1, c_2, \ldots, c_N] \in \mathscr{A}_i$ require that $c_1 v_1 + c_2 v_2 + \cdots + c_N v_N < \alpha v_i$. The condition $v > 0$ can be written as N inequalities $v_i > 0$: one for each coordinate. Using a polynomial algorithm for linear programming we can decide whether a solution v satisfying all these linear inequalities exists.

All other decision procedures, based on (13)–(16), are similar. The condition $v \geqslant 0, v \neq 0$ can be represented as a disjunction of N linear systems $v_j > 0 \wedge \bigwedge_{i=1}^{N} v_i \geqslant 0$. ◀

▶ **Theorem 19.** *Given two finite IRU-sets of nonnegative matrices $\mathscr{A}$ and $\mathscr{B}$ with rational elements, and a number $\alpha \in \mathbb{Q}_+$, the decision problem of whether $\mathtt{mm}(\mathscr{A}, \mathscr{B}) < \alpha$ belongs to* NP∩coNP*. Moreover, if the matrices are positive, then the problem of whether $\mathtt{mm}(\mathscr{A}, \mathscr{B}) \leqslant \alpha$ is also in* NP ∩ coNP*.*

Proof. Consider the problem of deciding whether $\mathtt{mm}(\mathscr{A}, \mathscr{B}) < \alpha$, which can be rewritten as $\min_{A \in \mathscr{A}} \max_{B \in \mathscr{B}} \rho(BA) < \alpha$, or equivalently $\exists A_0 \in \mathscr{A}(\hat{\rho}(\mathscr{B}A_0) < \alpha)$. The nondeterministic polynomial algorithm proceeds as follows:

- guess non-deterministically a matrix $A_0 \in \mathscr{A}$;
- compute the representation of $\mathscr{B}A_0$ as an IRU-set generated by the row sets $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_N$;
- check the inequality $\hat{\rho}(\mathscr{B}A_0) < \alpha$ in polynomial time using Thm 18.

We conclude that the problem $\mathtt{mm}(\mathscr{A}, \mathscr{B}) < \alpha$ is in NP. The complementary problem $\mathtt{mm}(\mathscr{A}, \mathscr{B}) \geqslant \alpha$ is also in NP, as it can be rewritten as $\max_{B \in \mathscr{B}} \min_{A \in \mathscr{A}} \rho(AB) \geqslant \alpha$, or equivalently $\exists B_0 \in \mathscr{B}(\check{\rho}(\mathscr{A}B_0) \geqslant \alpha)$, and decided by a non-deterministic polynomial algorithm similarly. We conclude that the two problems belong to NP ∩ coNP.

For positive matrices, the proof for the other decision problem based on the second statement of Thm 18 is similar. ◀

Our main complexity result follows immediately.

▶ **Theorem 20.** *Given an EG or an MMG with finite IRU-sets of non-negative matrices with rational elements and $\alpha \in \mathbb{Q}_+$, the decision problem for its value: $V < \alpha$ is in* NP ∩ coNP*.*

6 Related Models

6.1 Weighted Entropy Games

Up to now we have considered entropy games with *simple* transitions, but it is straightforward to add multiplicities (weights) to them. A *weighted entropy game* is played on a *weighted arena* $\mathbf{A} = (D, T, \Sigma, \Delta, w)$ with a function $w : \Delta \to \mathbb{N}_+$ assigning weights to transitions (informally a weight is the number of ways in which a transition can be taken). Strategies and plays are defined as in the unweighted case. Let L be some set of (infinite) plays. For every $u \in \mathtt{pref}(L)$ we define its weight $w(u)$ as the product of weights of all the transitions taken along u. We define $w_n(L) = \sum_{u \in \mathtt{pref}_{4n}(L)} w(u)$, and finally the payoff corresponding to strategies σ and τ of two players is defined as: $P = \limsup_{n \to \infty} \left(w_n(L(\sigma, \tau))\right)^{1/n}$. Our main results on EGs (Thms 16 and 20) extend straightforwardly to weighted EGs.

6.2 Mean-Payoff Games

Well-known mean-payoff finite-state games (MPG) [12] can be considered as a deterministic subclass of weighted entropy games. A (variant of) MPG is played on arena (D, T, Δ, w) with transition relation $\Delta \subseteq D \times T \cup T \times D$ and weight function $w : \Delta \to \mathbb{N}$. The play starts in some state $d_0 \in D$, and the two players choose transitions in turn. The resulting play is an infinite word $\gamma_{d_0} \in (D \cdot T)^\omega$. The mean-payoff corresponding to the play $\gamma_{d_0} = d_0, t_0, d_1, t_1, \dots$ is the limit of the average weight of transitions taken: $\mathtt{mp}(\gamma_{d_0}) = \limsup_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} (w(d_{i-1}, t_{i-1}) + w(t_{i-1}, d_i))$. Finally, player D wants to minimize and player T to maximize the payoff $\max_{d_0 \in D} \mathtt{mp}(\gamma_{d_0})$. As proved in [12], MPGs are determined and their optimal strategies are positional. As for complexity, [25] shows that testing whether the value of an MPG is smaller than a rational α is in $\mathsf{NP} \cap \mathsf{coNP}$ and becomes polynomial for weights presented in the unary system.

An MPG $\mathbf{A} = (D, T, \Delta, w)$ can be transformed into a weighted EG $\mathbf{A}' = (D, T, \Sigma, \Delta', w')$ as follows. The states of both players are the same, Σ is large enough, and for each transition $(p, q) \in \Delta$ there is a corresponding transition $(p, a, q) \in \Delta'$ with some a (occurring only in this transition). Its weight is $w'(p, a, q) = 2^{w(p,q)}$. We notice that the EG obtained is deterministic: due to unique transition labels for any strategies σ and τ, the language $L(\sigma, \tau)$ contains one play for each initial state. Strategies and plays of both games $\mathbf{A}$ and $\mathbf{A}'$ are now in natural bijection and the payoff of $\mathbf{A}$ equals the logarithm of the payoff of $\mathbf{A}'$.

This way, we obtain the classical results that MPGs are determined and both players have optimal positional strategies. Due to the exponential encoding of payoffs, the complexity obtained using our approach is, however, not as good as using direct algorithms, see [25].

6.3 Population Dynamics

Consider an EG with arena $\mathbf{A} = (D, T, \Sigma, \Delta)$. It can be interpreted as the following population game between two players, Damien and Theo. Elements of D and T correspond to species (forms of viruses, microorganisms, etc.). Initially there is one (or any non-zero number of) organism(s) for each species in D. At his turn Damien chooses an action $a \in \Sigma$ and applies it to each organism. An organism of species d, when subject to action a, turns into the set of organisms of species $\{t \mid (d, a, t) \in \Delta\}$. Theo plays similarly. The aim of Damien is to minimize the growth rate of the population, while Theo wants to maximize it. The value of the game and the optimal (positional) strategies are the same as for the EG.

7 Conclusions

We have introduced two (closely interrelated) families of games: entropy games played on finite arenas (graphs), and matrix multiplication games. The main result is that entropy games are determined and optimal strategies are positional in EG, while MMGs for IRU-sets of non-negative matrices are determined and optimal strategies are constant. These results are based on a new minimax theorem on spectral radii of products of IRU-sets of matrices. The results obtained prove the existence of equilibria in zero-sum games with a new type of limit payoffs, which is neither computed on a single play of the game nor probabilistic. On the other hand, they rely upon and generalize important results on the computability of joint spectral radii and subradii, an important problem in switching dynamic systems.

A presumably straightforward extension would be the "probabilization" of our game models, in that both Despot and Tribune would be allowed to play randomized strategies. The minimax theorem ensures the existence of optimal pure strategies for both players.

However the entropy-based payoff of the game needs to be given a proper generalization to this probabilistic setting. We may mention that such a generalization could be seen as entropy games on stochastic branching processes, and provide interesting links with this research domain. Finally, both our games are turn-based games with perfect information. The first generalization to be considered is to go to concurrent games – where perhaps some polynomial-size memory is needed, similarly to the classic case of concurrent games played on graphs in infinite time. The more difficult case is that of games of imperfect information: corresponding matrix games no longer have a simple structure (independent row uncertainty), and we conjecture that analysis of such games is non-computable. Last but not least, potential applications sketched in the introduction should be addressed.

References

1 Eugene Asarin, Nicolas Basset, and Aldric Degorre. Entropy of regular timed languages. *Inform. Comput.*, 241:142–176, 2015. doi:10.1016/j.ic.2015.03.003.

2 Eugene Asarin, Michel Blockelet, Aldric Degorre, Cătălin Dima, and Chunyan Mu. Asymptotic behaviour in temporal logic. In *Proc. CSL-LICS*, pages 10:1–10:9. ACM, 2014. doi:10.1145/2603088.2603158.

3 Eugene Asarin, Julien Cervelle, Aldric Degorre, Cătălin Dima, Florian Horn, and Victor Kozyakin. Entropy Games and Matrix Multiplication Games. https://hal.archives-ouvertes.fr/hal-01164086, 2015.

4 Marc A. Berger and Yang Wang. Bounded semigroups of matrices. *Linear Algebra Appl.*, 166:21–27, 1992. doi:10.1016/0024-3795(92)90267-E.

5 Vincent D. Blondel and Yurii Nesterov. Polynomial-time computation of the joint spectral radius for some sets of nonnegative matrices. *SIAM J. Matrix Anal. A.*, 31(3):865–876, 2009. doi:10.1137/080723764.

6 Vincent D. Blondel and John N. Tsitsiklis. The boundedness of all products of a pair of matrices is undecidable. *Syst. Control Lett.*, 41(2):135–140, 2000. doi:10.1016/S0167-6911(00)00049-9.

7 Noam Chomsky and George A. Miller. Finite state languages. *Inform. Control*, 1(2):91–112, 1958. doi:10.1016/S0019-9958(58)90082-2.

8 Adam Czornik. On the generalized spectral subradius. *Linear Algebra Appl.*, 407:242–248, 2005. doi:10.1016/j.laa.2005.05.006.

9 Ingrid Daubechies and Jeffrey C. Lagarias. Sets of matrices all infinite products of which converge. *Linear Algebra Appl.*, 161:227–263, 1992. doi:10.1016/0024-3795(92)90012-Y.

10 Ingrid Daubechies and Jeffrey C. Lagarias. Corrigendum/addendum to [9]. *Linear Algebra Appl.*, 327(1-3):69–83, 2001. doi:10.1016/S0024-3795(00)00314-1.

11 Aldric Degorre, Laurent Doyen, Raffaella Gentilini, Jean-François Raskin, and Szymon Torunczyk. Energy and mean-payoff games with imperfect information. In *Proc. CSL*, LNCS 6247, pages 260–274. Springer, 2010. doi:10.1007/978-3-642-15205-4_22.

12 Andrzej Ehrenfeucht and Jan Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, 1979.

13 Leonid Gurvits. Stability of discrete linear inclusion. *Linear Algebra Appl.*, 231:47–85, 1995. doi:10.1016/0024-3795(95)90006-3.

14 Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 2013.

15 Raphaël Jungers. *The Joint Spectral Radius: Theory and Applications*. LNCIS 385. Springer, 2009. doi:10.1007/978-3-540-95980-9.

16 Victor Kozyakin. An annotated bibliography on convergence of matrix products and the theory of joint/generalized spectral radius. Preprint, Institute for Information Transmission

Problems, Moscow, 2013. http://dx.doi.org/10.13140/2.1.4257.5040. doi:10.13140/2.1.4257.5040.

17 Yurii Nesterov and Vladimir Yu. Protasov. Optimizing the spectral radius. *SIAM J. Matrix Anal. A.*, 34(3):999–1013, 2013. doi:10.1137/110850967.

18 Vladimir Yu. Protasov. Spectral simplex method. *Math. Program.*, pages 1–27, 2015. doi:10.1007/s10107-015-0905-2.

19 Gian-Carlo Rota and Gilbert Strang. A note on the joint spectral radius. *Nederl. Akad. Wetensch. Proc. Ser. A 63 = Indag. Math.*, 22:379–381, 1960.

20 Ludwig Staiger. Entropy of finite-state omega-languages. *Probl. Control Inform.*, 14(5):383–392, 1985.

21 Jacques Theys. *Joint Spectral Radius: Theory and Approximations.* PhD thesis, Université Catholique de Louvain, 2005.

22 John N. Tsitsiklis and Vincent D. Blondel. The Lyapunov exponent and joint spectral radius of pairs of matrices are hard – when not impossible – to compute and to approximate. *Math. Control Signals Systems*, 10(1):31–40, 1997. doi:10.1007/BF01219774.

23 John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, 1947.

24 Klaus Weihrauch. *Computable Analysis.* Springer, 2000.

25 Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158(1–2):343–359, 1996. doi:http://dx.doi.org/10.1016/0304-3975(95)00188-3.

Good Predictions Are Worth a Few Comparisons

Nicolas Auger[1], Cyril Nicaud[2], and Carine Pivoteau[3]

1 **Université Paris-Est, LIGM (UMR 8049), 77454 Marne-la-Vallée, France**
2 **Université Paris-Est, LIGM (UMR 8049), 77454 Marne-la-Vallée, France**
3 **Université Paris-Est, LIGM (UMR 8049), 77454 Marne-la-Vallée, France**

Abstract

Most modern processors are heavily parallelized and use predictors to guess the outcome of conditional branches, in order to avoid costly stalls in their pipelines. We propose predictor-friendly versions of two classical algorithms: exponentiation by squaring and binary search in a sorted array. These variants result in less mispredictions on average, at the cost of an increased number of operations. These theoretical results are supported by experimentations that show that our algorithms perform significantly better than the standard ones, for primitive data types.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases branch misses, binary search, exponentiation by squaring, Markov chains

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.12

1 Introduction

As an introductory example, consider the simple problem of computing both the minimum and the maximum of an array of size n. The naive approach is to compare each entry to the current minimum and maximum, which uses $2n$ comparisons. A better solution, in terms of number of comparisons, is to look at the elements of the array two by two, and to compare the smallest to the current minimum and the greatest to the current maximum. This uses only $3n/2$ comparisons, which is optimal.[1]

In order to observe the benefit of this optimization, we implemented both versions (see Figure 3) and measured their execution time[2] for large arrays of uniform random `float` in $[0, 1]$. The results are given in Figure 1 and are very far from what was expected, since the naive implementation is almost twice as fast as the optimized one. Clearly, counting comparisons cannot explain these counterintuitive performances. An obvious explanation could be a difference in the number of cache misses. However, both implementations make the same memory accesses, in the same order. Instead, we turn our attention to the comparisons themselves. Most modern processors are heavily parallelized and use predictors to guess the outcome of conditional branches in order to avoid costly stalls in their pipelines. Every time a conditional is used in a program, there is a mechanism that tries to predict whether the corresponding conditional jump will be taken or not. The cost of a misprediction can be quite large compared to a basic instruction, and should be taken into account in order to explain accurately the behavior of algorithms that use a fair amount of comparisons.

In this matter, our example is quite revealing since the trick used to lower the number of comparisons relies on a conditional branch that is unpredictable (for an input taken uniformly

[1] More precisely, an adversary argument can be used to establish a lower bound of $\lfloor \frac{3n}{2} \rfloor - 2$ comparisons, in the "decision tree with comparisons" model of computation [11].
[2] We used a Linux machine with a 3.40 GHz Intel Core i7-2600 CPU.


© Nicolas Auger, Cyril Nicaud, and Carine Pivoteau;
licensed under Creative Commons License CC-BY
33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 12; pp. 12:1–12:14

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

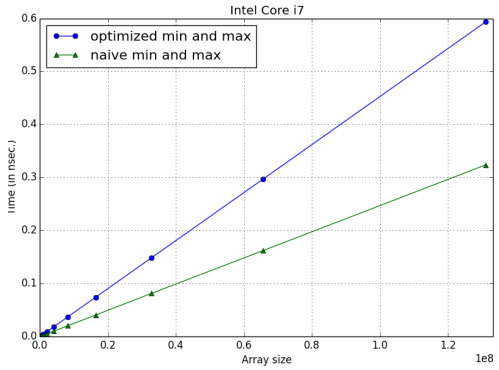



Figure 1 Execution time of simultaneous minimum and maximum searching.

at random) and will cause a substantial increase in the number of mispredictions. As we will see in the sequel, the expected number of mispredictions caused by the naive algorithm is $\Theta(\log n)$, whereas it is $\Theta(n)$ for the "optimal" one.

The influence of branch predictors over comparison based algorithms has already been studied, mostly to acknowledge the over-cost induced by mispredictions. Our approach is quite the opposite as we propose to take advantage of this feature, by proposing predictor-friendly versions of two classical algorithms.

Our contributions. After dealing with our introductory example using combinatorial arguments, we turn our attention towards the classical exponentiation by squaring and give a simple alternative algorithm, which reduces the number of mispredictions without increasing the number of multiplications. The analysis is based on the study of the Markov chains that describe the dynamic local predictors (see next section for a brief description of predictors). Finally, in the same vein, we propose biased versions of the binary search in a sorted array. We analyze the expected number of mispredictions for local predictors and we also give the (first to our knowledge) analysis of a global predictor. For these two different problems, we manage to significantly lower the number of mispredictions by breaking the perfect balance usually favored in the divide and conquer strategy. In practice, the trade-off between comparisons and mispredictions allows a noticeable speed-up in the execution time, when the comparisons involve primitive data types, which supports our theoretical results.

Related work. Over the past decade, several articles began to address the influence of branch predictors, and especially the cost of mispredictions, in comparison based algorithms. For instance, Biggar and his coauthors [1] investigated the behavior of branches for many sorting algorithms, in an extensive experimental study. Brodal, Fagerberg and Moruz reviewed the trade-offs between comparisons and mispredictions for several sorting algorithms [3] and studied how the number of inversions in the data affects statistics such as the number of mispredictions [2]. Moreover, these works introduced the first theoretical analysis of static branch predictors.

Also interested by the influence of mispredictions on the running time of sorting algorithms, Sanders and Winkel considered the possibility to dissociate comparisons from branches in their SampleSort, which allows to avoid most of the misprediction cost [13]. Elmasry, Katajainen and Stenmark then proposed a version of MergeSort that is not affected

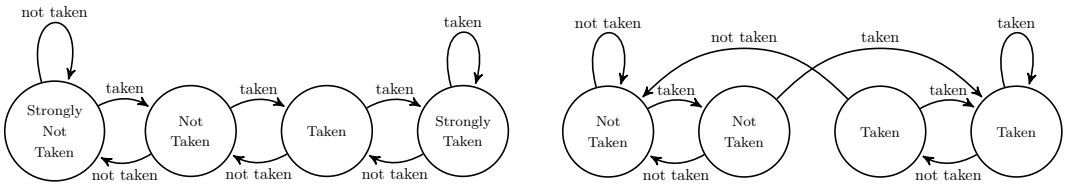


Figure 2 Two different 2-bit predictors (left: saturating counter, right: flip-on-consecutive).

by mispredictions [6], by taking advantage of some processor-specific instructions.[3] The influence of mispredictions was also studied for QUICKSORT: Kaligosi and Sanders gave an in-depth analysis of simple dynamic branch predictors to explain how mispredictions affect this classical algorithm [8]; however, Martínez, Nebel and Wild pointed out that this is not enough to explain the "better than expected" performances of the dual-pivot version of QUICKSORT [10] implemented in Java's standard library.

Besides, Brodal and Moruz conducted an experimental study of skewed binary search trees in [4], highlighting that such data structures can outperform well-balanced trees, since branching to the right or left does not necessarily have the same cost, due to branch prediction schemes. Our work follows the same line, as we also want to take advantage of the branch predictions, but we focus on algorithms rather than on data structures.

2 Elements of Computer Architecture

To analyze the complexity of searching or sorting algorithms, the standard model consists in counting the number of comparison operations performed. However most modern processors are pipelined. And to avoid stalling the pipeline when coming across a conditional jump, the processor tries to predict if the jump will occur and proceeds according to its prediction. A correctly predicted jump does not stall the pipeline whereas mispredictions lead this one to be flushed, causing a significant performance loss.[4] Therefore, the cost of a comparison in an "if" statement actually depends on the quality of the prediction.

For any conditional jump, a branch predictor will "guess" if the corresponding branch will be taken or not. For this purpose, many different strategies have been designed. The simplest one is a *static* branch predictor that does not use information from the code execution. It can, for example, predict that all branches will be taken. To improve its accuracy, a *dynamic* branch predictor uses the outcome of past branches to guess whether a particular branch should be taken or not. We now describe some techniques of dynamic branch prediction [7].

A *1-bit predictor* is a state buffer which remembers the last outcome of the branch; the guess is that the next outcome will be the same. As an improvement, the *2-bit predictors* try to avoid making two mispredictions when a branch takes an unlikely path. Two slightly different schemes are given by Figure 2. The saturating counter scheme can be further improved by keeping more information (*k-bit predictors* using 2^k states). All these predictors are *local*: there is one for every conditional (up to some limit in practice).

An *history table* has 2^n entries indexed by the sequence of the last n branches (1 for taken, 0 otherwise). The entries themselves are usually k-bit predictors. Such a table is said to be *local* when its entries correspond to the behavior of one specific branch and are used for

[3] Namely, conditional moves, which are now widely available on computers.
[4] For instance, on an Intel Core i7, a misprediction causes a penalty of about 15 cycles [7].

NaiveMinmax

```
1  min = max = T[0]
2  for(i=1; i<n; i++){
3    if (T[i] < min)
4      min = T[i]
5    if (T[i] > max)
6      max = T[i]
7  }
```

T is an array of size n. Both min and max are returned.

$\frac{3}{2}$-Minmax

```
1  min = max = T[n-1]
2  for(i=0; i<n-1; i+=2){
3    if (T[i] < T[i+1]){
4      if (T[i] < min)     min = T[i]
5      if (T[i+1] > max)   max = T[i+1]
6    }
7    else{
8      if (T[i+1] < min)   min = T[i+1]
9      if (T[i] > max)     max = T[i]
10   }
11 }
```

Figure 3 Naive and optimized implementations of simultaneous maximum and minimum finding.

this one only. On the contrary, in a *global* history table, the outcomes of the most recently executed branches are used to index the table, which is shared by all the conditionals.

To get the best of both worlds, *correlating branch predictors* use local and global information mixed together, and *tournament predictors* use an additional dynamic scheme to decide if they follow the local or the global prediction. These types of predictors are far beyond what we study in this article, but are worth mentioning for further analysis.

Strictly speaking, mispredictions can only be analyzed on a given assembly code, as they occur at conditional jumps. In the sequel, we use C-style pseudo code. We implicitly work on the non-optimized assembly code (compiled[5] with `gcc -O0`), where control structures are translated into conditional jumps in the standard way (i.e., not into conditional moves). For our experimental results, we checked that it was indeed the case. Furthermore, we remarked that our good results still hold for fully optimized binaries (compiled with `gcc -O3`).

3 Simultaneous Maximum and Minimum Finding

In this section, we go back to the example given in the introduction: We consider two algorithms that simultaneously compute the minimum and the maximum of an array of length n. These algorithms are given in Figure 3 (see also [5, Sec. 9.1]). For our analysis, we consider the local predictors presented in Section 2.

In the classical settings for the analysis, the algorithm $\frac{3}{2}$-Minmax is optimal (see the footnote[1] in the introduction), the number of comparisons performed being asymptotically equivalent to $\frac{3}{2}n$. Obviously, NaiveMinmax needs $2n-2$ comparisons.

In order to give an explanation of the experimental results presented in Figure 1, where NaiveMinmax outperforms $\frac{3}{2}$-Minmax, we estimate the expected number of mispredictions for both algorithms. Our probabilistic model is the following: we consider the *uniform random distribution on arrays of size n*, where each element is chosen uniformly and independently in $[0,1]$. Up to an event of probability 0 (when the elements of the input are not pairwise distinct), this is the same as choosing a uniform random permutation of $\{1,\dots,n\}$, since we only use comparisons on the elements in both algorithms.

Recall that a *min-record* (resp. *max-record*) in an array or a permutation is an element that is strictly smaller (resp. greater) than any element to its left. Obviously, in NaiveMinmax,

[5] We used version `4.5.3` of `gcc`.

the first conditional at line 3 (resp. the one at line 5) is true for each min-record (resp. max-record), except for the first position. The number of records in a random permutation is a well-known statistics, which we can use to establish the following proposition.

▶ **Proposition 1.** *The expected number of mispredictions performed by* NaiveMinmax*, for the uniform distribution on arrays of size n, is asymptotically equivalent to $4\log n$ for the 1-bit predictor and to $2\log n$ for the two 2-bit predictors and the 3-bit saturating counter. The expected number of mispredictions performed by $\frac{3}{2}$-*Minmax *is asymptotically equivalent to $\frac{n}{4} + \mathcal{O}(\log n)$ for all the considered predictors.*

In light of these results, we observe that the mispredictions occurring in NaiveMinmax are negligible towards the number of comparisons. On the other hand, the additional test used to optimize $\frac{3}{2}$-Minmax (line 3) causes the number of mispredictions to be comparable to the number of comparisons performed. We believe this is enough to explain why the naive implementation performs better (Figure 1), since we know that mispredictions can cost many CPU cycles and that comparisons are cheap operations. Of course, we are aware that other factors can influence the performances of such simple programs, including cache effects. In our implementation, we took care to fetch each element of the array only once and in the same order, so that the cache behavior should not interfere with our results. We also tried the most commonly used optimization of the `gcc`[5] compiler (`-O3`) to check that these results withstand strong code optimization.[6] In this particular case, all the branches but the one at line 3 in $\frac{3}{2}$-Minmax are replaced by conditional moves that are not vulnerable to misprediction. Hence, $\frac{3}{2}$-Minmax still causes approximatively $\frac{1}{4}n$ mispredictions on average.

Of course, these results do not hold when considering a non-uniform distribution over the entries. It would be interesting to study the behavior of both algorithms on random permutations with, for instance, a given number of records.

4 Exponentiation by Squaring

We saw in the previous section that conditional branches with equal probabilities of going one way or another are particularly harmful when using branch prediction. Besides, several divide and conquer algorithms feature such branches, since they tend to split problems into parts of equal size to reach an optimal complexity. In the sequel, we explore two different ways of disrupting this balance, to end up with better performances for two classical algorithms: exponentiation by squaring and binary search.

4.1 Modified algorithms

The classical divide and conquer algorithm to compute x^n consists in rewriting $x^n = (x^2)^{\lfloor n/2 \rfloor} x^{n_0}$, where $n_k \dots n_1 n_0$ is the binary decomposition of n, in order to divide the size n of the problem by two. This is the algorithm ClassicalPow of Figure 4. As expected, the conditional branch of line 3 is taken with probability $\frac{1}{2}$, which is what we want to avoid.[7] In order to introduce some imbalance in the algorithm, we first unroll the loop (UnrolledPow, Figure 4) using the decomposition $x^n = (x^4)^{\lfloor n/4 \rfloor}(x^2)^{n_1} x^{n_0}$. Still, both conditional branches are taken with probability $\frac{1}{2}$, but we can now guide the algorithm by injecting the test that determines whether the last two bits of n are 00 or not. This is the third algorithm

[6] Both algorithms are faster, as expected, but the naive version is still almost twice as fast.
[7] In our model, n is chosen uniformly at random between 0 and $4^k - 1$ for some positive k.

ClassicalPow (x,n)

```
1  r = 1;
2  while (n > 0) {
3    // n is odd
4    if (n & 1)
5      r = r * x;
6    n /= 2;
7    x = x * x;
8  }
```

x is a floating-point number, n is an integer and r is the returned value.

UnrolledPow (x,n)

```
1  r = 1;
2  while (n > 0) {
3    t = x * x;
4    // n0 == 1
5    if (n & 1)
6      r = r * x;
7    // n1 == 1
8    if (n & 2)
9      r = r * t;
10   n /= 4;
11   x = t * t;
12 }
```

GuidedPow (x,n)

```
1  r = 1;
2  while (n > 0) {
3    t = x * x;
4    // n1n0 != 00
5    if (n & 3) {
6      if (n & 1)
7        r = r * x;
8      if (n & 2)
9        r = r * t;
10   }
11   n /= 4;
12   x = t * t;
13 }
```

Figure 4 Three versions of the exponentiation by squaring, in C. The `&` denotes the bitwise `AND` in the C language.

Pow	time (in sec.)	loops $\times 10^9$	mult. $\times 10^9$	branches $\times 10^9$	mispred. $\times 10^9$
classical	7.230	1.250	1.900	1.300	0.674
unrolled	6.316	0.633	1.917	1.317	0.683
guided	5.606	0.633	1.917	1.658	0.554

Figure 5 Some parameters measured during 5.10^7 computations of x^n with the three algorithms of Figure 4, using the PAPI library.[8] The values of n are chosen uniformly at random between 0 and $2^{26}-1$. The number of branches is given excluding the ones caused by loops, as they do not yield mispredictions.

of Figure 4. Note that this conditional branch (line 4) is absolutely unnecessary in the algorithm, as it is redundant with the tests of line 5 and 7. But on the other hand, this branch is taken with probability $\frac{3}{4}$ and the branches of line 5 and 7 are now both taken with probability $\frac{2}{3}$. This is how we aim at using the branch predictions.

To compare their performances experimentally, we computed the floating-point value of x^n using each of the algorithms 5.10^7 times, with n chosen uniformly at random in $\{0,\ldots,2^{26}-1\}$. We measured the execution time, as well as some other parameters given by the latest version of the PAPI library,[8] which give access, for instance, to the number of mispredictions occurring during the execution. These results are depicted in Figure 5. The first observation is that GuidedPow is 14% faster than UnrolledPow and 29% faster than ClassicalPow and yet, the number of multiplications performed is essentially the same for the three algorithms. The main explanation we have come across for the speed-up between UnrolledPow and ClassicalPow is that the number of loops is divided by two. As for GuidedPow, the number of loops is the same as for UnrolledPow and it uses 25% more comparisons, but still the guided version is faster. The main difference between the two is that the test added at line 4 allows to decrease the number of mispredictions by about a quarter (this test causes additional mispredictions, but it also modifies the probabilities associated to the inner conditionals of line 6 and 8, which leads to an overall decrease in the number of mispredictions). We are in similar settings as for the simultaneous minimum and maximum, where the increased number of comparisons is balanced by less mispredictions.

[8] PAPI 5.4.1.0 , see `http://icl.cs.utk.edu/papi`.

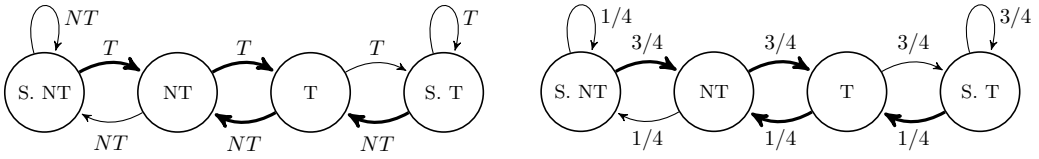


Figure 6 The saturating counter and its associated Markov chain for the first conditional of GuidedPow. The bold edges correspond to mispredictions.

We now proceed with the analysis of this phenomenon.

4.2 Analysis of the Average Number of Mispredictions for GuidedPow

For the analysis, we consider that n is taken uniformly at random in $\{0, \dots, N-1\}$, for $N = 4^k$ and with $k \geq 1$. This model is exactly the same as choosing each of the $2k$ bits of the binary representation of n uniformly at random and independently. We consider the local predictors presented in Section 2.

Let $L_k(n)$ be the number of loop iterations of GuidedPow. This is a random variable, which is easy to analyze since it is equal to the smallest integer ℓ such that 4^ℓ is greater than n. In particular, we have $\mathbb{E}[L_k] = k - \frac{1}{3} + o(1) \sim k$.

We now recall, using our algorithm as an example, why Markov chains are the key tools for that kind of analysis (as shown in [8, 10]). Let us consider the first conditional of line 4. In our model, at each iteration, the condition is true with probability $\frac{3}{4}$, as it is not satisfied when the last two bits are 00. It yields that the behavior of the predictor associated to this conditional is exactly described by the Markov chain obtained when changing the edges labels "taken" by $\frac{3}{4}$ and the labels "not taken" by $\frac{1}{4}$ (see Figure 6). A misprediction occurs whenever an edge labeled by "taken" (resp. "not taken") is used from a state that predicts "not taken" (resp. "taken"). We also need to know the initial state of the predictor, but it has no influence on our asymptotic results, as we shall see.

Hence, we reduced our problem to counting the number of times some particular edges are taken in a Markov chain, when we perform a random walk of (random) length L_k. We can therefore conclude using the classical Ergodic Theorem [9], which we restated bellow in order to fit our needs.

▶ **Theorem 2** (Ergodic Theorem). *Let (M, π_0) be a primitive and aperiodic Markov Chain on the finite set S. Let π be its stationary distribution. Let E be a set of edges of M, that is, a set of pairs $(i, j) \in S^2$ such that $M(i, j) > 0$.*

For any nonnegative integer n, let L_n be a random variable on nonnegative integers such that $\lim_{n\to\infty} \mathbb{E}[L_n] = +\infty$. Let X_n be the random variable that counts the number of edges in E that are used during a random walk of length L_n in M (starting from the initial distribution π_0). Then the following asymptotic equivalence holds: $\mathbb{E}[X_n] \sim \mathbb{E}[L_n] \sum_{(i,j)\in E} \pi(i)M(i, j)$.

When considering a given predictor, under the model where the condition is satisfied with probability p, we denote by M_p its transition matrix, by π_p its stationary vector and by $\mu(p)$ its *expected misprediction probability* defined by $\mu(p) = \sum_{(i,j)\in E} \pi_p(i)M_p(i, j)$, where E is the set of edges corresponding to mispredictions. As shown in [10], if we denote by $\mu_1(p)$, $\mu_2(p)$ and $\mu'_2(p)$ the expected misprediction probability of the 1-bit, 2-bit saturating counter and the flip-on-consecutive 2-bit, respectively, then we have:

$$\mu_1(p) = 2p(1-p); \quad \mu_2(p) = \frac{p(1-p)}{1-2p(1-p)}; \quad \mu'_2(p) = \frac{2p^2(1-p)^2 + p(1-p)}{1-p(1-p)}. \tag{1}$$

Similarly, the expected misprediction probability $\mu_3(p)$ of the 3-bit saturated counter is

$$\mu_3(p) = \frac{p(1-p)\,(1-3p(1-p))}{1-2p(1-p)\,(2-p(1-p))}. \tag{2}$$

Applying these mathematical tools to GuidedPow yields the following results. The theorem is stated for values of N that are not powers of 4, which is more complicated since the bits are not exactly 0's and 1's with probability $\frac{1}{2}$ (and not independent). In Section 5 we show how to deal with the cases where we slightly deviate from the ideal case.

▶ **Theorem 3.** *Assume that n is taken uniformly at random in $\{0,\dots,N-1\}$. The expected number of conditional tests in* ClassicalPow *and* UnrolledPow *is asymptotically equivalent to $\log_2 N$, whereas it is asymptotically equivalent to $\frac{5}{4}\log_2 N$ for* GuidedPow*. The expected number of mispredictions is asymptotically equivalent to $\frac{1}{2}\log_2 N$ for* ClassicalPow *and* UnrolledPow*, for any kind of predictor. For* GuidedPow*, it is asymptotically equivalent to $\alpha\log_2 N$, where $\alpha = \frac{1}{2}\mu(3/4) + \frac{3}{4}\mu(2/3)$, where μ is the expected misprediction probability associated with the local predictor.*

Using Theorem 3 and Equations (1) and (2), we get that α is equal to $\frac{25}{48} \approx 0.52$, $\frac{9}{20} = 0.45$, $\frac{2045}{4368} \approx 0.47$ and $\frac{1095}{2788} \approx 0.39$ for the 1-bit, 2-bit saturated, flip-on-consecutive 2-bit and 3-bit saturated counter, respectively. These values are to be compared with the $\frac{1}{2}$ of the other two algorithms. In particular, for the 1-bit predictor, the expected number of mispredictions is greater for GuidedPow than for ClassicalPow or UnrolledPow. This predictor is not efficient enough to offset the mispredictions caused by the additional conditional. For the 3-bit saturated counter, GuidedPow therefore uses $\approx 0.25\log_2 n$ more comparisons than UnrolledPow, but $\approx 0.11\log_2 n$ less mispredictions.

5 Binary Search and Variants

5.1 Unbalancing the Binary Search

We first consider the classical binary search which partitions a sorted array of size n into two parts of size $\frac{n}{2}$ and compares the value x that is searched for to the middle of the array in order to determine in which part of the array to continue the search. As before, if we consider arrays of uniform random floating-point numbers, we get a conditional branch that is taken with probability $\frac{1}{2}$. A simple way to change that is to partition another way, for instance with parts of size about $\frac{n}{4}$ and $\frac{3n}{4}$, as in the BiasedBinarySearch (see Figure 7). Carrying on with the divide and conquer strategy but partitioning the array into three parts of size about $\frac{n}{3}$, gives a ternary search. The main issue with this approach is that, in practice, the division by 3 is costly in terms of hardware. Thus, to limit the cost of partitioning, we choose to slice the array into two parts of size $\frac{n}{4}$ and one part of size $\frac{n}{2}$. This can be done using only divisions by powers of two, which are simple binary shifts, as in the initial binary search (see SkewSearch in Figure 7).

5.2 Experiments

As expected at this point in our work, the BiasedBinarySearch experimentally performs better than the classical binary search and the SkewSearch performs much better. Unlike our previous examples, the changes we brought in the binary search are quite sensitive to cache effects, since the way we partition the array influences the location where the memory is accessed. Thus we conducted experiments on arrays that fit in the last-level cache of our

BiasedBinarySearch

```
1  d = 0; f = n;
2  while (d < f){
3      m = (3*d+f)/4;
4      if (T[m] < x)
5          d = m+1;
6      else
7          f = m;
8  }
9  return f;
```

In both cases, `T` is an array of floats of size `n` and `x` is the number that is searched for. The classical binary search is obtained by replacing line 3 of BiasedBinarySearch by `m = (d+f)/2;`

SkewSearch

```
1  d = 0; f = n;
2  while (d < f){
3      m1 = (3*d+f)/4;
4      if (T[m1] > x)
5          f = m1;
6      else {
7          m2 = (d+f)/2;
8          if (T[m2] > x){
9              f = m2;
10             d = m1+1;
11         }
12         else d = m2+1;
13     }
14 }
15 return f;
```

Figure 7 Algorithms for the biased binary search and skew search. Both return the position where the element should be inserted.

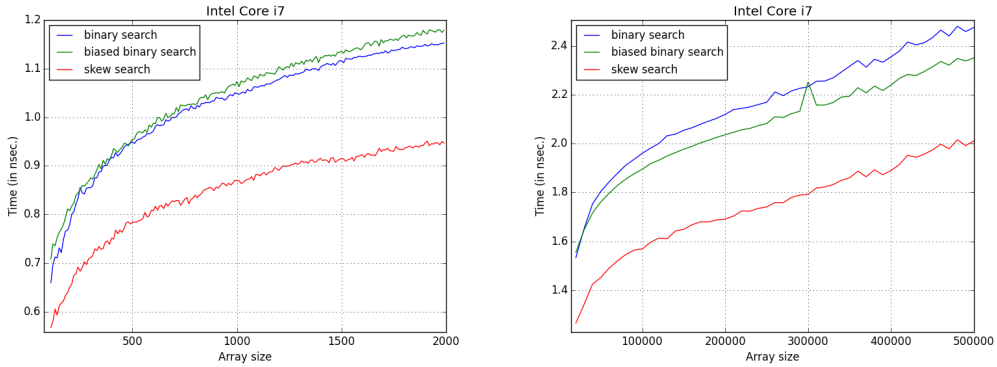


Figure 8 Execution time of the three searching algorithms of Figure 7 for small-size arrays (that fit in the first-level cache) and medium-size arrays (that fit in the last-level cache).

machine[2] in order to mostly measure the effects of branch prediction. The results are depicted in Figure 8: one can see that, for medium-size arrays, SkewSearch is up to 23% faster than the classical binary search (the programs were compiled with `gcc` without optimization, in order to keep track of what really happens during the execution). Experiments in JAVA using a dedicated micro-benchmarking library[9] gave roughly the same results (but with a lesser speedup of about 12%), when comparing our skew search to the implementation of the binary search on doubles in the standard library.

5.3 Local Predictors Analysis

As in Section 4, we aim at using the Ergodic Theorem (page 7) to obtain a good asymptotic estimate of the number of mispredictions. We therefore need to compute the expected number of times each given conditional is performed, in our different algorithms. We consider that

[9] Benchmark using jmh: `http://openjdk.java.net/projects/code-tools/jmh/`. Our algorithms are compared to `Arrays.binarySearch(double[] a, double key)`.

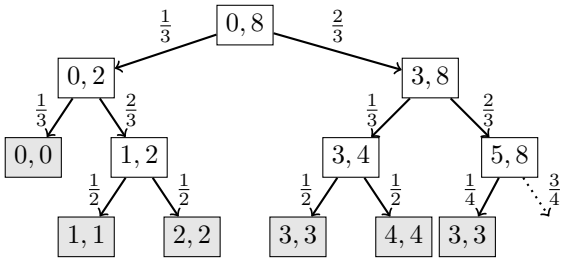


Figure 9 The decomposition tree of BiasedBinarySearch for $n = 8$.

each possible output is equally likely (*i.e.* the uniform distribution on $\{0, \dots, n\}$).

A first order estimation of the expected number of times a given conditional is executed can be obtained using the following version of Roura's Master Theorem [12], which has been simplified for our specific case:[10]

▶ **Theorem 4** (Master Theorem). *Let $k \geq 1$, and $a_1, \dots, a_k$ and $b_1, \dots, b_k$ be positive real numbers such that $\sum_{i=1}^{k} a_i = 1$. For every $i \in \{1, \dots, k\}$, let also $\varepsilon_i(n)$ be a real valued sequence such that $b_i n + \varepsilon_i(n)$ is a positive integer and $\varepsilon_i(n) = \mathcal{O}(\frac{1}{n})$. Let $T(n)$ be the real valued sequence that satisfies, for some positive constants c and d,*

$$T(0) = c \quad \text{and} \quad T(n) = d + \sum_{i=1}^{k} a_i T\left(b_i n + \varepsilon_i(n)\right) + \mathcal{O}\left(\frac{\log n}{n}\right), \text{ for } n \geq 1.$$

Then $T(n) \sim \frac{d}{h} \log n$, with $h = -\sum_{i=1}^{k} a_i \log b_i$.

Before stating our main result, we describe the main steps of our analysis on the algorithm BiasedBinarySearch. The expected number of iterations $L(n)$ of BiasedBinarySearch satisfies the relation

$$L(n) = 1 + \frac{a_n}{n+1} L\left(a_n\right) + \frac{b_n}{n+1} L\left(b_n\right), \text{ with } a_n = \left\lfloor \frac{n}{4} \right\rfloor + 1, b_n = \left\lceil \frac{3n}{4} \right\rceil \text{ and } L(0) = 0.$$

Thus, Theorem 4 applies and $L(n) \sim \lambda \log n$, with $\lambda = \frac{4}{4 \log 4 - 3 \log 3} \approx 1.78$.

Unfortunately, we cannot directly transform the predictor into a Markov chain as we did in Section 4, because the probabilities $\frac{a_n}{n+1}$ and $\frac{b_n}{n+1}$ are not fixed anymore (they slightly depend on n). However, since $\frac{a_n}{n+1} = \frac{1}{4} + \mathcal{O}(\frac{1}{n})$ and $\frac{b_n}{n+1} = \frac{3}{4} + \mathcal{O}(\frac{1}{n})$, this Markov chain should still yield a good approximation of the number of mispredictions with Theorem 2.

A convenient way to prove this formally is to introduce the *decomposition tree* $\mathcal{T}$ associated with the search algorithms, which is defined as follows. If the input has size n, its root is labeled by the pair $(0, n)$, and each node corresponds to the possible values of d and f during one loop of the algorithm. The leaves are the pairs (i, i), for $i \in \{0, \dots, n\}$; they are identified with the output of the algorithm in $\{0, \dots, n\}$. There is a direct edge between (d, f) and (d', f') whenever the variables d and f can be changed into d' and f' during the current iteration of the loop. Such an edge is labeled with the probability $\frac{f'-d'+1}{f-d+1}$, which is the probability that this update happens in our model. An example of such a decomposition tree for BiasedBinarySearch is depicted in Figure 9.

By construction, following a path from the root to a leaf, by choosing between left and right according to the edge probability is exactly the same as choosing an integer uniformly

[10] For more general statements, we refer the reader to the seminal work of Roura [12].

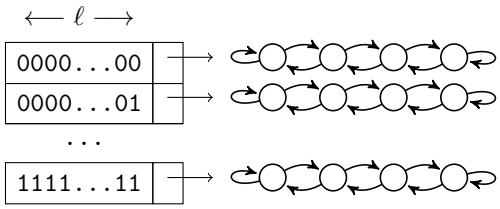


Figure 10 A fully global predictor scheme: The history table of size 2^ℓ keeps track of the outcomes of the last ℓ branches encountered during the execution, the last one corresponding to the rightmost bit. To each sequence of ℓ branches is associated a global 2-bit predictor (shared by all the conditional branches).

at random in $\{0,\dots,n\}$. Let $u = (u_0, u_1, \dots)$ be a infinite sequence of elements of $[0,1]$ taken uniformly at random and independently. To u is associated its path $\mathbf{Path}_n(\mathcal{T}, u)$ in $\mathcal{T}$ where, at step i, we go to the left if u_i is smaller than the left child edge probability and to the right otherwise. Let $L_n(\mathcal{T}, u)$ be the length of $\mathbf{Path}_n(\mathcal{T}, u)$. Let also $\mathbf{Path}_n(\mathcal{I}, u)$ be the path following the values in u in the ideal (infinite) tree $\mathcal{I}$ where we go to the left with probability $\frac{1}{4}$ and to the right with probability $\frac{3}{4}$. Then the following result holds.

▶ **Lemma 5.** *The probability that* $\boldsymbol{Path}_n(\mathcal{T}, u)$ *and* $\boldsymbol{Path}_n(\mathcal{I}, u)$ *differ at one of the first* $L_n(\mathcal{T}, u) - \sqrt{\log n}$ *steps is* $\mathcal{O}(\frac{1}{\log n})$.

Hence, the algorithm BiasedBinarySearch behaves almost like the idealized version, for most of the iterations of its main loop, and we have a sufficiently precise estimation of the error term. This is enough to prove that the idealized version is a correct first order approximation of the number of mispredictions. The same construction can be done for all three algorithms, yielding Theorem 6. For instance, with a 2-bit saturated counter, $\mu(\frac{1}{4}) = \frac{3}{10}$ and $\mu(\frac{1}{3}) = \frac{2}{5}$, thus $\mathbb{E}[C_n]/\log(n)$ is around 1.44, 1.78 and 1.68 for the binary, biased and skew search respectively, while $\mathbb{E}[M_n]/\log(n)$ is around 0.72, 0.53 and 0.58.

▶ **Theorem 6.** *Let* C_n *and* M_n *be the number of comparisons and mispredictions performed in our model of randomness. The following table give asymptotic equivalents,*

	BinarySearch	BiasedBinarySearch	SkewSearch
$\mathbb{E}[C_n]$	$\log n/\log 2$	$4\log n/(4\log 4 - 3\log 3)$	$7\log n/(6\log 2)$
$\mathbb{E}[M_n]$	$\log n/(2\log 2)$	$\mu(1/4)\mathbb{E}[C_n]$	$\big(4\mu(1/4)/7 + 3\mu(1/3)/7\big)\mathbb{E}[C_n]$

where μ *is the expected misprediction probability associated with the predictor.*

5.4 Analysis of the Global Predictor for SkewSearch

In this section we intend to give hints about the behavior of a global branching predictor, such as the one depicted in Figure 10 (see also Section 2), for the algorithm SkewSearch. Notice in particular that the predictor of each entry is a 2-bit saturated counter. This is not the only possible choice of a global predictor, but it is simple enough without being trivial. We make the analysis in the idealized framework that resemble the real case sufficiently well, by ignoring the rounding effects of dealing with integers. We saw in the previous section why these approximations still give the correct result for the first order asymptotic.

In our idealized model we only consider the sequence of taken / not taken produced by the two conditional tests of SkewSearch. We deliberately do not consider the conditional induced by the test within the "while" loop, which would be always not taken in our settings

Figure 11 On the left, the automaton $\mathcal{A}_{\text{if}}$. On the right, the Markovian automaton $\mathcal{M}_{\text{if}}$ of transition probabilities $\mathbb{P}(1 \mid \texttt{main}) = \frac{1}{4}$, $\mathbb{P}(0 \mid \texttt{main}) = \frac{3}{4}$, $\mathbb{P}(0 \mid \texttt{nested}) = \frac{2}{3}$ and $\mathbb{P}(1 \mid \texttt{nested}) = \frac{1}{3}$.

(except for the very last step). Adding it would complicate the model without adding interesting information to the branch predictor.[11] We encode a taken conditional by a 1 and a not taken conditional by a 0. The trace of an execution of the algorithm is thus a nonempty word on the binary alphabet $B = \{0, 1\}$. Because of the way the two conditional tests are nested within the algorithm, we can keep track of the current "if" by the use of the simple deterministic automaton $\mathcal{A}_{\text{if}}$ with two states depicted in Figure 11: `main` stands for the first conditional and `nested` for the second one. In our model, `main` is taken with probability $\frac{1}{4}$ and `nested` with probability $\frac{1}{3}$. As done in Section 4, $\mathcal{A}_{\text{if}}$ can be changed into a Markov chain $\mathcal{M}_{\text{if}}$ using this transition probabilities. A direct computation shows that its stationary vector π_{if} satisfies $\pi_{\text{if}}(\texttt{main}) = \frac{4}{7}$ and $\pi_{\text{if}}(\texttt{nested}) = \frac{3}{7}$.

For the same reason as above, in the global table, we only record the history for the two conditionals `main` and `nested`. Let ℓ denote the history length, that is, the number of bits used in the history table of Figure 10. We assume that ℓ is even. An *history* h is thus seen as a binary word of length ℓ. Let 0^ℓ be the history made of 0's only.

When a conditional is tested at time t, the predictor uses the entry at position h_t to make the prediction, where h_t is the current history. To follow the evolution of the algorithm at time $t+1$, we therefore only have to keep track of (1) the history table T_t, (2) the current history h_t and (3) which of the two conditionals IF_t is under consideration. Knowing IF_t is required in order to compute the probability that the next outcome is 0 or 1. This defines a Markov chain $\mathcal{M}_{\text{up}}$ for the updates in the history table. From $\mathcal{M}_{\text{up}}$, one can theoretically estimate the expected number of mispredictions using Theorem 2, as we did for local predictors. The main issue with this approach is that computing π_{up} is typically in $\mathcal{O}(m^3)$, where m is the number of states of $\mathcal{M}_{\text{up}}$. Since the number of states is exponential in ℓ, the computations are completely intractable for reasonable history lengths (such as $\ell \geq 6$), even if we first remove the unreachable states. In the sequel, we therefore use the particular structure of $\mathcal{M}_{\text{up}}$ to directly compute the typical number of mispredictions.

Let $h \in B^\ell$ be an history that is not equal to 0^ℓ. There is at least one 1 in h. Since reading a 1 always send to state `main` in $\mathcal{A}_{\text{if}}$, we know for sure the conditional IF_t under consideration when an occurrence of h has just happened at time t. Hence, we know the probability to have a 0 or a 1 at time $t+1$, given that $h_t = h$. As a consequence, each entry of $h \neq 0^\ell$ in the table T behaves like a fixed-probability local 2-bit saturating predictor, with probability $\frac{1}{4}$ (resp. $\frac{1}{3}$) for histories associated to `main` (resp. to `nested`). Therefore, $h = 0^\ell$ concentrates all the differences between the local and the global predictors.

What happens for the entry 0^ℓ is well described by considering the automaton on pairs (s, i), where s is a state of the predictor and i is the current conditional. This automaton can be turned into a Markov chain, and the Ergodic Theorem yields a precise estimation of the number of mispredictions. Following this idea yields the following result.

[11] Also, most modern architectures have "loop detectors" that are used to identify such conditionals.

▶ **Theorem 7.** *For the global predictor, the average number of mispredictions caused during* SkewSearch *on an input of size n is asymptotically equivalent to* $(\frac{12}{35} + \frac{1}{595 \cdot 2^\ell})\mathbb{E}[C_n]$.

By Theorem 6, if we use a local 2-bit predictor for each conditional, the expected number of mispredictions is asymptotically equivalent to $\frac{12}{35}\mathbb{E}[C_n]$. The difference with the global predictor is therefore extremely small, which is not surprising as there is a difference only when the history is 0^ℓ. However, if there is a competition between a global predictor and a more accurate local predictor (a 3-bit saturated counter for instance), then the local predictor performs better; it is probably slightly disrupted by the global one, as the dynamic selector between both predictors can choose to follow the global predictor from time to time.

6 Conclusion

In this article we propose unbalanced predictor-friendly versions of two very classical algorithms, namely the exponentiation by squaring and the binary search. Using a precise estimation on the expected number of mispredictions, we show that our new algorithms are worth considering when the cost of a comparison is reasonable compared to the cost of a misprediction. This is typically the case for primitive data types.

We believe that these theoretical results, supported by experiments, advocate strongly for considering this particular feature of modern computers in the design and analysis of algorithms: we showed that taking branch prediction into account can yield significant improvements, even on very classical algorithms.

References

1. Paul Biggar, Nicholas Nash, Kevin Williams, and David Gregg. An experimental study of sorting and branch prediction. *Journal of Experimental Algorithmics*, 12:1, June 2008. doi:10.1145/1227161.1370599.
2. Gerth Stølting Brodal, Rolf Fagerberg, and Gabriel Moruz. On the adaptiveness of quicksort. *ACM Journal of Experimental Algorithmics*, 12, 2008. doi:10.1145/1227161.1402294.
3. Gerth Stølting Brodal and Gabriel Moruz. Tradeoffs Between Branch Mispredictions and Comparisons for Sorting Algorithms. In *Algorithms and Data Structures*, volume 3608, pages 385–395. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.
4. Gerth Stølting Brodal and Gabriel Moruz. Skewed Binary Search Trees. In *Algorithms – ESA 2006*, volume 4168, pages 708–719. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
5. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, third edition, 2009.
6. Amr Elmasry, Jyrki Katajainen, and Max Stenmark. Branch Mispredictions Don't Affect Mergesort. In *Experimental Algorithms*, volume 7276, pages 160–171. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
7. John L. Hennessy and David A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2011.
8. Kanela Kaligosi and Peter Sanders. How Branch Mispredictions Affect Quicksort. In *Algorithms – ESA 2006*, volume 4168, pages 780–791. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
9. David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov Chains and Mixing Times*. American Mathematical Society, 2008. URL: http://pages.uoregon.edu/dlevin/MARKOV/markovmixing.pdf.

10 Conrado Martínez, Markus E. Nebel, and Sebastian Wild. Analysis of branch misses in quicksort. In *Proceedings of the Twelfth Workshop on Analytic Algorithmics and Combinatorics, ANALCO 2015, San Diego, CA, USA, January 4, 2015*, pages 114–128, 2015. doi:10.1137/1.9781611973761.11.

11 Ira Pohl. A sorting problem and its complexity. *Communications of the ACM*, 15(6):462–464, 1972.

12 Salvador Roura. Improved master theorems for divide-and-conquer recurrences. *Journal of the ACM*, 48(2):170–205, 2001. doi:10.1145/375827.375837.

13 Peter Sanders and Sebastian Winkel. Super scalar sample sort. In *Algorithms – ESA 2004*, volume 3221 of *Lecture Notes in Computer Science*, pages 784–796. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-30140-0_69.

Dense Subset Sum May Be the Hardest

Per Austrin[1], Petteri Kaski[2], Mikko Koivisto[3], and Jesper Nederlof[4]

1 **School of Computer Science and Communication, KTH Royal Institute of Technology, Sweden**
austrin@csc.kth.se
2 **Helsinki Institute for Information Technology HIIT & Department of Computer Science, Aalto University, Finland**
petteri.kaski@aalto.fi
3 **Helsinki Institute for Information Technology HIIT & Department of Computer Science, University of Helsinki, Finland**
mikko.koivisto@helsinki.fi
4 **Department of Mathematics and Computer Science, Technical University of Eindhoven, The Netherlands**
j.nederlof@tue.nl

Abstract

The Subset Sum problem asks whether a given set of n positive integers contains a subset of elements that sum up to a given target t. It is an outstanding open question whether the $O^*(2^{n/2})$-time algorithm for Subset Sum by Horowitz and Sahni [J. ACM 1974] can be beaten in the worst-case setting by a "truly faster", $O^*(2^{(0.5-\delta)n})$-time algorithm, with some constant $\delta > 0$. Continuing an earlier work [STACS 2015], we study Subset Sum parameterized by the maximum bin size β, defined as the largest number of subsets of the n input integers that yield the same sum. For every $\epsilon > 0$ we give a truly faster algorithm for instances with $\beta \leq 2^{(0.5-\epsilon)n}$, as well as instances with $\beta \geq 2^{0.661n}$. Consequently, we also obtain a characterization in terms of the popular density parameter $n/\log_2 t$: if all instances of density at least 1.003 admit a truly faster algorithm, then so does every instance. This goes against the current intuition that instances of density 1 are the hardest, and therefore is a step toward answering the open question in the affirmative. Our results stem from a novel combinatorial analysis of mixings of earlier algorithms for Subset Sum and a study of an extremal question in additive combinatorics connected to the problem of Uniquely Decodable Code Pairs in information theory.

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems, F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics

Keywords and phrases subset sum, additive combinatorics, exponential-time algorithm, homomorphic hashing, littlewood–offord problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.13

1 Introduction

The Subset Sum problem and its generalization to the Knapsack problem are two of the most famous NP-complete problems. In the Subset Sum problem, we are given positive integers $w_1, w_2, \ldots, w_n, t \in \mathbb{Z}$ as input, and need to decide whether there exists a subset $X \subseteq [n]$ with $\sum_{j \in X} w_j = t$. In the Knapsack problem, we are additionally given integers $v_1, v_2, \ldots, v_n$ and are asked to find a subset $X \subseteq [n]$ maximizing $\sum_{j \in X} v_j$ subject to the constraint $\sum_{j \in X} w_j \leq t$. While the study of Subset Sum is, among others, motivated by cryptographic applications or balancing problems, Knapsack has numerous applications


© Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof;
licensed under Creative Commons License CC-BY
33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 13; pp. 13:1–13:14

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


in combinatorial optimization. We study the exact worst-case time complexity of these problems. The earliest and probably most important algorithms for both problems are simple applications of dynamic programming, pioneered by Bellman [5], solving both problems in $O^*(t)$ time (where the $O^*(\cdot)$ notation suppresses factors polynomial in the input size). In terms of n, the best algorithms for both problems are due to Schroeppel and Shamir [18], using $O^*(2^{n/2})$ time and $O^*(2^{n/4})$ space, based on the *meet-in-the-middle* technique by Horowitz and Sahni [9]. Nederlof et al. [16] show that there is an $O^*(T^n)$-time, $O^*(S^n)$-space algorithm for Subset Sum if and only if there is an $O^*(T^n)$-time, $O^*(S^n)$-space algorithm for Knapsack. A major open question since the paper by Horowitz and Sahni [9] is whether we can do "truly faster" for both problems:

Open Question 1: Can Subset Sum be solved in $O^*\big(2^{(0.5-\delta)n}\big)$ time for some constant $\delta > 0$?

In this paper we discuss Monte Carlo algorithms in the following sense: the algorithm never returns false positives and constructs solutions of yes-instances with at least inverse polynomial probability. All randomized algorithms discussed in this paper are of this type, but for Open Question 1 we would be satisfied with two-sided error as well.

Zooming out, one motivation of this question is as follows. It is commonly believed that there are no polynomial time or even sub-exponential time algorithms for Subset Sum. So how fast can the fastest algorithm be? It would be an elegant situation if the simple meet-in-the-middle algorithm was optimal. But this would also be quite surprising, and so we aim to show that at least this is not the case.

In 2010, Howgrave-Graham and Joux [10] gave an algorithm that answered Open Question 1 in the affirmative in an *average case* setting. To state their result, let us describe the setting where it applies. The *density* of a Subset Sum instance is defined as $n/\log_2 t$. A random instance of density $d > 0$ is constructed by fixing $t \approx 2^{n/d}$ and picking the integers $w_1, \ldots, w_n, t$ independently and uniformly at random between 1 and $2^{n/d}$. Howgrave-Graham and Joux [10] showed that random instances of density 1 can be solved in $O^*(2^{0.311n})$ time, and later this has been improved to $O^*(2^{0.291n})$ time by Becker et al. [4]. These results resolve Open Question 1 in the average case setting since Impagliazzo and Naor [11] showed that random instances are the *hardest when they have density* 1. Indeed, a vast body of research has given better algorithms for random instances with density deviating from 1, like reductions of sparse instances to the shortest vector problem (e.g. [14, 6]) and the algorithm by Flaxman and Przydatek [7].

The algorithms discussed thus far all use exponential space, which can be a serious bottleneck. Therefore many studies also emphasize the setting where the algorithm is restricted to using polynomial space. It is known that the running time of the dynamic programming based algorithms can be achieved also in polynomial space: Lokshtanov and Nederlof [15] give polynomial space algorithms solving Subset Sum in $O^*(t)$ time and Knapsack in pseudo-polynomial time. On the other hand, in terms of n, no polynomial space algorithm significantly faster than naïvely going through all 2^n subsets is known, and the following has been stated as an open problem by a number of researchers (see e.g. [20, 8]):

Open Question 2: Can Subset Sum be solved in polynomial space and $O^*\big(2^{(1-\delta)n}\big)$ time for some constant $\delta > 0$?

1.1 Our results

We aim to make progress on Open Question 1, and show that a large class of instances can be solved truly faster. An optimist may interpret this as an indication that truly faster algorithms indeed exist, while a pessimist may conclude the remaining instances must be the (strictly) hardest instances.

Algorithmic Results. To define classes of instances that admit truly faster algorithms, we consider several natural parameters. The key parameter that seems to capture the range of our algorithmic technique the best is the *maximum bin size* $\beta(w) = \max_{x\in\mathbb{Z}} |\{S \subseteq [n] : \sum_{i\in S} w_i = x\}|$. Our main technical result is:

▶ **Theorem 1.1.** *There exists a Monte Carlo algorithm that, for any $0 \leq \epsilon \leq 1/6$, solves all instances of* Subset Sum *with $\beta(w) \leq 2^{(0.5-\epsilon)n}$ in $O^*\big(2^{(0.5-\epsilon/4+3\epsilon^2/4)n}\big)$ time.*

We have not optimized the precise constants in Theorem 1.1 – the main message is that any instance with bin size up to $2^{(0.5-\epsilon)n}$ can be solved in time $2^{(0.5-\Omega(\epsilon))n}$. For $\epsilon \geq 1/6$, the running time of $2^{23n/48}$ obtained for $\epsilon = 1/6$ is still valid since $2^{(0.5-1/6)n}$ remains an upper bound on $\beta(w)$. In a previous work [2], we solved Subset Sum in time $O^*(2^{0.3399n}\beta(w)^4)$, which is faster than Theorem 1.1 for small $\beta(w)$, but Theorem 1.1 shows that we can beat the meet-in-the-middle bound for a much wider class of instances.

From the other end, we also prove that when the maximum bin size becomes too large, we can again solve Subset Sum truly faster:

▶ **Theorem 1.2.** *There exist a constant $\delta > 0$ and a deterministic algorithm that solves all instances of* Subset Sum *with $\beta(w) \geq 2^{0.661n}$ in $O^*\big(2^{(0.5-\delta)n}\big)$ time.*

Combinatorial Results. Given Theorem 1.1, the natural question is how instances with $\beta(w) \geq 2^{0.5n}$ look like. This question is an instantiation of the inverse Littlewood-Offord problem, a subject well-studied in the field of additive combinatorics. Ideally we would like to find structural properties of instances with $\beta(w) \geq 2^{0.5n}$, that can be algorithmically exploited by other means than Theorem 1.1 in order to resolve Open Question 1 in the affirmative. While there is a large amount of literature on the inverse Littlewood-Offord problem, the typical range of $\beta(w)$ studied there is $\beta(w) = 2^n/\operatorname{poly}(n)$ which is not relevant for our purposes. However, we did manage to determine additional properties that any instance that is not solved by Theorem 1.1 must satisfy.

In particular, we study a different natural parameter, the *number of distinct sums* generated by w, defined as $|w(2^{[n]})| = \{w(X) : X \subseteq [n]\}$ (where we denote $w(X) = \sum_{i\in X} w_i$). This parameter can be viewed as a measure of the "true" density of an instance, in the following sense. An instance with density $d = n/\log_2 t$ has $|w(2^{[n]})| \leq n2^{n/d}$ (assuming without loss of generality that $t \leq \max_i w_i$). On the other hand, by standard hashing arguments (e.g., Lemma 2.2 with $B = 10|w(2^{[n]})|$), any instance can be hashed down to an equivalent instance of density roughly $n/\log_2 |w(2^{[n]})|$.

The relationship between $|w(2^{[n]})|$ and $\beta(w)$ is more complicated. Intuitively, one would expect that if one has so much concentration that $\beta(w) \geq 2^{0.5n}$, then w should not generate too many sums. We are not aware of any such results from the additive combinatorics literature. However, by establishing a new connection to *Uniquely Decodable Code Pairs*, a well-studied object in information theory, we can derive the following bound.

▶ **Lemma 1.3.** *If $|w(2^{[n]})| \geq 2^{0.997n}$ then $\beta(w) \leq 2^{0.4996n}$.*

Unfortunately, we currently do not know how to algorithmically exploit $|w(2^{[n]})| \leq 2^{0.997n}$. But we do know how to exploit a set S with $|S| \leq n/2$ and $|w(2^S)| \leq 2^{0.4999n}$ (see Lemma 3.2). This suggests the question of how large $\beta(w)$ can be in instances lacking such an S, and we prove the following bound.

▶ **Lemma 1.4.** *There is a universal constant $\delta > 0$ such that the following holds for all sufficiently large n. Let S, T be a partition of $[n]$ with $|S| = |T| = n/2$ such that $|w(2^S)|, |w(2^T)| \geq 2^{(1/2-\delta)n}$. Then $\beta(w) \leq 2^{0.661n}$.*

Further Consequences. Combining Lemma 1.3 and Theorem 1.1, we see directly that instances that generate almost 2^n distinct sums can be solved faster than $2^{0.5n}$.

▶ **Theorem 1.5.** *There exists a Monte Carlo algorithm that solves all instances of* Subset Sum *with $|w(2^{[n]})| \geq 2^{0.997n}$ in time $O^*(2^{0.49991n})$.*

Combining this with the view described above of $|w(2^{[n]})|$ as a refined version of the density of an instance, we have the following result, to support the title of our paper:

▶ **Theorem 1.6.** *Suppose there exist a constant $\epsilon > 0$ and an algorithm that solves all* Subset Sum *instances of density at least* 1.003 *in time $O^*(2^{(0.5-\epsilon)n})$. Then there exists a Monte Carlo algorithm that solves* Subset Sum *in time $O^*\big(2^{\max\{0.49991, 0.5-\epsilon\}n}\big)$.*

After the result by Howgrave-Graham and Joux [10], this may be a next step towards resolving Open Question 1. Intuitively, one should be able to exploit the fact that the integers in a dense instance have fewer than n bits. For example, even if only the target is picked uniformly at random, in expectation there will be an exponential number of solutions, which can easily be exploited.[1]

Finally, let us note a somewhat curious consequence of our results. As mentioned earlier, in the context of Open Question 2, it is known that the $O^*(2^{n/d})$ running time for instances of density d achieved through dynamic programming can be achieved in polynomial space [15] (see also [13, Theorem 1(a)]). Combining this with Corollary 1.5 and hashing, we directly get the following "interleaving" of Open Questions 1 and 2.

▶ **Corollary 1.7.** *There exist two Monte Carlo algorithms, one running in $O^*(2^{0.49991n})$ time and the other in $O^*(2^{0.999n})$ time and polynomial space, such that every instance of* Subset Sum *is solved by at least one of the algorithms.*

Organization of the paper. This paper is organized as follows: In Section 2 we review some preliminaries. In Section 3, we provide the proofs of our main algorithmic results. In Section 4 we prove two combinatorial lemmas. In Section 5 we give the proof for Theorem 1.6. Finally we end with some discussion on in Section 6.

2 Preliminaries

For a modulus $m \in \mathbb{Z}_{\geq 1}$ and $x, y \in \mathbb{Z}$, we write $x \equiv y \pmod m$, or $x \equiv_m y$ for short, to indicate that m divides $x - y$. Throughout this paper, $w_1, w_2, \ldots, w_n, t$ will denote the input

[1] For example, assuming there are at least $2^{\sigma n}$ solutions for a constant $\sigma \geq 0$, use a dynamic programming table data structure to randomly sample the subsets in the congruence class t mod q for q a random prime with about $(1-\sigma)n/2$ bits within linear time per sample. A solution is found within $O^*(2^{(1-\sigma)n/2})$ samples with high probability.

integers of a Subset Sum instance. We associate the set function $w : 2^{[n]} \to \mathbb{Z}$ with these integers by letting $w(X) = \sum_{i \in X} w_i$, and for a set family $\mathcal{F} \subseteq 2^{[n]}$ we write $w(\mathcal{F})$ for the image $\{w(X) : X \in \mathcal{F}\}$.

For $0 \leq x_1, x_2, \ldots, x_\ell \leq 1$ with $\sum_{i=1}^{\ell} x_i = 1$ we write $h(x_1, x_2, \ldots, x_\ell) = \sum_{i=1}^{\ell} -x_i \log_2 x_i$ for the entropy function. Here, $0 \log_2 0$ should be interpreted as 0. We shorthand $h(x, 1-x)$ with $h(x)$. We routinely use the standard fact (easily proved using Stirling's formula) that for non-negative integers $n_1, \ldots, n_\ell$ (where ℓ is a constant) summing to n, it holds that $\binom{n}{n_1,\ldots,n_\ell} = 2^{h(n_1/n,\ldots,n_\ell/n)n} \cdot \mathrm{poly}(n)$.

▶ **Claim 2.1.** *For every sufficiently large integer r the following holds. If p is a prime between r and $2r$ selected uniformly at random and x is a nonzero integer, then p divides x with probability at most $(\log_2 x)/r$.*

▶ **Lemma 2.2** (Bit-length reduction)**.** *There exists a randomized algorithm that takes as input a Subset Sum instance $w_1, w_2, \ldots, w_n, t \in \mathbb{Z}$ and an integer $B \in \mathbb{Z}$, and in time $O^*(1)$ outputs a new Subset Sum instance $w'_1, w'_2, \ldots, w'_n, t' \in \mathbb{Z}$ such that with probability at least inversely polynomial in the input size, the following properties all simultaneously hold.*

1. *$0 \leq w'_1, w'_2, \ldots, w'_n, t' < 4nB \log_2 B$.*
2. *If $B \geq 10 \cdot |w(2^{[n]})|$, then $X \subseteq [n]$ satisfies $w(X) = t$ if and only if $w'(X) = t'$.*
3. *If $B \geq 10 \cdot |w(2^{[n]})|$, then $|w(2^{[n]})|/2 \leq |w'(2^{[n]})| \leq n|w(2^{[n]})|$.*
4. *If $B \geq 5 \cdot |w(2^{[n]})|^2$, then $\beta(w)/n \leq \beta(w') \leq \beta(w)$.*

The proofs of Claim 2.1 and Lemma 2.2 use standard techniques and are presented in the full version [3].

3 Algorithmic Results

This section establishes Theorems 1.1 and 1.2. We begin with two lemmas showing how one can exploit a subset of the input integers if it generates either many or few distinct sums. The case of many sums is the main technical challenge and addressed by the following result, which is our main algorithmic contribution.

▶ **Lemma 3.1.** *There is a randomized algorithm that, given positive integers $w_1, \ldots, w_n, t \leq 2^{O(n)}$ and a set $M \in \binom{[n]}{\mu n}$ satisfying $\mu \leq 0.5$ and $|w(2^M)| \geq 2^{\gamma|M|}$ for some $\gamma \in [0, 1]$, finds a subset $X \subseteq [n]$ satisfying $w(X) = t$ with probability at least inversely polynomial in the input size (if such an X exists) in time $O^*\big(2^{(0.5+0.8113\mu-\gamma\mu)n} + \beta(w)2^{(1.5-\gamma)\mu n}\big)$.*

The proof is given in Section 3.1. Informally, it uses an algorithm that simultaneously applies the meet-in-the-middle technique of Horowitz and Sahni [9] on the set $[n] \setminus M$ and the "representation technique" of Howgrave-Graham and Joux [10] on the set M. Specifically, we pick an arbitrary equi-sized partition L, R of $[n] \setminus M$ and construct lists $\mathcal{L} \subseteq 2^{L \cup M}$ and $\mathcal{R} \subseteq 2^{R \cup M}$. Note that without restrictions on $\mathcal{L}$ and $\mathcal{R}$, one solution X is witnessed by $2^{|M \cap X|}$ pairs (S, T) from $\mathcal{L} \times \mathcal{R}$ in the sense that $S \cup T = X$. Now the crux is that since M generates many sums, $M \cap X$ generates many sums (say $2^{\pi|M|}$): this allows us to uniformly choose a congruence class t_L of $\mathbb{Z}_p$ where p is a random prime of order $2^{\pi|M|}$ and restrict attention only to sets $S \subseteq L \cup M$ and $T \subseteq R \cup M$ such that $w(S) \equiv_p t_L$ and $w(T) \equiv_p t - t_L$, while still finding solutions with good probability. This ensures that the to-be-constructed lists $\mathcal{L}$ and $\mathcal{R}$ are small enough. As an indication for this, note that if $|M \cap X| = |M|/2$ and $|w(\binom{M \cap X}{|M|/4})|$ is $\Omega(2^{|M|/2})$, the expected sizes of $\mathcal{L}$ and $\mathcal{R}$ are at most $2^{((1-\mu)/2 + h(1/4)\mu - \mu/2)n} \leq 2^{(1/2 - 0.18\mu)n}$.

In contrast to Lemma 3.1, it is straightforward to exploit a small subset that generates few sums:

▶ **Lemma 3.2.** *There is a deterministic algorithm that, given positive integers $w_1, \ldots, w_n, t$ and a set $M \in \binom{[n]}{\mu n}$ satisfying $\mu \leq 0.5$ and $|w(2^M)| \leq 2^{\gamma|M|}$ for some $\gamma \in [0,1]$, finds a subset $X \subseteq [n]$ satisfying $w(X) = t$ (if such an X exists) in time $O^*\big(2^{\frac{1-\mu(1-\gamma)}{2}n}\big)$.*

Proof. Let L be an arbitrary subset of $[n] \setminus M$ of size $\frac{1-\mu(1-\gamma)}{2}n$ and let $R = [n] \setminus L$. Then $|w(2^L)| \leq 2^{|L|} = 2^{\frac{1-\mu(1-\gamma)}{2}n}$, and

$$|w(2^R)| \leq |w(2^M)| \cdot |w(2^{[n]\setminus L\setminus M})| \leq 2^{\gamma\mu n} 2^{\left(1-\frac{1-\mu(1-\gamma)}{2}-\mu\right)n} = 2^{\frac{1-\mu(1-\gamma)}{2}n}.$$

Now apply routine dynamic programming to construct $w(2^L)$ in time $O^*(|w(2^L)|)$ and $w(2^R)$ in time $O^*(|w(2^R)|)$; build a look-up table data structure for $w(2^L)$, and for each $x \in w(2^R)$, check in $O(n)$ time whether $t - x \in w(2^L)$. ◀

Given these lemmas, we are now in the position to exploit small bins:

Proof of Theorem 1.1. We start by preprocessing the input with Lemma 2.2, taking $B = 2^{3n} \gg |w(2^{[n]})|^2$. Let $\gamma = 1 - \epsilon/2, \mu = 3\epsilon/2$, and partition $[n]$ into $1/\mu$ parts $M_1, \ldots, M_{1/\mu}$ of size at most μn arbitrarily. We distinguish two cases. First, suppose that $|w(2^{M_i})| \geq 2^{\gamma\mu n}$ for some M_i (note that this can be easily determined within the claimed time bound). We then apply the algorithm of Lemma 3.1 with $M = M_i$ and solve the instance (with probability $\Omega^*(1)$) in time

$$O^*\Big(2^{(0.5+0.8113\mu-\gamma\mu)n} + \beta(w)2^{(1.5-\gamma)\mu n}\Big).$$

The coefficient of the exponent of the first term is $0.5 + 0.8113 \cdot 3\epsilon/2 - (1-\epsilon/2) \cdot 3\epsilon/2 = 0.5 - 0.28305\epsilon + 0.75\epsilon^2$. The coefficient of the exponent of the second term is $0.5 - \epsilon + (1.5 - (1-\epsilon/2)) \cdot 3\epsilon/2 = 0.5 - \epsilon/4 + 0.75\epsilon^2$.

Second, suppose that $|w(2^{M_i})| \leq 2^{\gamma\mu n}$ for all i. Let $L = \bigcup_{i=1}^{\frac{1}{2\mu}} M_i$ and $R = [n] \setminus M$. We see that

$$|w(2^L)| \leq \prod_{i \leq \frac{1}{2\mu}} |w(2^{M_i})| \leq 2^{\gamma n/2} \qquad \text{and} \qquad |w(2^R)| \leq \prod_{i > \frac{1}{2\mu}} |w(2^{M_i})| \leq 2^{\gamma n/2}.$$

Using standard dynamic programming to construct $w(2^L)$ and $w(2^R)$ in $O^*(|w(2^L)|)$ and $O^*(|w(2^R)|)$ time, we can therefore solve the instance within $O^*(2^{\gamma n/2}) = O^*(2^{(0.5-\epsilon/4)n})$ time using linear search. ◀

Exploiting large bins is easy using Lemma 1.4 (the proof of Lemma 1.4 is given in Section 4):

Proof of Theorem 1.2. Pick an arbitrary equi-sized partition S, T of $[n]$. By the contrapositive of Lemma 1.4, one of S and T generates at most $2^{(1/2-\delta)n}$ sums. Applying Lemma 3.2 with the set in question as M, we get a running time of $O^*\big(2^{(1-\delta)n/2}\big)$. ◀

3.1 Proof of Lemma 3.1

We now prove Lemma 3.1. Let $s := |X \cap M|$. Without loss of generality, we may assume that $s \geq |M|/2$ (by considering the actual target t and the complementary target $t' := w([n]) - t$). We may further assume that s is known by trying all $O(n)$ possible values. The algorithm is listed in Algorithm 1.

Algorithm $\mathsf{A}(w_1, \ldots, w_n, t, M, s, \gamma)$ Assumes $|w(2^M)| \geq 2^{\gamma|M|}$
Output: **yes**, if there exists an $X \subseteq [n]$ with $w(X) = t$ and $|X \cap M| = s$
1: Let $\sigma = s/|M|$
2: Let $\pi = \gamma - 1 + \sigma$
3: Pick a random prime p satisfying $2^{\pi|M|} \leq p \leq 2^{\pi|M|+1}$
4: Pick a random number $0 \leq t_L \leq p - 1$
5: **for all** $0 \leq s_1 \leq s_2 \leq |M|$ such that $s_1 + s_2 = s$ **do**
6: Let $\sigma_1 = s_1/|M|, \sigma_2 = s_2/|M|$
7: Let $\lambda = (1 - \mu)/2 + \big(h(\sigma/2) - h(\sigma_1)\big)\mu$
8: Let L, R be an *arbitrary* partition of $[n] \setminus M$ such that $|L| = \lceil \lambda n \rceil$
9: Construct $\mathcal{L} = \{S \in 2^{L \cup M} : w(S) \equiv_p t_L$ and $|S \cap M| = s_1\}$
10: Construct $\mathcal{R} = \{T \in 2^{R \cup M} : w(T) \equiv_p t - t_L$ and $|T \cap M| = s_2\}$
11: **for all** $(S, T) \in \mathcal{L} \times \mathcal{R}$ such that $w(S) + w(T) = t$ **do**
12: **if** $S \cap T = \emptyset$ **then return yes**
13: **return no**

Algorithm 1 Exploiting a small subset generating many sums.

Expected Running time. We will analyze the expected running time of Algorithm 1 in two parts: (i) the generation of the lists $\mathcal{L}$ and $\mathcal{R}$ on Lines 9 and 10, and (ii) the iteration over pairs in $\mathcal{L} \times \mathcal{R}$ in Line 11 (the typical bottleneck). Let $W_L := 2^{|L|}\binom{M}{s_1} \leq 2^{\lambda n} 2^{h(\sigma_1)\mu n} = 2^{((1-\mu)/2 + h(\sigma/2)\mu)n}$ denote the size of the search space for $\mathcal{L}$.

▶ **Proposition 3.3.** *The lists $\mathcal{L}$ and $\mathcal{R}$ in Lines 9 and 10 can be constructed in expected time $O^*\big(W_L^{1/2} + W_L/2^{\pi\mu n}\big)$, where the expectation is over the choice of p and t_L.*

Proof. By splitting the search space for $\mathcal{L}$ appropriately, we get two "halves" each of which has size $W_L^{1/2}$. Specifically, we arbitrarily pick a subset $L_1 \subseteq L$ of size $\lambda_1 n$ with $\lambda_1 = (\lambda + h(\sigma/2)\mu)/2$ and generate using brute-force $w(2^{L_1})$ and $w(\mathcal{L}_2)$ where $\mathcal{L}_2 = \{Y \cup Z : Y \subseteq L \setminus L_1$ and $Z \in \binom{M}{s_1}\}$. Then we store $w(2^{L_1})$ in a dictionary data structure and, for each sum $x \in w(\mathcal{L}_2)$, we look up all solutions with sum $t - x \bmod p$ in the dictionary of $w(2^{L_1})$ and list for such a pair its union. This yields a running time of $O^*(|\mathcal{L}| + W_L^{1/2})$. The expected size of $|\mathcal{L}|$ over the random choices of t_L is $\mathbb{E}[|\mathcal{L}|] \leq O(W_L/2^{\pi\mu n})$.

The analysis for $\mathcal{R}$ is analogous and we get a running time of $O^*\big(W_R^{1/2} + W_R/2^{\pi\mu n}\big)$ where $W_R := 2^{|R|}\binom{|M|}{s_2}$. Let $\rho = |R|/n$. Since $h(\cdot)$ is concave and, in particular, $h(\sigma_1) + h(\sigma_2) \leq 2h(\sigma/2)$, we then have (up to a negligible term caused by rounding λn to an integer)

$$\rho = 1 - \mu - \lambda = (1 - \mu)/2 - \big(h(\sigma/2) - h(\sigma_1)\big)\mu \leq (1 - \mu)/2 + (h(\sigma/2) - h(\sigma_2))\mu\,.$$

Thus the case of R is symmetric to the situation for L and $W_R \leq 2^{((1-\mu)/2 + h(\sigma/2)\mu)n} = O^*(W_L)$. ◀

The term $W_L/2^{\pi\mu n}$ can be bounded by using the definition of $\pi = \gamma - 1 + \sigma$ and we get $W_L/2^{\pi\mu n} = 2^{(\frac{1}{2} + \mu(\frac{1}{2} + h(\sigma/2) - \gamma - \sigma))n}$. Since $1/2 + h(\sigma/2) - \sigma$ subject to $1/2 \leq \sigma \leq 1$ is maximized at $\sigma = 1/2$ where it is $h(1/4) \leq 0.8113$, we have that $W_L/2^{\pi\mu n} \leq 2^{(0.5 + 0.8113\mu - \gamma\mu)n}$.

The term $W_L^{1/2}$ is naively bounded by $2^{(1+\mu)n/4}$, which is dominated by the term $O^*\big(2^{(0.5+0.8113\mu-\gamma\mu)n}\big)$ since $\mu \leq 1/2$ and $\gamma \leq 1$. It follows that Line 9 and Line 10 indeed run within the claimed time bounds.

▶ **Proposition 3.4.** *The expected number of pairs considered in Line 11 is $O^*\big(\beta(w)2^{\mu(1.5-\gamma)n}\big)$, where the expectation is over the choice of p and t_L.*

Proof. Define $\mathcal{B} = \{(P,Q) \in 2^{[n]} \times 2^M : w(P) + w(Q) = t\}$, and note that the set of pairs $(S,T) \in 2^{L\cup M} \times 2^{R\cup M}$ satisfying $w(S) + w(T) = t$ are in one-to-one correspondence with pairs in $\mathcal{B}$ (by the map $(S,T) \mapsto (S \cup (T \cap R), T \cap M)$). Furthermore, the size of $\mathcal{B}$ is bounded by $|\mathcal{B}| \leq \beta(w)2^{|M|}$: for each of the $2^{|M|}$ possible choices of Q, there are at most $\beta(w)$ subsets R that sum to $t - w(Q)$.

Any given pair $(S,T) \in 2^{L\cup M} \times 2^{R\cup M}$ satisfying $w(L) + w(R) = t$ is considered only if $w(S) \equiv_p t_L$, which happens with probability $O(2^{-\pi n})$ (over the uniformly random choice of t_L). Thus the expected number of pairs considered in Line 11 is upper bounded by $O(|B|/2^{\pi\mu n}) = O(\beta(w)2^{\mu(1-\pi)n}) = O(\beta(w)2^{\mu(2-\sigma-\gamma)n})$. Using $\sigma \geq 1/2$, the desired bound follows. ◀

Succes Probability. To establish Lemma 3.1, we run Algorithm 1 for $\Omega(|M|)$ times the expected number of computation steps and return **no** if it did not terminate yet. By our previous analysis it is clear that this algorithm runs within the required time bound, and clearly it only return **yes** if a solution is found on Line 12.

Now suppose there exists an $X \subseteq [n]$ with $w(X) = t$ and $|X \cap M| = s$. It remains to lower bound the probability that the modified algorithm return **yes**. Note that by Markov's inequality we return **no** due to premature termination with probability at most $O(1/|M|)$, so by a union bound it will be sufficient to lower bound the probability that Algorithm 1 returns **yes** by $\Omega(1/|M|)$.

To this end, note that $2^{\gamma|M|} \leq |w(2^M)| \leq |w(2^{M\cap X})| \cdot |w(2^{M\setminus X})|$, and since $|w(2^{M\setminus X})| \leq 2^{|M|-s}$, we have that $|w(2^{M\cap X})| \geq 2^{\gamma|M|-(1-\sigma)|M|} = 2^{\pi|M|}$.

Thus there must exist positive $s_1 + s_2 = s$ such that $|w(\binom{M\cap X}{s_1})| \geq 2^{\pi|M|}/|M|$. Let us focus on the corresponding iteration of Algorithm 1. Let $w_L := w(X \cap L)$ be the contribution of L to the solution X. We claim that in this iteration, the following holds.

▶ **Proposition 3.5.**

$$\Pr\left[\exists Q \in \binom{M \cap X}{s_1} : w(Q) \equiv_p t_L - w_L\right] \geq \Omega\left(\frac{1}{|M|}\right). \tag{1}$$

Note that, conditioned on the event $\exists Q \in \binom{M\cap X}{s_1} : w(Q) \equiv_p t_L - w_L$, Algorithm 1 will include $S := Q \cup (L \cap X)$ in $\mathcal{L}$ and $T := X \setminus S$ in $\mathcal{R}$ and recover X. Therefore, this concludes the proof of Lemma 3.1.

Proof. Let $\mathcal{F} \subseteq \binom{M\cap X}{s_1}$ be a maximal injective subset, i.e., satisfying $|\mathcal{F}| = |w(\mathcal{F})| = |w\left(\binom{M\cap X}{s_1}\right)| \geq \Omega^*(2^{\pi|M|})$. Let $c_i = |\{Y \in \mathcal{F} : w(Y) \equiv_p i\}|$ be the number of sets from $\mathcal{F}$ in the i'th bin mod p. Our goal is to lower bound the probability that $c_{t_L - w_L} > 0$ (where $t_L - w_L$ is taken modulo p). We can bound the expected ℓ^2 norm (e.g., the number of collisions) by

$$\mathbb{E}\left[\sum_i c_i^2\right] = \sum_{Y,Z\in\mathcal{F}} \Pr\left[p \text{ divides } w(Y) - w(Z)\right] \leq |\mathcal{F}| + O^*\big(|\mathcal{F}|^2/2^{\pi|M|}\big), \tag{2}$$

where the inequality uses Claim 2.1 and the assumption that the w_i's are $2^{O(n)}$. By Markov's inequality, $\sum_i c_i^2 \leq O^*(|\mathcal{F}|^2/2^{\pi|M|})$ with probability at least $\Omega^*(1)$ over the choice of p (here we used $|\mathcal{F}| = \Omega^*(2^{\pi|M|})$ to conclude that the second term in (2) dominates the first). Conditioned on this, Cauchy-Schwarz implies that the number of non-zero c_i's is at least $|\mathcal{F}|^2 / \sum_i c_i^2 \geq \Omega^*(2^{\pi|M|})$. When this happens, the probability that $c_{t_L - w_L} > 0$ (over the uniformly random choice of t_L) is $\Omega^*(1)$. ◀

4 Combinatorial Results (Lemma 1.3 and Lemma 1.4)

In this section we provide two non-trivial quantitative relations between several structural parameters of the weights. Our results are by no means tight, but will be sufficient for proving our main results.

For the purposes of this section, it is convenient to use vector notation for subset sums. In particular, for a vector $x \in \mathbb{Z}^n$, we write $x \cdot w = \sum_{i=1}^{n} x_i w_i$, and $x^{-1}(j) \subseteq [n]$ for the set of $i \in [n]$ such that $v_i = j$.

Our approach to relate the number of sums $|w(2^{[n]})|$ to the largest bin size $\beta(w)$ is to establish a connection to the notion of Uniquely Decodable Code Pairs from information theory, defined as follows.

▶ **Definition 4.1** (Uniquely Decodable Code Pair, UDCP)**.** If $A, B \subseteq \{0,1\}^n$ such that

$$|A + B| = |\{a + b : a \in A, b \in B\}| = |A| \cdot |B|\,,$$

then (A, B) is called *uniquely decodable*. Note that here addition is performed over $\mathbb{Z}^n$ (and **not** mod $\mathbb{Z}_2^n$).

UDCP's capture the zero error region of the so-called *binary adder channel*, and there is a fair amount of work on how large the sets A and B can be (for a survey, see [19, §3.5.1]). The connection between UDCP's and Subset Sum is that a Subset Sum instance that both generates many sums and has a large bin yields a large UDCP, as captured in the following proposition.

▶ **Proposition 4.2.** *If there exist weights $w_1, \ldots, w_n$ such that $|w(2^{[n]})| = a$ and $\beta(w) = b$, then there exists a UDCP (A, B) with $|A| = a$ and $|B| = b$.*

Proof. Let $A \subseteq \{0,1\}^n$ be an injective set, i.e., $x \cdot w \neq x' \cdot w$ for all $x, x' \in A$ with $x \neq x'$. Note that there exists such an A with $|A| = a$. Let $B \subseteq \{0,1\}^n$ be a bin, i.e., $y \cdot w = y' \cdot w$ for all $y, y' \in B$. Note that we can take these to have sizes $|A| = a$ and $|B| = b$.

We claim that (A, B) is a UDCP. To see this, let $x, x' \in A$ and $y, y' \in B$ with $x+y = x'+y'$. Then

$$x \cdot w + y \cdot w = (x + y) \cdot w = (x' + y') \cdot w = x' \cdot w + y' \cdot w\,.$$

Thus $x \cdot w = x' \cdot w$, and so by the injectivity property of A, we have $x = x'$, which in turn implies $y = y'$ since $x + y = x' + y'$. ◀

We have the following result by Ordentlich and Shayevitz [17, Theorem 1, setting $R_1 = 0.997$ and $\alpha = 0.07$].

▶ **Theorem 4.3** ([17])**.** *Let $A, B \subseteq \{0,1\}^n$ such that (A, B) is a UDCP and $|A| \geq 2^{.997n}$. Then $|B| \leq 2^{0.4996n}$.*

With this connection in place, the proof of Lemma 1.3 is immediate.

▶ **Lemma 1.3** (restated)**.** *If $|w(2^{[n]})| \geq 2^{0.997n}$, then $\beta(w) \leq 2^{0.4996n}$.*

Proof. Combine Theorem 4.3 with the contrapositive of Proposition 4.2. ◀

The remainder of this section is devoted to Lemma 1.4. The proof also (implictly) uses a connection to Uniquely Decodable Code Pairs, but here the involved sets of strings are not binary. There is no reason to believe that the constant 0.661 is tight. However, because a random instance w of density 2 satisfies the hypothesis for all partitions S, T and has $\beta(w) \approx 2^{0.5n}$ with good probability, just improving the constant 0.661 will not suffice for settling Open Question 1.

4.1 Proof of Lemma 1.4

For a subset $S \subseteq [n]$, define a function $b_S : \mathbb{Z} \to \mathbb{Z}$ by letting $b_S(x)$ be the number of subsets $S' \subseteq S$ such that $w(S') = x$. Note that $|w(2^S)|$ equals the support size of b_S, or $\|b_S\|_0$, and that $\beta_w(S) = \max_x b_S(x) = \|b_S\|_\infty$. Instead of working with these extremes, it is more convenient to work with the ℓ^2 norm of b_S, and the main technical claim to obtain Lemma 1.4 is the following.

▶ **Proposition 4.4.** *There exists a $\delta > 0$ such that for all sufficiently large $|S|$ the following holds: if $|w(2^S)| \geq 2^{(1-\delta)|S|}$, then $\|b_S\|_2 \leq 2^{0.661|S|}$.*

Proof of Proposition 4.4. Without loss of generality we take $S = [n]$, and to simplify notation we omit the subscript S from b_S and simply write $b : \mathbb{Z} \to \mathbb{Z}$ for the function such that $b(r)$ is the number of subsets of $w_1, \ldots, w_n$ summing to r. Note that

$$\|b\|_2^2 = \sum_{\substack{U,V \subseteq [n] \\ [w(U)=w(V)]}} 1 = \sum_{\substack{U,V \subseteq [n] \\ U \cap V = \emptyset \\ [w(U)=w(V)]}} 2^{n-|U|-|V|} = \sum_{y \in \{-1,0,1\}^n} [y \cdot w = 0] \cdot 2^{|y^{-1}(0)|},$$

where $[p]$ denotes 1 if p holds and 0 otherwise. Defining $B_\sigma = \left\{ y \in \{-1,0,1\}^n : y \cdot w = 0 \text{ and } \|y\|_1 = \sigma n \right\}$, we thus have

$$\|b\|_2^2 = \sum_{i=0}^{n} |B_{i/n}| 2^{n-i} \leq n \max_\sigma |B_\sigma| 2^{(1-\sigma)n} \,. \tag{3}$$

We now proceed to bound the size of B_σ by an encoding argument. To this end, let $A \subseteq \{0,1\}^n$ be a maximal injective set of vectors. In other words, $|A| = |w(2^{[n]})| \geq 2^{0.99n}$, and for all pairs $x \neq x' \in A$, it holds that $x \cdot w \neq x' \cdot w$. We claim that $|A + B_\sigma| = |A| \cdot |B_\sigma|$. To see this note that, similarly to the proof of Proposition 4.2, if $x + y = x' + y'$ (with $x, x' \in A$ and $y, y' \in B_\sigma$) then $x \cdot w = x' \cdot w$ (since $y' \cdot w = 0$) and thus $x = x'$ and $y = y'$.

Define P_σ to be all pairs (x, y) in $A \times B_\sigma$ that are *balanced*, in the sense that for some $\gamma > 0$ the following conditions hold:

$$\begin{aligned} |x^{-1}(1) \cap y^{-1}(-1)| &= \tfrac{1}{2}|y^{-1}(-1)| \pm \gamma n \,, \\ |x^{-1}(1) \cap y^{-1}(0)| &= \tfrac{1}{2}|y^{-1}(0)| \pm \gamma n \,, \\ |x^{-1}(1) \cap y^{-1}(1)| &= \tfrac{1}{2}|y^{-1}(1)| \pm \gamma n \,. \end{aligned} \tag{4}$$

▶ **Claim 4.5.** *For $\gamma = \sqrt{\delta}$ and n sufficiently large, we have that $|P_\sigma| \geq |A| \cdot |B_\sigma|/2$.*

The postponed proof of Claim 4.5 can be found in the full version [3].

Setting $\gamma = \sqrt{\delta}$, we can now proceed to upper bound $|P_\sigma|$. Consider the encoding $\eta : P_\sigma \to \{-1,0,1,2\}^n$ defined by $\eta(x, y) = x + y$. By the property $|A + B_\sigma| = |A| \cdot |B|$, it follows that η is an injection, and thus $|P_\sigma|$ equals the size of the image of η. For a pair $(x, y) \in P_\sigma$, if $y \in B_\sigma$ has $\tau\sigma n$ many 1's, and $(1-\tau)\sigma n$ many -1's, then $z = \eta(x, y)$ has the following frequency distribution:

$$\begin{aligned} \frac{|z^{-1}(-1)|}{n} &= \frac{\tau\sigma}{2} \pm o_\gamma(1) \,, & \frac{|z^{-1}(0)|}{n} &= \frac{\tau\sigma}{2} + \frac{1-\sigma}{2} \pm o_\gamma(1) \,, \\ \frac{|z^{-1}(1)|}{n} &= \frac{1-\sigma}{2} + \frac{(1-\tau)\sigma}{2} \pm o_\gamma(1) \,, & \frac{|z^{-1}(2)|}{n} &= \frac{(1-\tau)\sigma}{2} \pm o_\gamma(1) \,, \end{aligned}$$

where, for a variable ϵ, we write $o_\epsilon(1)$ to indicate a term that converges to 0 when ϵ tends to 0. Since $\gamma = \sqrt{\delta}$, we have $o_\gamma(1) = o_\delta(1)$. The number of z's with such a frequency distribution is bounded by

$$\binom{n}{\frac{\tau\sigma}{2}n, (\frac{\tau\sigma}{2} + \frac{1-\sigma}{2})n, (\frac{1-\sigma}{2} + \frac{(1-\tau)\sigma}{2})n, \frac{(1-\tau)\sigma}{2}} 2^{o_\delta(1)n}. \tag{5}$$

Then, $|P_\sigma|$ is bounded by

$$\log |P_\sigma| \leq \max_{\tau\in[0,1]} \big(g(\sigma,\tau)+o_\gamma(1)\big)n, \text{ where } g(\sigma,\tau) = h\Big(\tfrac{\tau\sigma}{2}, \tfrac{\tau\sigma}{2}+\tfrac{1-\sigma}{2}, \tfrac{1-\sigma}{2}+\tfrac{(1-\tau)\sigma}{2}, \tfrac{(1-\tau)\sigma}{2}\Big).$$

It can be verified that $g(\sigma,\tau)$ is maximized for $\tau = 1/2$ and we have

$$\max_{\tau\in[0,1]} g(\sigma,\tau) = h\big(\tfrac{\sigma}{4}, \tfrac{1}{2} - \tfrac{\sigma}{4}, \tfrac{1}{2} - \tfrac{\sigma}{4}, \tfrac{\sigma}{4}\big) = 1 + h\big(\tfrac{\sigma}{2}\big).$$

Combining this with the bounds $|P_\sigma| \geq |A| \cdot |B| \cdot 2^{-O(\delta^2)n}$ and $|A| \geq 2^{(1-\delta)n}$, we get that $|B_\sigma| \leq 2^{(h(\sigma/2)+o_\delta(1))n}$. Plugging this into (3) we see that

$$\|b\|_2^2 \leq \max_\sigma 2^{(1+h(\sigma/2)-\sigma+o_\epsilon(1))n}.$$

The expression $h(\sigma/2) - \sigma$ is maximized at $\sigma = 2/5$, and we obtain

$$\|b\|_2^2 \leq 2^{(h(1/5)+3/5+o_\epsilon(1))} \leq 2^{(1.32195+o_\delta(1))n}.$$

Thus if δ is sufficiently small, we have $\|b\|_2^2 \leq 2^{1.322n}$, as desired. ◀

Using Proposition 4.4, the desired bound of Lemma 1.4 follows immediately, since

$$\beta([n]) = \max_{x\in\mathbb{Z}} \sum_{y\in\mathbb{Z}} b_S(y) b_T(x-y) \leq \max_{x\in\mathbb{Z}} \|b_S\|_2 \|b_T\|_2 \leq 2^{0.661n},$$

where the first inequality is by Cauchy–Schwarz and the second inequality by Proposition 4.4.

5 Proof of Theorem 1.6

Proof of Theorem 1.6. Given oracle access to an algorithm that solves Subset Sum instance of density at least 1.003 in $O^*(2^{(0.5-\epsilon)n})$ time for some $\epsilon > 0$, we solve an arbitrary instance $w_1, w_2, \ldots, w_n, t$ of Subset Sum in time $O^*\big(2^{\max\{0.49991, 0.5-\epsilon\}n}\big)$ as follows.

As Step 1, run the algorithm of Theorem 1.5 for $\Theta^*(2^{0.49991n})$ timesteps. If it terminates within this number of steps, return YES if it found a solution and NO otherwise. Otherwise, as Step 2, run the preprocessing of Lemma 2.2 with $B = 10 \cdot 2^{0.997n}$. This yields a new instance with density $1/0.997 > 1.003$, which we solve using the presumed oracle for such instances. If the oracle returns a solution, we verify that it is indeed a solution to our original instance and if so return YES. Otherwise we return NO.

If there is no solution this algorithm clearly returns NO. If there is a solution and $|w(2^{[n]})| \geq 2^{0.997n}$, we find a solution with inversely polynomial probability in Step 1. If there is a solution and $|w(2^{[n]})| \leq 2^{0.997n}$, Property 2 of Lemma 2.2 guarantees that the solution to the reduced instance is a solution to the original instance with probability at least polynomial in the input size, and the oracle will then provide us with the solution. ◀

6 Further Discussion

Our original ambition was to resolve Open Question 1 affirmatively by a combination of two algorithms that exploit small and large concentration of the sums, respectively. Since we only made some partial progress on this, it remains an intruiging question whether this approach can fulfill this ambition. In this section we speculate about some further directions to explore.

Exploiting Large Density. For exploiting a density $1.003 \leq d \leq 2$, the meet-in-the-middle technique [9] does not seem directly extendable. A different, potentially more applicable $O^*(2^{n/2})$ algorithm works as follows: pick a prime p of order $2^{n/2}$, build the dynamic programming table that counts the number of subsets with sum congruence to t mod p, and use this as a data structure to uniformly sample solutions mod p with linear delay; try $O^*(2^{n/2})$ samples and declare a no-instance if no true solution is found (see also Footnote 1). As such, this does not exploit large density at all, but to this end one could seek a similar sampler that is more biased to smaller bins.

Sharper Analysis of Algorithm 1. The analysis of Algorithm 1 in Lemma 3.1, and in particular the typical bottleneck $\beta(w)2^{(1.5-\gamma)\mu n}$ in the running time, is quite naive. For example, since we can pick M as we like (and assume it generates many sums), for the algorithm to fail we need an instance where big bins are encountered by the algorithm for many choices of M. It might be a good approach to first try to extend the set of instances that can be solved 'truly faster' in this way, e.g. to the set of all instances with $\beta(w) \leq 2^{(.5+\delta)n}$ for some small $\delta > 0$. As an illustration of the looseness, let us mention that in a previous version of this manuscript, we used a more sophisticated analysis to show the following: there exists some $\delta > 0$, such that if $|w(2^{[n]})| \geq 2^{(1-\delta)n}$, then $|\{(P,Q) \in \binom{[n]}{n/2}^2 : w(P) + w(Q) = t\}| \leq 2^{0.5254n}$. We used this to show that all instances with $|w(2^{[n]})| \geq 2^{(1-\delta)n}$ can be solved via a mild variant of Algorithm 1 with $M = [n]$, indicating that Algorithm 1 gives non-trivial algorithms even for large M.

Sharper Combinatorial Bounds. Lemma 1.3 and Lemma 1.4 seem to be rather crude estimates. In fact, we don't even know the following (again, borrowing notation from the proof of Proposition 4.4):

> **Open Question 3:** Suppose $|w(2^{[n]})| \geq 2^{(1-\epsilon)n}$. Can $\beta(w)$ and $\|b_{[n]}\|_2$ be bounded by $2^{o_\epsilon(1)n}$ and $2^{(0.5+o_\epsilon(1))n}$, respectively?

Note that the second bound would follow from the first bound. Furthermore, if the second bound holds, we would be able to solve, for all $\epsilon > 0$, all instances with $|\beta(w)| \geq 2^{(0.5+\epsilon)n}$ in time $O^*(2^{(0.5-\epsilon')n})$ for some $\epsilon' > 0$ depending on ϵ, via the proof of Theorem 1.2.

In recent work [1] we proved the following modest progress:

▶ **Lemma 6.1.** *There exists $\delta > 0$ such that if $A, B \subseteq \{0,1\}^n$ is a UDCP and $|A| \geq 2^{(1-\delta)n}$, then $|B| \leq 2^{0.4115n}$.*

Plugging this into the proof of Lemma 1.3, this gives that $\beta(w) \leq 2^{(0.4115+o_\epsilon(1))n}$ in the setting of Open Question 3. We would like to remark that improving this beyond $2^{(0.25+o_\epsilon(1))n}$ via Lemma 1.3 is not possible since UDCP pairs (A,B) with $|A| \geq 2^{(1-o(1))n}$ and $|B| \geq 2^{n/4}$ do exist [12]. One may also wonder whether we can deal with instances

with $|w(2^{[n]})| \geq 2^{(0.5+\epsilon)n}$, for all $\epsilon > 0$ by arguing $\beta(w)$ must be small but this does not work directly: there are instances with $|w(2^{[n]})| = 3^{n/2}$ and $\beta(w) = 2^{n/2}$ (the instance $1, 1, 3, 3, 9, 9, 27, 27, \ldots$ has this, though it is easily attacked via Lemma 3.2).

Acknowledgements. This work was funded by the Swedish Research Council, Grant 621-2012-4546 (P.A.), the European Research Council, Starting Grant 338077 "Theory and Practice of Advanced Search and Enumeration" (P.K.), the Academy of Finland, Grant 276864 "Supple Exponential Algorithms" (M.K.), and NWO VENI project 639.021.438 (J.N.).

References

1 Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Improved uniquely decodable code pair bounds for unbalanced pairs. Unpublished.

2 Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Subset sum in the absence of concentration. 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 48–61. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. URL: `http://dx.doi.org/10.4230/LIPIcs.STACS.2015.48`, `doi:10.4230/LIPIcs.STACS.2015.48`.

3 Per Austrin, Mikko Koivisto, Petteri Kaski, and Jesper Nederlof. Dense subset sum may be the hardest. *CoRR*, abs/1508.06019, 2015. URL: `http://arxiv.org/abs/1508.06019`.

4 Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 364–385. Springer, 2011.

5 Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, N. J., 1957.

6 Matthijs J. Coster, Antoine Joux, Brian A. Lamacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. Improved low-density subset sum algorithms. *Computational Complexity*, 2:111–128, 1992.

7 Abraham Flaxman and Bartosz Przydatek. Solving medium-density subset sum problems in expected polynomial time. In Volker Diekert and Bruno Durand, editors, *STACS 2005, 22nd Annual Symposium on Theoretical Aspects of Computer Science, Stuttgart, Germany, February 24-26, 2005, Proceedings*, volume 3404 of *Lecture Notes in Computer Science*, pages 305–314. Springer, 2005. URL: `http://dx.doi.org/10.1007/978-3-540-31856-9_25`, `doi:10.1007/978-3-540-31856-9_25`.

8 Open problems for FPT school 2014. `http://fptschool.mimuw.edu.pl/opl.pdf`.

9 Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, 1974. URL: `http://doi.acm.org/10.1145/321812.321823`, `doi:10.1145/321812.321823`.

10 Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 235–256. Springer, 2010.

11 Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9(4):199–216, 1996. URL: `http://dx.doi.org/10.1007/BF00189260`, `doi:10.1007/BF00189260`.

12 T. Kasami, Shu Lin, V.K. Wei, and Saburo Yamamura. Graph theoretic approaches to the code construction for the two-user multiple-access binary adder channel. *IEEE Transactions on Information Theory*, 29(1):114–130, 1983. `doi:10.1109/TIT.1983.1056614`.

13 Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Homomorphic hashing for sparse coefficient extraction. In Dimitrios M. Thilikos and Gerhard J. Woeginger, editors, *Parameterized*

and Exact Computation – 7th International Symposium, IPEC 2012, Ljubljana, Slovenia, September 12-14, 2012. Proceedings, volume 7535 of *Lecture Notes in Computer Science*, pages 147–158. Springer, 2012. URL: http://dx.doi.org/10.1007/978-3-642-33293-7_15, doi:10.1007/978-3-642-33293-7_15.

14 Jeffrey C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, 1985.

15 Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In Leonard J. Schulman, editor, *STOC*, pages 321–330. ACM, 2010.

16 Jesper Nederlof, Erik Jan van Leeuwen, and Ruben van der Zwaan. Reducing a target interval to a few exact queries. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012 – 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 of *Lecture Notes in Computer Science*, pages 718–727. Springer, 2012. URL: http://dx.doi.org/10.1007/978-3-642-32589-2_62, doi:10.1007/978-3-642-32589-2_62.

17 Or Ordentlich and Ofer Shayevitz. A VC-dimension-based outer bound on the zero-error capacity of the binary adder channel. *CoRR*, abs/1412.8670, 2014. URL: http://arxiv.org/abs/1412.8670.

18 Richard Schroeppel and Adi Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981.

19 Christian Sleger and Alex Grant. *Coordinated Multiuser Communications*. Springer, 2006.

20 Gerhard J. Woeginger. Open problems around exact algorithms. *Discrete Appl. Math.*, 156(3):397–405, 2008. URL: http://dx.doi.org/10.1016/j.dam.2007.03.023, doi:10.1016/j.dam.2007.03.023.

Computing the L_1 Geodesic Diameter and Center of a Polygonal Domain

Sang Won Bae[1], Matias Korman[2], Joseph S. B. Mitchell[3], Yoshio Okamoto[4], Valentin Polishchuk[5], and Haitao Wang[6]

1 **Kyonggi University, Suwon, South Korea**
swbae@kgu.ac.kr
2 **Tohoku University, Sendai, Japan**
mati@dais.is.tohoku.ac.jp
3 **Stony Brook University, New York, USA**
jsbm@ams.stonybrook.edu
4 **The University of Electro-Communications, Tokyo, Japan**
okamotoy@uec.ac.jp
5 **Linköping University, Linköping, Sweden**
valentin.polishchuk@liu.se
6 **Utah State University, Utah, USA**
haitao.wang@usu.edu

Abstract

For a polygonal domain with h holes and a total of n vertices, we present algorithms that compute the L_1 geodesic diameter in $O(n^2+h^4)$ time and the L_1 geodesic center in $O((n^4+n^2h^4)\alpha(n))$ time, where $\alpha(\cdot)$ denotes the inverse Ackermann function. No algorithms were known for these problems before. For the Euclidean counterpart, the best algorithms compute the geodesic diameter in $O(n^{7.73})$ or $O(n^7(h+\log n))$ time, and compute the geodesic center in $O(n^{12+\epsilon})$ time. Therefore, our algorithms are much faster than the algorithms for the Euclidean problems. Our algorithms are based on several interesting observations on L_1 shortest paths in polygonal domains.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, I.1.2 Algorithms, I.3.5 Computational Geometry and Object Modeling

Keywords and phrases geodesic diameter, geodesic center, shortest paths, polygonal domains, L_1 metric

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.14

1 Introduction

A *polygonal domain* $\mathcal{P}$ is a closed and connected polygonal region in the plane $\mathbb{R}^2$, with $h \geq 0$ holes (i.e., simple polygons). Let n be the total number of vertices of $\mathcal{P}$. Regarding the boundary of $\mathcal{P}$ as obstacles, we consider shortest obstacle-avoiding paths lying in $\mathcal{P}$ between any two points $p, q \in \mathcal{P}$. Their *geodesic distance* $d(p, q)$ is the length of a shortest path between p and q in $\mathcal{P}$. The *geodesic diameter* (or simply *diameter*) of $\mathcal{P}$ is the maximum geodesic distance over all pairs of points $p, q \in \mathcal{P}$, i.e., $\max_{p\in\mathcal{P}} \max_{q\in\mathcal{P}} d(p, q)$. Closely related to the diameter is the min-max quantity $\min_{p\in\mathcal{P}} \max_{q\in\mathcal{P}} d(p, q)$, in which a point p^* that minimizes $\max_{q\in\mathcal{P}} d(p^*, q)$ is called a *geodesic center* (or simply *center*) of $\mathcal{P}$. Each of the above quantities is called *Euclidean* or L_1 depending on which of the Euclidean or L_1 metric is adopted to measure the length of paths.

For simple polygons (i.e., $h = 0$), the Euclidean diameter and center have been studied since the 1980s [2, 8, 23]. Hershberger and Suri [16] gave a linear-time algorithm for computing

© Sang W. Bae, Matias Korman, Joseph Mitchell, Yoshio Okamoto, Valentin Polishchuk, and Haitao Wang;
licensed under Creative Commons License CC-BY

the diameter. Pollack, Sharir, and Rote [21] gave an $O(n \log n)$ time algorithm for computing the geodesic center; recently, Ahn et al. [1] solved the problem in $O(n)$ time. For the general case (i.e., $h > 0$), the Euclidean diameter problem was solved in $O(n^{7.73})$ or $O(n^7(h + \log n))$ time [4], and the Euclidean center problem was solved in $O(n^{12+\epsilon})$ time for any $\epsilon > 0$ [5].

For the L_1 versions, the diameter and the center of simple polygons can be computed in linear time [6, 22]. In this paper, we present the first algorithms that compute the diameter and center of a polygonal domain $\mathcal{P}$ (as defined above) in $O(n^2 + h^4)$ and $O((n^4 + n^2h^4)\alpha(n))$ time, respectively, where $\alpha(\cdot)$ is the inverse Ackermann function. Comparing with the algorithms for the same problems under the Euclidean metric, our algorithms are much more efficient, especially when h is significantly smaller than n.

As discussed in [4], a main difficulty of polygonal domains seemingly arises from the fact that there can be several topologically different shortest paths between two points, which is not the case for simple polygons. Bae, Korman, and Okamoto [4] observed that the Euclidean diameter can be realized by two interior points of a polygonal domain, in which case the two points have at least five distinct shortest paths. This difficulty makes their algorithm suffer a fairly large running time. Similar issues also arise in the L_1 metric, where a diameter may also be realized by two interior points (this can be seen by easily extending the examples in [4]). Further, under the L_1 metric, it seems that at least eight topologically different shortest paths are needed to pin the solution; thus, even if we manage to adapt all techniques used in [4] to the L_1 metric, this would result in an algorithm whose running time is significantly larger than $O(n^{7.73})$.

We take a different approach from [4]. We first construct an $O(n^2)$-sized cell decomposition of $\mathcal{P}$ such that the L_1 geodesic distance function restricted in any pair of two cells can be explicitly described in $O(1)$ complexity. Consequently, the L_1 diameter and center can be obtained by exploring these cell-restricted pieces of the geodesic distance. This leads to simple algorithms that compute the diameter in $O(n^4)$ time and the center in $O(n^6\alpha(n))$ time. With the help of an "extended corridor structure" of $\mathcal{P}$ [9, 10, 11, 12], we reduce the $O(n^2)$ complexity of our decomposition to another "coarser" decomposition of $O(n + h^2)$ complexity; with another crucial observation (Lemma 7), one may compute the diameter in $O(n^3 + h^4)$ time by using our techniques for the above $O(n^4)$ time algorithm. One of our main contributions is an additional series of observations (Lemmas 9 to 18) that allow us to further reduce the running time to $O(n^2 + h^4)$. These observations along with the decomposition may also have other applications. The idea for computing the center is similar.

We are motivated to study the L_1 versions of the diameter and center problems in polygonal (even non-rectilinear) domains for several reasons. First, the L_1 metric is natural and well studied in optimization and routing problems, as it models actual costs in rectilinear road networks and certain robotics/VLSI applications. Indeed, the L_1 diameter and center problems in the simpler setting of simply connected domains have been studied [6, 22]. Second, the L_1 metric approximates the Euclidean metric. Further, improved understanding of algorithmic results in one metric can assist in understanding in other metrics.

1.1 Preliminaries

For any subset $A \subset \mathbb{R}^2$, denote by ∂A the boundary of A. Denote by $\overline{pq}$ the line segment with endpoints p and q. For any path $\pi \in \mathbb{R}^2$, let $|\pi|$ be the L_1 *length* of π. A path is *xy-monotone* (or *monotone* for short) if every vertical or horizontal line intersects it in at most one connected component.

▶ **Fact 1.** *For any monotone path π between two points $p, q \in \mathbb{R}^2$, $|\pi| = |\overline{pq}|$ holds.*

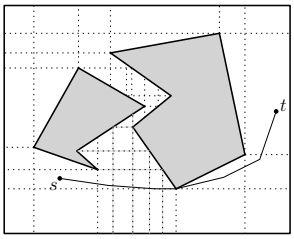


Figure 1 The cell decomposition $\mathcal{D}$ of $\mathcal{P}$, and a shortest path from s to t.

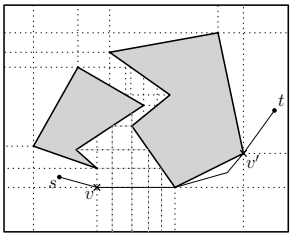


Figure 2 Illustrating Lemma 1: a shortest path through vertices $v \in V_\sigma$ and $v' \in V_{\sigma'}$.

We view the boundary $\partial\mathcal{P}$ of $\mathcal{P}$ as a series of *obstacles* so that no path in $\mathcal{P}$ is allowed to cross $\partial\mathcal{P}$. Throughout the paper, unless otherwise stated, a shortest path always refers to an L_1 shortest path and the distance/length of a path (e.g., $d(p, q)$) always refers to its L_1 distance/length. The diameter/center always refers to the L_1 geodesic diameter/center.

▶ **Fact 2** ([14, 15]). *In any simple polygon P, there is a unique Euclidean shortest path π between any two points in P. The path π is also an L_1 shortest path in P.*

The rest of the paper is organized as follows. In Section 2, we introduce our cell decomposition of $\mathcal{P}$ and exploit it to have preliminary algorithms for computing the diameter and center of $\mathcal{P}$. The algorithms will be improved later in Section 4, based on the extended corridor structure and new observations discussed in Section 3.

Due to the space limit, most lemma and theorem proofs are omitted but can be found in the full version of the paper [3].

2 The Cell Decomposition and Preliminary Algorithms

We first build the *horizontal trapezoidal map* by extending a horizontal line from each vertex of $\mathcal{P}$ until each end of the line hits $\partial\mathcal{P}$. Next, we compute the *vertical trapezoidal map* by extending a vertical line from each vertex of $\mathcal{P}$ *and* each of the ends of the above extended lines. We then overlay the two trapezoidal maps, resulting in a *cell decomposition* $\mathcal{D}$ of $\mathcal{P}$ (e.g., see Fig. 1). The above extended horizontal or vertical line segments are called the *diagonals* of $\mathcal{D}$. Note that $\mathcal{D}$ has $O(n)$ diagonals and $O(n^2)$ cells. Each cell σ of $\mathcal{D}$ appears as a trapezoid or a triangle; let V_σ be the set of vertices of $\mathcal{D}$ that are incident to σ (note that $|V_\sigma| \leq 4$). We let $\mathcal{D}$ also denote the set of all the cells of the decomposition.

Each cell of $\mathcal{D}$ is an intersection between a trapezoid of the horizontal trapezoidal map and another one of the vertical trapezoidal map. Two cells of $\mathcal{D}$ are *aligned* if they are contained in the same trapezoid of the horizontal or vertical trapezoidal map, and *unaligned* otherwise. Lemma 1 is crucial for computing both the diameter and the center of $\mathcal{P}$.

▶ **Lemma 1.** *Let σ, σ' be any two cells of $\mathcal{D}$. For any point $s \in \sigma$ and any point $t \in \sigma'$, if σ and σ' are aligned, then $d(s, t) = |\overline{st}|$; otherwise, there exists an L_1 shortest path between s and t that passes through two vertices $v \in V_\sigma$ and $v' \in V_{\sigma'}$ (e.g., see Fig. 2).*

2.1 Computing the Geodesic Diameter

The general idea is to consider every pair of cells of $\mathcal{D}$ separately. For each pair of such cells $\sigma, \sigma' \in \mathcal{D}$, we compute the maximum geodesic distance between σ and σ', that is, $\max_{s\in\sigma, t\in\sigma'} d(s, t)$, called the (σ, σ')-*constrained diameter*. Since $\mathcal{D}$ is a decomposition of $\mathcal{P}$,

the diameter of $\mathcal{P}$ is equal to the maximum value of the constrained diameters over all pairs of cells of $\mathcal{D}$. We handle two cases depending on whether σ and σ' are aligned.

If σ and σ' are aligned, by Lemma 1, for any $s \in \sigma$ and $t \in \sigma'$, we have $d(s,t) = |\overline{st}|$, i.e, the L_1 distance of $\overline{st}$. Since the L_1 distance function is convex, the (σ,σ')-constrained diameter is always realized by some pair (v,v') of two vertices with $v \in V_\sigma$ and $v' \in V'_\sigma$. We are thus done by checking at most 16 pairs of vertices, in $O(1)$ time.

In the following, we assume that σ and σ' are unaligned. Consider any point $s \in \sigma$ and any point $t \in \sigma'$. For any vertex $v \in V_\sigma$ and any vertex $v' \in V_{\sigma'}$, consider the path from s to t obtained by concatenating $\overline{sv}$, a shortest path from v to v', and $\overline{v't}$, and let $d_{vv'}(s,t)$ be its length. Lemma 1 ensures that $d(s,t) = \min_{v\in V_\sigma, v'\in V_{\sigma'}} d_{vv'}(s,t)$. Since $d_{vv'}(s,t) = |\overline{sv}| + |\overline{v't}| + d(v,v')$ and $d(v,v')$ is constant over all $(s,t) \in \sigma \times \sigma'$, the function $d_{vv'}$ is linear on $\sigma \times \sigma'$. Thus, it is easy to compute the (σ,σ')-constrained diameter once we know the value of $d(v,v')$ for every pair (v,v') of vertices.

▶ **Lemma 2.** *For any two cells $\sigma,\sigma' \in \mathcal{D}$, the (σ,σ')-constrained diameter can be computed in constant time, provided that $d(v,v')$ for every pair (v,v') with $v \in V_\sigma$ and $v' \in V_{\sigma'}$ has been computed.*

For each vertex v of $\mathcal{D}$, an easy way can compute $d(v,v')$ for all other vertices v' of $\mathcal{D}$ in $O(n^2 \log n)$ time, by first computing the shortest path map $SPM(v)$ [19, 20] in $O(n \log n)$ time and then computing $d(v,v')$ for all $v' \in \mathcal{D}$ in $O(n^2 \log n)$ time. We instead have a faster algorithm in Lemma 3, due to that all vertices on every diagonal of $\mathcal{D}$ are sorted.

▶ **Lemma 3.** *For each vertex v of $\mathcal{D}$, we can evaluate $d(v,v')$ for all vertices v' of $\mathcal{D}$ in $O(n^2)$ time.*

Thus, after $O(n^4)$-time preprocessing, for any two cells $\sigma,\sigma' \in \mathcal{D}$, the (σ,σ')-constrained diameter can be computed in $O(1)$ time by Lemma 2. Since $\mathcal{D}$ has $O(n^2)$ cells, it suffices to handle at most $O(n^4)$ pairs of cells, resulting in $O(n^4)$ candidates for the diameter, and the maximum is the diameter.

▶ **Theorem 4.** *The L_1 geodesic diameter of $\mathcal{P}$ can be computed in $O(n^4)$ time.*

2.2 Computing the Geodesic Center

For any point $q \in \mathcal{P}$, we define $R(q)$ to be the maximum geodesic distance between q and any point in $\mathcal{P}$, i.e., $R(q) := \max_{p\in\mathcal{P}} d(p,q)$. A center q^* of $\mathcal{P}$ is defined to be a point with $R(q^*) = \min_{q\in\mathcal{P}} R(q)$. Our approach is again based on the decomposition $\mathcal{D}$: for each cell $\sigma \in \mathcal{D}$, we want to find a point $q \in \sigma$ that minimizes the maximum geodesic distance $d(p,q)$ over all $p \in \mathcal{P}$. We call such a point $q \in \sigma$ a *σ-constrained center*. Thus, if q' is a σ-constrained center, then we have $R(q') = \min_{q\in\sigma} R(q)$. Clearly, the center q^* of $\mathcal{P}$ must be a σ-constrained center for some $\sigma \in \mathcal{D}$. Our algorithm thus finds a σ-constrained center for every $\sigma \in \mathcal{D}$, which at last results in $O(n^2)$ candidates for a center of $\mathcal{P}$.

Consider any cell $\sigma \in \mathcal{D}$. To compute a σ-constrained center, we investigate the function R restricted to σ and exploit Lemma 1 again. To utilize Lemma 1, we define $R_{\sigma'}(q) := \max_{p\in\sigma'} d(p,q)$ for any $\sigma' \in \mathcal{D}$. For any $q \in \sigma$, $R(q) = \max_{\sigma'\in\mathcal{D}} R_{\sigma'}(q)$, that is, R is the upper envelope of all the $R_{\sigma'}$ on the domain σ. Our algorithm explicitly computes the functions $R_{\sigma'}$ for all $\sigma' \in \mathcal{D}$ and computes the upper envelope $\mathcal{U}$ of the graphs of the $R_{\sigma'}$. Then, a σ-constrained center corresponds to a lowest point on $\mathcal{U}$.

▶ **Lemma 5.** *The function $R_{\sigma'}$ is piecewise linear on σ and has $O(1)$ complexity.*

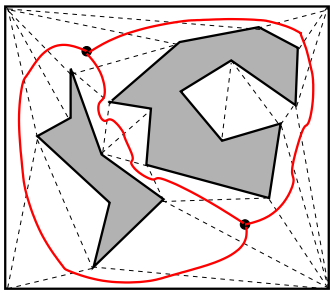


Figure 3 A triangulation $\mathcal{T}$ of $\mathcal{P}$ and the 3-regular graph obtained from the dual graph of $\mathcal{T}$ whose nodes and edges are depicted by black dots and red solid curves.

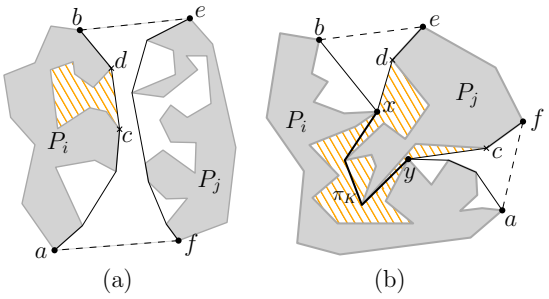


Figure 4 Hourglasses H_K in corridors K. (a) H_K is open. Five bays can be seen. A bay with gate $\overline{cd}$ is shown as the shaded region. (b) H_K is closed. There are three bays and a canal, and the shaded region depicts the canal with two gates $\overline{dx}$ and $\overline{cy}$.

To compute a σ-constrained center, we first handle every cell $\sigma' \in \mathcal{D}$ to compute the graph of $R_{\sigma'}$ and thus gather its linear patches. Let Γ be the family of those linear patches for all $\sigma' \in \mathcal{D}$. We then compute the upper envelope of Γ and find a lowest point on the upper envelope, which corresponds to a σ-constrained center. Since $|\Gamma| = O(n^2)$ by Lemma 5, the upper envelope can be computed in $O(n^4\alpha(n))$ time by executing the algorithm by Edelsbrunner et al. [13], where $\alpha(\cdot)$ denotes the inverse Ackermann function.

▶ **Theorem 6.** *An L_1 geodesic center of $\mathcal{P}$ can be computed in $O(n^6\alpha(n))$ time.*

3 Exploiting the Extended Corridor Structure

In this section, we briefly review the extended corridor structure of $\mathcal{P}$ and present new observations, which will be crucial for our improved algorithms in Section 4. The corridor structure has been used for solving shortest path problems [9, 17, 18]. Later some new concepts such as "bays," "canals," and the "ocean" were introduced [10, 11], referred to as the "extended corridor structure."

3.1 The Extended Corridor Structure

Let $\mathcal{T}$ denote an arbitrary triangulation of $\mathcal{P}$ (e.g., see Figure 3). We can obtain $\mathcal{T}$ in $O(n \log n)$ time or $O(n + h\log^{1+\epsilon} h)$ time for any $\epsilon > 0$ [7]. Based on the dual graph of $\mathcal{T}$, one can obtain a planar 3-regular graph, possibly with loops and multi-edges, by repeatedly removing all degree-one nodes and then contracting all degree-two nodes. The resulting 3-regular graph has $O(h)$ faces, nodes, and edges [18]. Each node of the graph corresponds to a triangle in $\mathcal{T}$, called a *junction triangle*. The removal of all junction triangles from $\mathcal{P}$ results in $O(h)$ components, called *corridors*, each of which corresponds to an edge of the graph. See Figure 3. Refer to [18] for more details.

Let $P_1, \ldots, P_h$ be the h holes of $\mathcal{P}$ and P_0 be the outer polygon of $\mathcal{P}$. For simplicity, a hole may also refer to the unbounded region outside P_0 hereafter. The boundary ∂K of a corridor K consists of two diagonals of $\mathcal{T}$ and two paths along the boundary of holes P_i and P_j, respectively (it is possible that P_i and P_j are the same hole, in which case one may consider P_i and P_j as the above two paths respectively). Let $a, b \in P_i$ and $e, f \in P_j$ be the endpoints of the two paths, respectively, such that $\overline{be}$ and $\overline{fa}$ are diagonals of $\mathcal{T}$, each of which bounds a junction triangle. See Figure 4. Let π_{ab} (resp., π_{ef}) denote the *Euclidean*

shortest path from a to b (resp., e to f) inside K. The region H_K bounded by π_{ab}, π_{ef}, $\overline{be}$, and $\overline{fa}$ is called an *hourglass*, which is either *open* if $\pi_{ab} \cap \pi_{ef} = \emptyset$, or *closed*, otherwise. If H_K is open, then both π_{ab} and π_{ef} are convex chains and are called the *sides* of H_K; otherwise, H_K consists of two "funnels" and a path $\pi_K = \pi_{ab} \cap \pi_{ef}$ joining the two apices of the two funnels, called the *corridor path* of K. The two funnel apices (e.g., x and y in Figure 4(b)) connected by π_K are called the *corridor path terminals*. Note that each funnel comprises two convex chains.

We consider the region of K minus the interior of H_K, which consists of a number of simple polygons facing (i.e., sharing an edge with) one or both of P_i and P_j. We call each of these simple polygons a *bay* if it is facing a single hole, or a *canal* if it is facing both holes. Each bay is bounded by a portion of the boundary of a hole and a segment $\overline{cd}$ between two obstacle vertices c, d that are consecutive along a side of H_K. We call the segment $\overline{cd}$ the *gate* of the bay. (See Figure 4(a).) On the other hand, there exists a unique canal for each corridor K only when H_K is closed and the two holes P_i and P_j both bound the canal. The canal in K in this case completely contains the corridor path π_K. A canal has two *gates* $\overline{xd}$ and $\overline{yc}$ that are two segments facing the two funnels, respectively, where x, y are the corridor path terminals and d, c are vertices of the funnels. (See Figure 4(b).)

Let $\mathcal{M} \subseteq \mathcal{P}$ be the union of all junction triangles, open hourglasses, and funnels. We call $\mathcal{M}$ the *ocean*. Its boundary $\partial\mathcal{M}$ consists of $O(h)$ convex vertices and $O(h)$ reflex chains each of which is a side of an open hourglass or of a funnel. Note that each bay or canal is a simple polygon and $\mathcal{P} \setminus \mathcal{M}$ consists of all bays and canals of $\mathcal{P}$.

For convenience of discussion, we define each bay/canal in such a way that they do not contain their gates and hence their gates are contained in $\mathcal{M}$; therefore, each point of $\mathcal{P}$ is either in a bay/canal or in $\mathcal{M}$, but not in both. The following lemma is one of our key observations for our improved algorithms in Section 4.

▶ **Lemma 7.** *Let $s \in \mathcal{P}$ be any point and A be a bay or canal of $\mathcal{P}$. Then, for any $t \in A$, there exists $t' \in \partial A$ such that $d(s,t) \leq d(s,t')$. Equivalently,* $\max_{t \in A} d(s,t) = \max_{t \in \partial A} d(s,t)$.

3.2 Shortest Paths in the Ocean $\mathcal{M}$

We now discuss shortest paths in $\mathcal{M}$. Recall that corridor paths are contained in canals, but their terminals are on $\partial\mathcal{M}$. By using the corridor paths and $\mathcal{M}$, finding an L_1 or Euclidean shortest path between two points s and t in $\mathcal{M}$ can be reduced to the convex case since $\partial\mathcal{M}$ consists of $O(h)$ convex chains. For example, suppose both s and t are in $\mathcal{M}$. Then, there must be a shortest s-t path π that lies in the union of $\mathcal{M}$ and all corridor paths [9, 11, 18].

Consider any two points s and t in $\mathcal{M}$. A shortest s-t path $\pi(s,t)$ in $\mathcal{P}$ is a shortest path in $\mathcal{M}$ that possibly contains some corridor paths. Intuitively, one may view corridor paths as "shortcuts" among the components of the space $\mathcal{M}$. As in [18], since $\partial\mathcal{M}$ consists of $O(h)$ convex vertices and $O(h)$ reflex chains, the complementary region $\mathcal{P}' \setminus \mathcal{M}$ (where $\mathcal{P}'$ refers to the union of $\mathcal{P}$ and all its holes) can be partitioned into a set $\mathcal{B}$ of $O(h)$ convex objects with a total of $O(n)$ vertices (e.g., by extending an angle-bisecting segment inward from each convex vertex [18]). If we view the objects in $\mathcal{B}$ as obstacles, then π is a shortest path avoiding all obstacles of $\mathcal{B}$ but possibly containing some corridor paths. Note that our algorithms can work on $\mathcal{P}$ and $\mathcal{M}$ directly without using $\mathcal{B}$; but for ease of exposition, we will discuss our algorithm with the help of $\mathcal{B}$.

Each convex obstacle P of $\mathcal{B}$ has at most four *extreme vertices*: the topmost, bottommost, leftmost, and rightmost vertices, and there may be some corridor path terminals on the boundary of P. We connect the extreme vertices and the corridor path terminals on ∂P

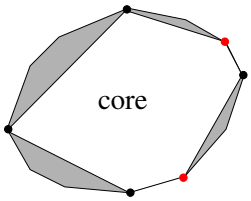


Figure 5 Illustrating the core of a convex obstacle: the red points are corridor path terminals.

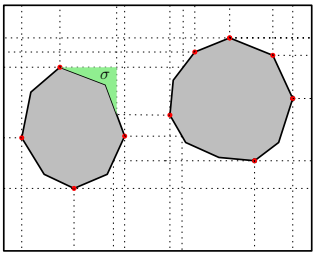


Figure 6 Illustrating the core-based cell decomposition $\mathcal{D}_{\mathcal{M}}$: the red vertices are core vertices and the green cell σ is a boundary cell.

consecutively by line segments to obtain another polygon, denoted by $core(P)$ and called the *core* of P (see Figure 5). Let $\mathcal{P}_{core}$ denote the complement of the union of all cores $core(P)$ for all $P \in \mathcal{B}$ and corridor paths in $\mathcal{P}$. Note that the number of vertices of $\mathcal{P}_{core}$ is $O(h)$ and $\mathcal{M} \subseteq \mathcal{P}_{core}$. For $s, t \in \mathcal{P}_{core}$, let $d_{core}(s,t)$ be the geodesic distance between s and t in $\mathcal{P}_{core}$.

The core structure leads to a more efficient way to find an L_1 shortest path between two points in $\mathcal{P}$. Chen and Wang [9] proved that an L_1 shortest path between $s, t \in \mathcal{M}$ in $\mathcal{P}_{core}$ can be locally modified to an L_1 shortest path in $\mathcal{P}$ without increasing its L_1 length.

▶ **Lemma 8** ([9]). *For any two points s and t in $\mathcal{M}$, $d(s,t) = d_{core}(s,t)$ holds.*

Hence, to compute $d(s,t)$ between two points s and t in $\mathcal{M}$, it is sufficient to consider only the cores and the corridor paths, that is, $\mathcal{P}_{core}$. We thus reduce the problem size from $O(n)$ to $O(h)$. Let $SPM_{core}(s)$ be a shortest path map for any source point $s \in \mathcal{M}$. Then, $SPM_{core}(s)$ has $O(h)$ complexity and can be computed in $O(h \log h)$ time [9].

We introduce a core-based cell decomposition $\mathcal{D}_{\mathcal{M}}$ of the ocean $\mathcal{M}$ (see Figure 6) in order to fully exploit the advantage of the core structure in designing algorithms computing the L_1 geodesic diameter and center. For any $P \in \mathcal{B}$, the vertices of $core(P)$ are called *core vertices*.

The construction of $\mathcal{D}_{\mathcal{M}}$ is analogous to that of $\mathcal{D}$ for $\mathcal{P}$. We first extend a horizontal line only from each *core vertex* until it hits $\partial\mathcal{M}$ to have a horizontal diagonal, and then extend a vertical line from each core vertex and each endpoint of the above horizontal diagonal. The resulting cell decomposition induced by the above diagonals is $\mathcal{D}_{\mathcal{M}}$. Hence, $\mathcal{D}_{\mathcal{M}}$ is constructed in $\mathcal{M}$ with respect to core vertices. Note that $\mathcal{D}_{\mathcal{M}}$ consists of $O(h^2)$ cells and can be built in $O(n \log n + h^2)$ time by a typical plane sweep algorithm. We call a cell σ of $\mathcal{D}_{\mathcal{M}}$ a *boundary cell* if $\partial\sigma \cap \partial\mathcal{M} \neq \emptyset$. For any boundary cell σ, the portion $\partial\sigma \cap \partial\mathcal{M}$ appears as a convex chain of $P \in \mathcal{B}$ by our construction of its core and $\mathcal{D}_{\mathcal{M}}$; since $\partial\sigma \cap \partial\mathcal{M}$ may contain multiple vertices of $\mathcal{M}$, the complexity of σ may not be constant. Any non-boundary cell of $\mathcal{D}_{\mathcal{M}}$ is a rectangle bounded by four diagonals. Each vertex of $\mathcal{D}_{\mathcal{M}}$ is either an endpoint of its diagonal or an intersection of two diagonals; thus, the number of vertices of $\mathcal{D}_{\mathcal{M}}$ is $O(h^2)$. Below we prove an analogue of Lemma 1 for the decomposition $\mathcal{D}_{\mathcal{M}}$ of $\mathcal{M}$. Let V_σ be the set of vertices of $\mathcal{D}_{\mathcal{M}}$ incident to σ. Note that $|V_\sigma| \leq 4$. We define the alignedness relation between two cells of $\mathcal{D}_{\mathcal{M}}$ analogously to that for $\mathcal{D}$. We then observe an analogy to Lemma 1.

▶ **Lemma 9.** *Let σ, σ' be any two cells of $\mathcal{D}_{\mathcal{M}}$. If they are aligned, then $d(s,t) = |\overline{st}|$ for any $s \in \sigma$ and $t \in \sigma'$; otherwise, there exists a shortest s-t path in $\mathcal{P}$ containing two vertices $v \in V_\sigma$ and $v' \in V_{\sigma'}$ with $d(s,t) = |\overline{sv}| + d(v,v') + |\overline{v't}|$.*

4 Improved Algorithms

In this section, we further explore the geometric structures and give more properties about our decomposition. These results, together with our results in Section 3, help us to give improved algorithms that compute the diameter and center, using a similar algorithmic framework as in Section 2.

4.1 The Cell-to-Cell Geodesic Distance Functions

Recall that our preliminary algorithms in Section 2 rely on the nice behavior of the cell-to-cell geodesic distance function: specifically, d restricted to $\sigma \times \sigma'$ for any two cells $\sigma, \sigma' \in \mathcal{D}$ is the lower envelope of $O(1)$ linear functions. We now have two different cell decompositions, $\mathcal{D}$ of $\mathcal{P}$ and $\mathcal{D}_{\mathcal{M}}$ of $\mathcal{M}$. Here, we observe analogues of Lemmas 1 and 9 for any two cells in $\mathcal{D} \cup \mathcal{D}_{\mathcal{M}}$, by extending the alignedness relation between cells in $\mathcal{D}$ and $\mathcal{D}_{\mathcal{M}}$, as follows.

Consider the geodesic distance function d restricted to $\sigma \times \sigma'$ for any two cells $\sigma, \sigma' \in \mathcal{D} \cup \mathcal{D}_{\mathcal{M}}$. We call a cell $\sigma \in \mathcal{D} \cup \mathcal{D}_{\mathcal{M}}$ *oceanic* if $\sigma \subset \mathcal{M}$, or *coastal*, otherwise. If both $\sigma, \sigma' \in \mathcal{D} \cup \mathcal{D}_{\mathcal{M}}$ are coastal, then $\sigma, \sigma' \in \mathcal{D}$ and the case is well understood as discussed in Section 2. Otherwise, there are two cases: the *ocean-to-ocean case* where both σ and σ' are oceanic, and the *coast-to-ocean case* where only one of them is oceanic.

For the ocean-to-ocean case, we extend the alignedness relation for all oceanic cells in $\mathcal{D} \cup \mathcal{D}_{\mathcal{M}}$. To this end, when both σ and σ' are in $\mathcal{D}$ or $\mathcal{D}_{\mathcal{M}}$, the alignedness has already been defined. For any two oceanic cells $\sigma \in \mathcal{D}$ and $\sigma' \in \mathcal{D}_{\mathcal{M}}$, we define their alignedness relation in the following way. If σ is contained in a cell $\sigma'' \in \mathcal{D}_{\mathcal{M}}$ that is aligned with σ', then we say that σ and σ' are *aligned*. However, σ may not be contained in a cell of $\mathcal{D}_{\mathcal{M}}$ because the endpoints of horizontal diagonals of $\mathcal{D}_{\mathcal{M}}$ that are on bay/canal gates are not vertices of $\mathcal{D}$ and those endpoints create vertical diagonals in $\mathcal{D}_{\mathcal{M}}$ that are not in $\mathcal{D}$. To resolve this issue, we augment $\mathcal{D}$ by adding the vertical diagonals of $\mathcal{D}_{\mathcal{M}}$ to $\mathcal{D}$. Specifically, for each vertical diagonal l of $\mathcal{D}_{\mathcal{M}}$, if no diagonal in $\mathcal{D}$ contains l, then we add l to $\mathcal{D}$ and extend l vertically until it hits the boundary of $\mathcal{P}$. In this way, we add $O(h)$ vertical diagonals to $\mathcal{D}$, and the size of $\mathcal{D}$ is still $O(n^2)$. Further, all results we obtained before are still applicable to the new $\mathcal{D}$. By abusing the notation, we still use $\mathcal{D}$ to denote the new version of $\mathcal{D}$. Now, for any two oceanic cells $\sigma \in \mathcal{D}$ and $\sigma' \in \mathcal{D}_{\mathcal{M}}$, there must be a unique cell $\sigma'' \in \mathcal{D}_{\mathcal{M}}$ that contains σ, and σ and σ' are defined to be *aligned* if and only if σ'' and σ' are aligned. Lemmas 1 and 9 are naturally extended as follows, along with this extended alignedness relation.

▶ **Lemma 10.** *Let $\sigma, \sigma' \in \mathcal{D} \cup \mathcal{D}_{\mathcal{M}}$ be two oceanic cells. For any $s \in \sigma$ and $t \in \sigma'$, it holds that $d(s,t) = |\overline{st}|$ if σ and σ' are aligned; otherwise, there exists a shortest s-t path that passes through a vertex $v \in V_\sigma$ and a vertex $v' \in V_{\sigma'}$.*

We then turn to the coast-to-ocean case. We now focus on a bay or canal A. Since A has gates, we need to somehow incorporate the influence of its gates into the decomposition $\mathcal{D}$. To this end, we add $O(1)$ additional diagonals into $\mathcal{D}_{\mathcal{M}}$ as follows: extend a horizontal line from each endpoint of each gate of A until it hits $\partial\mathcal{M}$, and then extend a vertical line from each endpoint of each gate of A and each endpoint of the horizontal diagonals that are added above. Let $\mathcal{D}^A_{\mathcal{M}}$ denote the resulting decomposition of $\mathcal{M}$. Note that there are some cells of $\mathcal{D}_{\mathcal{M}}$ each of which is partitioned into $O(1)$ cells of $\mathcal{D}^A_{\mathcal{M}}$ but the combinatorial complexity of $\mathcal{D}^A_{\mathcal{M}}$ is still $O(h^2)$. For any gate g of A, let $C_g \subset \mathcal{P}$ be the cross-shaped region of points in $\mathcal{P}$ that can be joined with a point on g by a vertical or horizontal line segment inside $\mathcal{P}$. Since the endpoints of g are also obstacle vertices, the boundary of C_g is formed by four diagonals

of $\mathcal{D}$. Hence, any cell in $\mathcal{D}$ or $\mathcal{D}^A_{\mathcal{M}}$ is either completely contained in C_g or interior-disjoint from C_g. A cell of $\mathcal{D}$ or $\mathcal{D}^A_{\mathcal{M}}$ in the former case is said to be *g-aligned*.

In the following, we let $\sigma \in \mathcal{D}$ be any coastal cell that intersects A and $\sigma' \in \mathcal{D}^A_{\mathcal{M}}$ be any oceanic cell. Depending on whether σ and σ' are g-aligned for a gate g of A, there are three cases: (1) both cells are g-aligned; (2) σ' is not g-aligned; (3) σ' is g-aligned but σ is not. Lemma 11 handles the first case. Lemma 12 deals with a special case for the latter two cases. Lemma 13 is for the second case. Lemma 15 is for the third case and Lemma 14 is for proving Lemma 15. The proof of Lemma 16 summarizes the entire algorithm for all three cases.

▶ **Lemma 11.** *Suppose that σ and σ' are both g-aligned for a gate g of A. Then, for any $s \in \sigma$ and $t \in \sigma'$, we have $d(s,t) = |\overline{st}|$.*

Consider any path π in $\mathcal{P}$ from $s \in \sigma$ to $t \in \sigma'$, and assume π is directed from s to t. For a gate g of A, we call π *g-through* if g is the last gate of A crossed by π. The path π is a *shortest g-through path* if its L_1 length is the smallest among all g-through paths from s to t. Suppose π is a shortest path from s to t in $\mathcal{P}$. Since σ may intersect $\mathcal{M}$, if $s \in \sigma$ is not in A, then π may *avoid* A (i.e., π does not intersect A). If A is a bay, then either π avoids A or π is a shortest g-through path for the only gate g of A; otherwise (i.e., A is a canal), either π avoids A or π is a shortest g-through or g'-through path for the two gates g and g' of A. We have the following lemma, which is self-evident.

▶ **Lemma 12.** *Suppose that for any gate g of A, at least one of σ and σ' is not g-aligned. For any $s \in \sigma$ and $t \in \sigma'$, if there exists a shortest s-t path that avoids A, then a shortest s-t path passes through a vertex $v \in V_\sigma$ and another vertex $v' \in V_{\sigma'}$.*

We then focus on shortest g-through paths according to the g-alignedness of σ and σ'.

▶ **Lemma 13.** *Suppose σ' is not g-aligned for a gate g of A and there are no shortest s-t paths that avoid A. Then, for any $s \in \sigma$ and $t \in \sigma'$, there exists a shortest g-through s-t path containing a vertex $v \in V_\sigma$ and a vertex $v' \in V_{\sigma'}$.*

The remaining case is when $\sigma' \in \mathcal{D}^A_{\mathcal{M}}$ is g-aligned but $\sigma \in \mathcal{D}$ is not. Recall σ is coastal and intersects A, and σ' is oceanic (implying σ' does not intersect A).

▶ **Lemma 14.** *Let g be a gate of A, and suppose that σ is not g-aligned. Then, there exists a unique vertex $v_g \in V_\sigma \cap A$ such that for any $s \in \sigma$ and $x \in g$, the concatenation of segment $\overline{sv_g}$ and any L_1 shortest path from v_g to x inside $A \cup \sigma$ results in an L_1 shortest path from s to x in $A \cup \sigma$.*

From now on, let v_g be the vertex as described in Lemma 14 (v_g can be found efficiently, as shown in the proof of Lemma 16). Consider the union of the Euclidean shortest paths inside A from v_g to all points $x \in g$. Since A is a simple polygon, the union forms a funnel $F_g(v_g)$ with base g, plus the Euclidean shortest path from v_g to the apex of $F_g(v_g)$. Recall Fact 2 that any Euclidean shortest path inside a simple polygon is also an L_1 shortest path. Let $W_g(v_g)$ be the set of horizontally and vertically extreme points in each convex chain of $F_g(v_g)$, that is, $W_g(v_g)$ gathers the leftmost, rightmost, uppermost, and lowermost points in each chain of $F_g(v_g)$. Note that $|W_g(v_g)| \leq 8$ and $W_g(v_g)$ includes the endpoints of g and the apex of $F_g(v_g)$. We then observe the following lemma.

▶ **Lemma 15.** *Suppose that σ' is g-aligned but σ is not. Then, for any $s \in \sigma$ and $t \in \sigma'$, there exists a shortest g-through s-t path that passes through v_g and some $w \in W_g(v_g)$. Moreover, the length of such a path is $|\overline{sv_g}| + d(v_g, w) + |\overline{wt}|$.*

Proof. Since A is a simple polygon, any Euclidean shortest path in A is also an L_1 shortest path by Fact 2. Thus, the L_1 length of a shortest path from v_g to any point x in the funnel $F_g(v_g)$ is equal to the L_1 length of the unique Euclidean shortest path in A, which is contained in $F_g(v_g)$.

By Lemma 14 and the assumption that σ' is g-aligned, among the paths from s to t that cross the gate g, there exists an L_1 shortest g-through s-t path π consisting of three portions: $\overline{sv_g}$, the unique Euclidean shortest path from v_g to a vertex u on a convex chain of $F_g(v_g)$, and $\overline{ut}$. Let $w \in W_g(v_g)$ be the last one among $W_g(v_g)$ that we encounter during the walk from s to t along π. Consider the segment $\overline{wt}$, which may cross $\partial F_g(v_g)$. If $\overline{wt} \cap \partial F_g(v_g) = \emptyset$, then we are done by replacing the subpath of π from u to t by $\overline{wt}$. Otherwise, $\overline{wt}$ crosses $\partial F_g(v_g)$ at two points $p, q \in \partial F_g(v_g)$. Since $W_g(v_g)$ includes all extreme points of each chain of $F_g(v_g)$, there is no $w' \in W_g(v_g)$ on the subchain of $F_g(v_g)$ between p and q. Hence, we can replace the subpath of π from w to t by a monotone path from w to t, which consists of $\overline{wp}$, the convex path from p to q along $\partial F_g(v_g)$, and $\overline{qt}$, and the L_1 length of the above monotone path is equal to $|\overline{wt}|$ by Fact 1. Consequently, the resulting path is also an L_1 shortest path with the desired property. ◀

For any cell $\sigma \in \mathcal{D} \cup \mathcal{D}_\mathcal{M}$, let n_σ be the size of σ. If σ is a boundary cell of $\mathcal{D}_\mathcal{M}$, then n_σ may not be bounded by a constant; otherwise, σ is a trapezoid or a triangle, and thus $n_\sigma \leq 4$. The geodesic distance function d defined on $\sigma \times \sigma'$ for any two cells $\sigma, \sigma' \in \mathcal{D} \cup \mathcal{D}_\mathcal{M}$ can be explicitly computed in $O(n_\sigma n_{\sigma'})$ time after some preprocessing, as shown in Lemma 16.

▶ **Lemma 16.** *Let σ be any cell of $\mathcal{D}$ or $\mathcal{D}_\mathcal{M}$. After $O(n)$-time preprocessing, the function d on $\sigma \times \sigma'$ for any cell $\sigma' \in \mathcal{D} \cup \mathcal{D}_\mathcal{M}$ can be explicitly computed in $O(n_\sigma n_{\sigma'})$ time, provided that $d(v, v')$ has been computed for any $v \in V_\sigma$ and any $v' \in V_{\sigma'}$. Moreover, d on $\sigma \times \sigma'$ is the lower envelope of $O(1)$ linear functions.*

Proof. If both σ and σ' are oceanic, then Lemma 10 implies that for any $(s, t) \in \sigma \times \sigma'$, $d(s, t) = |\overline{st}|$ if they are aligned, or $d(s, t) = \min_{v \in V_\sigma, v' \in V_{\sigma'}} d_{vv'}(s, t)$, where $d_{vv'}(s, t) = |\overline{sv}| + d(v, v') + |\overline{v't}|$. On the other hand, if σ and σ' are coastal, then both are cells of $\mathcal{D}$ and Lemma 1 implies the same conclusion. Since $|V_\sigma| \leq 4$ and $|V_{\sigma'}| \leq 4$ in either case, the geodesic distance d on $\sigma \times \sigma$ is the lower envelope of at most 16 linear functions. Hence, provided that the values of $d(v, v')$ for all pairs (v, v') are known, the envelope can be computed in time proportional to the complexity of the domain $\sigma \times \sigma'$, which is $O(n_\sigma n_{\sigma'})$.

From now on, suppose that σ is coastal and σ' is oceanic. Then, σ is a cell of $\mathcal{D}$ and intersects some bay or canal A. If σ' is also a cell of $\mathcal{D}$, then Lemma 1 implies the lemma, as discussed in Section 2; thus, we assume σ' is a cell of $\mathcal{D}_\mathcal{M}$.

As above, we add diagonals extended from each endpoint of each gate of A to obtain $\mathcal{D}^A_\mathcal{M}$, and specify all g-aligned cells for each gate g of A in $O(n)$ time. In the following, let σ' be an oceanic cell of $\mathcal{D}$ or of $\mathcal{D}^A_\mathcal{M}$. Note that a cell of $\mathcal{D}_\mathcal{M}$ can be partitioned into $O(1)$ cells of $\mathcal{D}^A_\mathcal{M}$. We have two cases whether A is a bay or a canal.

First, suppose that A is a bay; let g be the unique gate of A. In this case, any L_1 shortest path is g-through, provided that it intersects A, since g is unique. There are two subcases depending on whether σ is g-aligned or not.

- If σ is g-aligned, then by Lemmas 11, 12, and 13, we have $d(s, t) = |\overline{st}|$ if σ' is g-aligned, or $d(s, t) = \min_{v \in V_\sigma, v' \in V_{\sigma'}} d_{vv'}(s, t)$, otherwise, where $d_{vv'}(s, t) = |\overline{sv}| + d(v, v') + |\overline{v't}|$. Thus, the lemma follows by an identical argument as above.
- Suppose that σ is not g-aligned. Then, $\sigma \subset A$ since A has a unique gate g. In this case, we need to find the vertex $v_g \in V_\sigma$. For the purpose, we compute at most four Euclidean shortest path maps $SPM_A(v)$ inside A for all $v \in V_\sigma$ in $O(n)$ time [14]. By

Fact 2, $SPM_A(v)$ is also an L_1 shortest path map in A. We then specify the L_1 geodesic distance from v to all points on g, which results in a piecewise linear function f_v on g. For each $v \in V_\sigma$, we test whether it holds that $f_v(x) + |\overline{vv'}| \leq f_{v'}(x)$ for all $x \in g$ and all $v' \in V_\sigma$. By Lemma 14, there exists a vertex in V_σ for which the above test is passed, and such a vertex is v_g. Since each shortest path map $SPM_A(v)$ is of $O(n)$ complexity, all the above effort to find v_g is bounded by $O(n)$. Next, we compute the funnel $F_g(v_g)$ and the extreme vertices $W_g(v_g)$ as done above by exploring $SPM_A(v_g)$ in $O(n)$ time.

If σ' is not g-aligned, we apply Lemma 13 to obtain $d(s,t) = \min_{v \in V_\sigma, v' \in V_{\sigma'}} d_{vv'}(s,t)$. Thus, d is the lower envelope of at most 16 linear functions over $\sigma \times \sigma'$. Otherwise, if σ' is g-aligned, then we have $d(s,t) = \min_{w \in W_g(v_g)} d_{v_g w}(s,t)$ by Lemma 15. Since $|W_g(v_g)| \leq 8$, d is the lower envelope of a constant number of linear functions.

Thus, in any case, we conclude the bay case.

Now, suppose that A is a canal. Then, A has two gates g and g', and σ falls into one of the three case: (i) σ is both g-aligned and g'-aligned, (ii) σ is neither g-aligned nor g'-aligned, or (iii) σ is g- or g'-aligned but not both. As a preprocessing, if σ is not g-aligned, then we compute v_g, $F_g(v_g)$, and $W_g(v_g)$ as done in the bay case; analogously, if not g'-aligned, compute $v_{g'}$, $F_{g'}(v_{g'})$, and $W_{g'}(v_{g'})$. Note that any shortest path in $\mathcal{P}$ is either g-through or g'-through, provided that it intersects A. Thus, $d(s,t)$ chooses the minimum among a shortest g-through path, a shortest g'-through path, and a shortest path avoiding A if possible. We consider each of the three cases of σ.

1. Suppose that σ is both g-aligned and g'-aligned. In this case, if σ' is either g-aligned or g'-aligned, then we have $d(s,t) = |\overline{st}|$ by Lemma 11. Otherwise, if σ' is neither g-aligned nor g'-aligned, then we apply Lemmas 12 and 13 to have $d(s,t) = \min_{v \in V_\sigma, v' \in V_{\sigma'}} d_{vv'}(s,t)$. Hence, the lemma follows.
2. Suppose that σ is neither g-aligned nor g'-aligned. If σ' is both g-aligned and g'-aligned, then by Lemma 15 the length of a shortest g-through path is equal to $\min_{w \in W_g(v_g)} d_{v_g w}(s,t)$ while the length of a shortest g'-through path is equal to $\min_{w \in W_{g'}(v_{g'})} d_{v_{g'} w}(s,t)$. The geodesic distance $d(s,t)$ is the minimum of the above two quantities, and thus the lower envelope of $O(1)$ linear function on $\sigma \times \sigma'$.

   If σ' is g-aligned but not g'-aligned, then by Lemmas 13 and 15, we have

$$d(s,t) = \min\{\min_{w \in W_g(v_g)} d_{v_g w}(s,t), \min_{v \in V_\sigma, v' \in V_{\sigma'}} d_{vv'}(s,t)\}.$$

   The case where σ' is g'-aligned but not g-aligned is analogous.

   If σ' is neither g-aligned nor g'-aligned, then $d(s,t) = \min_{v \in V_\sigma, v' \in V_{\sigma'}} d_{vv'}(s,t)$ by Lemma 13.
3. Suppose that σ is g'-aligned but not g-aligned. The other case where it is g-aligned but not g'-aligned can be handled symmetrically. If σ' is g'-aligned, then we have $d(s,t) = |\overline{st}|$ by Lemma 11. If σ' is neither g-aligned nor g'-aligned, then, by Lemmas 12 and 13, $d(s,t) = \min_{v \in V_\sigma, v' \in V_{\sigma'}} d_{vv'}(s,t)$.

   The remaining case is when σ' is g-aligned but not g'-aligned. In this case, the length of a shortest g-through path is equal to $\min_{w \in W_g(v_g)} d_{v_g w}(s,t)$ by Lemma 15 for gate g while the length of a shortest g'-through path is equal to $\min_{v \in V_\sigma, v' \in V_{\sigma'}} d_{vv'}(s,t)$ by Lemmas 12 and 13. Thus, the geodesic distance $d(s,t)$ is the smaller of the two quantities.

Consequently, we have verified every case of (σ, σ'). Finally, observe that it is sufficient to handle separately all the cells $\sigma' \in \mathcal{D}^A_{\mathcal{M}}$ whose union forms the original cell of $\mathcal{D}_{\mathcal{M}}$, since every cell of $\mathcal{D}_{\mathcal{M}}$ can be decomposed into $O(1)$ cells of $\mathcal{D}^A_{\mathcal{M}}$. ◀

4.2 Computing the Geodesic Diameter and Center

Lemma 7 assures that we can ignore coastal cells that are contained in the interior of a bay or canal, in order to find a farthest point from any $s \in \mathcal{P}$. This suggests a combined set $\mathcal{D}_f$ of cells from the two different decompositions $\mathcal{D}$ and $\mathcal{D}_\mathcal{M}$: Let $\mathcal{D}_f$ be the set of all cells σ such that either σ belongs to $\mathcal{D}_\mathcal{M}$ or $\sigma \in \mathcal{D}$ is a coastal cell with $\partial\sigma \cap \partial\mathcal{P} \neq \emptyset$. Note that $\mathcal{D}_f$ consists of $O(h^2)$ oceanic cells from $\mathcal{D}_\mathcal{M}$ and $O(n)$ coastal cells from $\mathcal{D}$. Since the boundary ∂A of any bay or canal A is covered by the cells of $\mathcal{D}_f$, Lemma 7 implies the following lemma.

▶ **Lemma 17.** *For any point $s \in \mathcal{P}$, $\max_{t\in\mathcal{P}} d(s,t) = \max_{\sigma'\in\mathcal{D}_f} \max_{t\in\sigma'} d(s,t)$.*

We apply the same approach as in Section 2 but we use $\mathcal{D}_f$ instead of $\mathcal{D}$.

To compute the diameter, we compute the (σ,σ')-constrained diameter for each pair of cells $\sigma,\sigma' \in \mathcal{D}_f$. Suppose we know the value of $d(v,v')$ for any $v \in V_\sigma$ and any $v \in V_{\sigma'}$ over all $\sigma,\sigma' \in \mathcal{D}_f$. Our algorithm handles each pair (σ,σ') of cells in $\mathcal{D}_f$ according to their types by applying Lemma 16. Lemma 18 computes $d(v,v')$ for all cell vertices v and v' of $\mathcal{D}_f$.

▶ **Lemma 18.** *In $O(n^2+h^4)$ time, one can compute the geodesic distances $d(v,v')$ between every $v \in V_\sigma$ and $v' \in V_{\sigma'}$ for all pairs of two cells $\sigma,\sigma' \in \mathcal{D}_f$.*

Our algorithms for computing the diameter and center are summarized in Theorem 19.

▶ **Theorem 19.** *The L_1 geodesic diameter and center of $\mathcal{P}$ can be computed in $O(n^2+h^4)$ and $O((n^4+n^2h^4)\alpha(n))$ time, respectively.*

Proof. We first discuss the diameter algorithm, whose correctness follows from Lemma 17.

After the execution of the procedure of Lemma 18 as a preprocessing, our algorithm considers three cases for two cells $\sigma,\sigma' \in \mathcal{D}_f$: (i) both are oceanic, (ii) both are coastal, or (iii) σ is coastal and σ' is oceanic. In either case, we apply Lemma 16.

For case (i), we have $O(h^2)$ oceanic cells and the total complexity is $\sum_{\sigma\in\mathcal{D}_\mathcal{M}} n_\sigma = O(n+h^2)$. Thus, the total time for case (i) is bounded by

$$\sum_{\sigma\in\mathcal{D}_\mathcal{M}} \sum_{\sigma'\in\mathcal{D}_\mathcal{M}} O(n_\sigma n_{\sigma'}) = \sum_{\sigma\in\mathcal{D}_\mathcal{M}} O(n_\sigma(n+h^2)) = O((n+h^2)^2) = O(n^2+h^4).$$

For case (ii), we have $O(n)$ coastal cells in $\mathcal{D}_\mathcal{M}$ and their total complexity is $O(n)$ since they are all trapezoidal. Thus, the total time for case (ii) is bounded by $O(n^2)$.

For case (iii), we fix a coastal cell $\sigma \in \mathcal{D}_f$ and iterates over all oceanic cells $\sigma' \in \mathcal{D}_\mathcal{M}$, after an $O(n)$-time preprocessing, as done in the proof of Lemma 16. For each σ, we take $O(n+h^2)$ time since $\sum_{\sigma\in\mathcal{D}_\mathcal{M}} n_\sigma = O(n+h^2)$. Thus, the total time for case (iii) is bounded by $O(n^2+nh^2) = O((n+h^2)^2) = O(n^2+h^4)$.

Next, we discuss our algorithm for computing a geodesic center of $\mathcal{P}$. We consider $O(n^2)$ cells $\sigma \in \mathcal{D}$ and compute all the σ-constrained centers. As a preprocessing, we spend $O(n^4)$ time to compute the geodesic distances $d(v,v')$ for all pairs of vertices of $\mathcal{D}$ by Lemma 3. Fix a cell $\sigma \in \mathcal{D}$. For all $\sigma' \in \mathcal{D}_f$, we compute the geodesic distance function d restricted to $\sigma\times\sigma'$ by applying Lemma 16. As in Section 2, compute the graph of $R_{\sigma'}(q) = \max_{p\in\sigma'} d(p,q)$ by projecting the graph of d over $\sigma\times\sigma'$, and take the upper envelope of the graphs of $R_{\sigma'}$ for all $\sigma' \in \mathcal{D}_f$. By Lemma 16, we have an analogue of Lemma 5 and thus a σ-constrained center can be computed in $O(m^2\alpha(m))$ time, where m denotes the total complexity of all $R_{\sigma'}$. Lemma 16 implies that $m = O(n+h^2)$.

For the time complexity, note that $\sum_{\sigma\in\mathcal{D}_\mathcal{M}} n_\sigma = O(n+h^2)$ and $\sum_{\sigma\in\mathcal{D}_f\setminus\mathcal{D}_\mathcal{M}} n_\sigma = O(n)$. Since each cell in $\mathcal{D}$ is either a triangle or a trapezoid, its complexity is $O(1)$. Thus, for each

$\sigma \in \mathcal{D}$, by Lemma 16, computing a σ-constrained center takes $O((n+h^2)^2\alpha(n))$ time, after an $O(n^4)$-time preprocessing (Lemma 3). Iterating over all $\sigma \in \mathcal{D}$ takes $O(n^2(n+h^2)^2\alpha(n)) = O((n^4+n^2h^4)\alpha(n))$ time. ◀

Acknowledgements. Work by S. W. Bae was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (2013R1A1A1A05006927), and by the Ministry of Education (2015R1D1A1A01057220). M. Korman was supported in part by the ELC project (MEXT KAKENHI No. 24106008). J. Mitchell acknowledges support from the US-Israel Binational Science Foundation (grant 2010074) and the National Science Foundation (CCF-1018388, CCF-1526406). Y. Okamoto is partially supported by Grant-in-Aid for Scientific Research (KAKENHI) 24106005, 24700008, 24220003, 15K00009. V. Polishchuk is supported in part by Grant 2014-03476 from the Sweden's innovation agency VINNOVA. H. Wang was supported in part by the National Science Foundation (CCF-1317143).

References

1 H.-K. Ahn, L. Barba, P. Bose, J.-L. De Carufel, M. Korman, and E. Oh. A linear-time algorithm for the geodesic center of a simple polygon. In *Proc. of the 31st Symposium on Computational Geometry (SoCG)*, pages 209–223, 2015.

2 T. Asano and G. Toussaint. Computing the geodesic center of a simple polygon. Technical Report SOCS-85.32, McGill University, Montreal, Canada, 1985.

3 S.W. Bae, M. Korman, J.S.B. Mitchell, Y. Okamoto, V. Polishchuk, and H. Wang. Computing the L_1 geodesic diameter and center of a polygonal domain. arXiv:1512.07160, 2015.

4 S.W. Bae, M. Korman, and Y. Okamoto. The geodesic diameter of polygonal domains. *Discrete and Computational Geometry*, 50:306–329, 2013.

5 S.W. Bae, M. Korman, and Y. Okamoto. Computing the geodesic centers of a polygonal domain. In *Proc. of the 26th Canadian Conference on Computational Geometry*, 2014.

6 S.W. Bae, M. Korman, Y. Okamoto, and H. Wang. Computing the L_1 geodesic diameter and center of a simple polygon in linear time. *Computational Geometry: Theory and Applications*, 48:495–505, 2015.

7 R. Bar-Yehuda and B. Chazelle. Triangulating disjoint Jordan chains. *International Journal of Computational Geometry and Applications*, 4(4):475–481, 1994.

8 B. Chazelle. A theorem on polygon cutting with applications. In *Proc. of the 23rd Annual Symposium on Foundations of Computer Science*, pages 339–349, 1982.

9 D.Z. Chen and H. Wang. A nearly optimal algorithm for finding L_1 shortest paths among polygonal obstacles in the plane. In *Proc. of the 19th European Symposium on Algorithms*, pages 481–492, 2011.

10 D.Z. Chen and H. Wang. Computing the visibility polygon of an island in a polygonal domain. In *Proc. of the 39th International Colloquium on Automata, Languages and Programming*, pages 218–229, 2012. Journal version published online in *Algorithmica*, 2015.

11 D.Z. Chen and H. Wang. L_1 shortest path queries among polygonal obstacles in the plane. In *Proc. of the 30th Symposium on Theoretical Aspects of Computer Science*, pages 293–304, 2013.

12 D.Z. Chen and H. Wang. Visibility and ray shooting queries in polygonal domains. *Computational Geometry: Theory and Applications*, 48:31–41, 2015.

13 H. Edelsbrunner, L.J. Guibas, and M. Sharir. The upper envelope of piecewise linear functions: Algorithms and applications. *Discrete and Computational Geometry*, 4:311–336, 1989.

14 L.J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R.E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1-4):209–233, 1987.

15 J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Computational Geometry: Theory and Applications*, 4(2):63–97, 1994.

16 J. Hershberger and S. Suri. Matrix searching with the shortest-path metric. *SIAM Journal on Computing*, 26(6):1612–1634, 1997.

17 R. Inkulu and S. Kapoor. Planar rectilinear shortest path computation using corridors. *Computational Geometry: Theory and Applications*, 42(9):873–884, 2009.

18 S. Kapoor, S.N. Maheshwari, and J.S.B. Mitchell. An efficient algorithm for Euclidean shortest paths among polygonal obstacles in the plane. *Discrete and Computational Geometry*, 18(4):377–383, 1997.

19 J.S.B. Mitchell. An optimal algorithm for shortest rectilinear paths among obstacles. In *the 1st Canadian Conference on Computational Geometry*, 1989.

20 J.S.B. Mitchell. L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica*, 8(1):55–88, 1992.

21 R. Pollack, M. Sharir, and G. Rote. Computing the geodesic center of a simple polygon. *Discrete and Computational Geometry*, 4(1):611–626, 1989.

22 S. Schuierer. Computing the L_1-diameter and center of a simple rectilinear polygon. In *Proc. of the International Conference on Computing and Information*, pages 214–229, 1994.

23 S. Suri. Computing geodesic furthest neighbors in simple polygons. *Journal of Computer and System Sciences*, 39:220–235, 1989.

Are Short Proofs Narrow? QBF Resolution is *not* Simple

Olaf Beyersdorff[1], Leroy Chew[2], Meena Mahajan[3], and Anil Shukla[4]

1 **School of Computing, University of Leeds, United Kingdom**
2 **School of Computing, University of Leeds, United Kingdom**
3 **The Institute of Mathematical Sciences, Chennai, India**
4 **The Institute of Mathematical Sciences, Chennai, India**

Abstract

The groundbreaking paper 'Short proofs are narrow – resolution made simple' by Ben-Sasson and Wigderson (J. ACM 2001) introduces what is today arguably *the* main technique to obtain resolution lower bounds: to show a lower bound for the width of proofs. Another important measure for resolution is space, and in their fundamental work, Atserias and Dalmau (J. Comput. Syst. Sci. 2008) show that space lower bounds again can be obtained via width lower bounds.

Here we assess whether similar techniques are effective for resolution calculi for quantified Boolean formulas (QBF). A mixed picture emerges. Our main results show that both the relations between size and width as well as between space and width drastically *fail* in Q-resolution, even in its weaker tree-like version. On the other hand, we obtain positive results for the expansion-based resolution systems ∀Exp+Res and IR-calc, however only in the weak tree-like models.

Technically, our negative results rely on showing width lower bounds together with simultaneous upper bounds for size and space. For our positive results we exhibit space and width-preserving simulations between QBF resolution calculi.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems: Complexity of proof procedures

Keywords and phrases proof complexity, QBF, resolution, lower bound techniques, simulations

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.15

1 Introduction

The main objective in *proof complexity* is to obtain precise bounds on the size of proofs in various formal systems; and this objective is closely linked to and motivated by foundational questions in computational complexity (Cook's program), first-order logic (separating theories of bounded arithmetic), and SAT solving. In particular, resolution is one of the best studied and most important propositional proof systems, as it forms the backbone of modern SAT solvers based on conflict-driven clause learning (CDCL). Complexity bounds for resolution proofs directly translate into bounds on the performance of SAT solvers.

What is arguably even more important than showing the actual bounds is to develop *general techniques* that can be applied to obtain lower bounds for important proof systems. A number of ingenious techniques have been designed to show lower bounds for the *size of resolution proofs*, among them feasible interpolation [22], which applies to many further systems. In their pioneering paper [7], Ben-Sasson and Wigderson showed that resolution size lower bounds can be elegantly obtained by showing lower bounds to the *width* of resolution proofs. Indeed, the discovery of this relation between width and size of resolution proofs was

© Olaf Beyersdorff, Leroy Chew, Meena Mahajan, and Anil Shukla;
licensed under Creative Commons License CC-BY

a milestone in our understanding of resolution, and today many if not most lower bounds for resolution are obtained via the size-width technique.

Another important measure for resolution is *space* [18], as it corresponds to memory requirements of solvers in the same way as resolution size relates to their running time. In their fundamental work [1], Atserias and Dalmau demonstrated that also space is tightly related to width. Indeed, showing lower bounds for width serves again as the primary method to obtain space lower bounds. Since these discoveries the relations between resolution size, width, and space have been subject to intense research (cf. [14]), and in particular sharp trade-off results between the measures have been obtained (cf. e.g. [4, 6, 24]).

In this paper we initiate the study of width and space in resolution calculi for quantified Boolean formulas (QBF) and address the question whether similar relations between size, width, and space as for classical resolution hold in QBF. Before explaining our results we sketch recent developments in QBF proof complexity.

QBF proof complexity is a relatively young field studying proof systems for quantified Boolean logic. Similarly as in the propositional case, one of the main motivations for the field comes via its intimate connection to solving. Although QBF solving is at an earlier state than SAT solving, due to its PSPACE completeness, QBF even applies to further fields such as formal verification or planning [25, 8, 17]. Each successful run of a solver on an unsatisfiable instance can be interpreted as a proof of unsatisfiability; and this connection turns proof complexity into the main theoretical tool to understand the performance of solving. As in SAT, QBF solvers are known to correspond to the resolution proof system and its variants.

However, compared to SAT, the QBF picture is more complex as there exist two main solving approaches utilising CDCL and expansion-based solving. To model the strength of these QBF solvers, a number of resolution-based QBF proof systems have been developed. Q-resolution (Q-Res) by Kleine Büning, Karpinski, and Flögel [21] forms the core of the CDCL-based systems. To capture further ideas from CDCL solving, Q-Res has been augmented to long-distance resolution [28, 2], universal resolution [27], and their combinations [3]. Powerful proof systems for expansion-based solving were recently developed in the form of ∀Exp+Res [20], and the stronger IR-calc and IRM-calc [10].

In this paper we concentrate on the three QBF resolution systems Q-Res, ∀Exp+Res, and IR-calc. This choice is motivated by the fact that Q-Res and ∀Exp+Res form the base systems for CDCL and expansion-based solving, respectively, and IR-calc unifies both approaches in a natural way, as it simulates both Q-Res and ∀Exp+Res [10]. Recent findings show that CDCL and expansion are indeed orthogonal paradigms as Q-Res and ∀Exp+Res are incomparable with respect to simulations [11].

Understanding which lower bound techniques are effective in QBF proof complexity is important for progress in the field. In [12], the feasible interpolation technique was shown to apply to all QBF resolution systems. Another successful transfer of a classical technique was obtained in [13] for a game-theoretic characterisation of proof size in tree-like Q-Res.

Our Contributions

The central question we address here is whether *lower bound techniques via width*, which have revolutionised classical proof complexity, are also effective for QBF resolution systems.

Though space and width have not been considered in QBF before, these notions straightforwardly apply to QBF resolution systems. However, due to the ∀-reduction rule in Q-Res handling universal variables, it is relatively easy to enforce that universal literals accumulate in clauses of Q-Res proofs, thus always leading to large width, irrespective of size and space requirements (Lemma 4). This prompts us to consider *existential width* – counting only

existential literals – as an appropriate width measure in QBF. This definition aligns both with Q-Res, resolving only on existential variables, as well as with ∀Exp+Res and IR-calc, which like all expansion systems only operate on existential literals.

1. Negative results. Our main results show that the size-width relation of [7] as well as the space-width relation of [1] dramatically *fail* for Q-Res, even when considering the tighter existential width. We first notice that the proof establishing the size-width result in [7] almost fully goes through, except for some very inconspicuous step that fails in QBF (Proposition 5). But not only the technique fails: we prove that Tseitin transformations of formulas expressing a natural completion principle from [20] have small size and space, but require large existential width in tree-like Q-Res (Theorem 6), thus refuting the size-width relation for tree-like Q-Res as well as the space-width relation for general dag-like Q-Res.

As the formulas for the completion principle have $O(n^2)$ variables, they do not rule out size-width relations in general Q-Res. However, we show that different formulas, hard for tree-like Q-Res [20], provide counterexamples for size-width relations in full Q-Res (Theorem 7).

Technically, our main contributions are width lower bounds for the above formulas, which we show by careful counting arguments. We complement these results by existential width lower bounds for parity-formulas from [11], providing an optimal width separation between Q-Res and ∀Exp+Res (Theorem 17).

2. Positive results and width-space-preserving simulations. Though the negative picture above prevails, we prove some positive results for size-width-space relations for tree-like versions of the expansion resolution systems ∀Exp+Res and IR-calc. Proofs in ∀Exp+Res can be decomposed into two clearly separated parts: an expansion phase followed by a classical resolution phase. This makes it easy to transfer almost the full spectrum of the classical relations to ∀Exp+Res (Theorem 18).

To lift these results to IR-calc (Theorem 19), we show a series of careful space and width-preserving simulations between tree-like Q-Res, ∀Exp+Res, and IR-calc. In particular, we show the surprising result that tree-like ∀Exp+Res and tree-like IR-calc are equivalent (Lemma 14), thus providing a rare example of two proof systems that coincide in the tree-like, but are separated in the dag-like model [11]. The only other such example that we are aware of is regular resolution vs. full resolution (although this is perhaps slightly less natural as regular resolution is just a sub-system of resolution). In addition, our simulations provide a simpler proof for the simulation of tree-like Q-Res by ∀Exp+Res (Corollary 16), shown in [20] via a very involved argument.

Our last positive result is a size-space relation in tree-like Q-Res (Theorem 19), which we show by a pebbling game analogous to the classical relation in [18]. Not surprisingly, this only positive result for Q-Res avoids any reference to the notion of width.

As the bottom line we can say that QBF proof complexity is not just a replication of classical proof complexity: it shows quite different and interesting effects as we demonstrate here. Especially for lower bounds it requires new ideas and techniques. We remark that in this direction, a new and 'genuine QBF technique' based on strategy extraction was recently developed, showing lower bounds for Q-Res [11] and indeed much stronger systems [9].

Organisation of the paper. We start by reviewing background information on classical and QBF resolution systems (Sect. 2), including definitions of size, space, and width together with their main classical relations (Sect. 3). In Sect. 4 we prove our main negative results on the failure of the transfer of the classical size-width and space-width results to QBF.

Section 5 contains the simulations between tree-like versions of Q-Res, ∀Exp+Res, and IR-calc, paying special attention to width and space. This enables us to show in Sect. 6 the positive results for relations between size, width, and space in these systems. We conclude in Sect. 7 with a discussion and directions for future research.

2 Notations and Preliminaries

Quantified Boolean formulas. A (closed prenex) *quantified Boolean formula* (QBF) is a formula in quantified propositional logic where each variable is quantified at the beginning of the formula, using either an existential or universal quantifier. We denote such formulas as $\mathcal{Q}\,.\,\phi$, where ϕ is a propositional Boolean formula in conjunctive normal form (CNF), called *matrix*, and $\mathcal{Q}$ is its *quantifier prefix*. The *quantification level* $\mathrm{lv}(y)$ of a variable y in $\mathcal{Q}\,.\,\phi$ is the number of alternations of quantifiers y has on its left in the quantifier prefix of $\mathcal{Q}\,.\,\phi$.

Classical resolution. *Resolution* (Res), introduced by Blake [15] and Robinson [26], is a refutational proof system manipulating unsatisfiable CNFs as sets of clauses. The only inference rule is $\dfrac{C \vee x \qquad D \vee \neg x}{C \vee D}$ where C, D denote clauses and x is a variable. A Res refutation derives the empty clause $\square$. If we only allow proofs in form of a tree, i.e., each derived clause can be used at most once, we speak of *tree-like resolution*, denoted Res_T.

QBF resolution calculi. *Q-resolution* (Q-Res) [21] is a resolution-like calculus that operates on QBFs in prenex form where the matrix is a CNF. It uses the propositional resolution rule above with the side conditions that variable x is existential and if $z \in C$, then $\neg z \notin D$. In addition Q-Res has a universal reduction rule $\dfrac{C \vee u}{C}$ (∀-Red) where variable u is universal and all other existential variables $x \in C$ are left of u in the quantifier prefix.

In addition to Q-Res we consider two further QBF resolution calculi that have been introduced to model *expansion-based QBF solving*. These calculi are based on *instantiation* of universal variables: ∀Exp+Res [20], and IR-calc [10]. Both calculi operate on clauses that comprise only existential variables from the original QBF, which are additionally *annotated* by a substitution to some universal variables, e.g. $\neg x^{u/0,v/1}$. For any annotated literal l^σ, the substitution σ must not make assignments to variables right of l, i.e. if $u \in \mathsf{dom}(\sigma)$, then u is universal and $\mathrm{lv}(u) < \mathrm{lv}(l)$. To preserve this invariant, we use the *auxiliary notation* $l^{[\sigma]}$, which for an existential literal l and an assignment σ to the universal variables filters out all assignments that are not permitted, i.e. $l^{[\sigma]} = l^{\{u/c \in \sigma \mid \mathrm{lv}(u) < \mathrm{lv}(l)\}}$. We say that an assignment is complete if its domain is all universal variables. Likewise, we say that a literal x^τ is fully annotated if all universal variables u with $\mathrm{lv}(u) < \mathrm{lv}(x)$ in the QBF are in $\mathsf{dom}(\tau)$, and a clause is fully annotated if all its literals are fully annotated.

The calculus ∀Exp+Res from [20] works with fully annotated clauses on which resolution is performed. For each clause C from the matrix and an assignment τ to all universal variables, ∀Exp+Res can use the axiom $\{l^{[\tau]} \mid l \in C, l \text{ existential}\} \cup \{\tau(l) \mid l \in C, l \text{ universal}\}$. As its only rule it uses the resolution rule on annotated variables

$$\frac{C \vee x^\tau \qquad D \vee \neg x^\tau}{C \vee D}\ (\text{Res}).$$

In contrast, the system IR-calc from [10] is more flexible. It uses 'delayed' expansion and can mix instantiation with resolution steps. Formally, IR-calc works with partial assignments on which we use auxiliary operations of *completion* and *instantiation*. For assignments τ and μ, we write $\tau \trianglelefteq \mu$ for the assignment σ defined as $\sigma(x) = \tau(x)$ if $x \in \mathsf{dom}(\tau)$, otherwise

$\sigma(x) = \mu(x)$ if $x \in \mathsf{dom}(\mu)$. The operation $\tau \veebar \mu$ is called *completion* as μ provides values for variables not defined in τ. For an assignment τ and an annotated clause C, the function $\mathsf{inst}(\tau, C)$ returns the annotated clause $\{l^{[\sigma \veebar \tau]} \mid l^\sigma \in C\}$.

Axioms in IR-calc allow to infer $\{x^{[\tau]} \mid x \in C, x \text{ is existential}\}$ for each non-tautological clause C from the matrix and $\tau = \{u/0 \mid u \text{ is universal in } C\}$, where the notation $u/0$ for literals u is shorthand for $x/0$ if $u = x$ and $x/1$ if $u = \neg x$. Rules in IR-calc comprise the (Res) rule above together with the instantiation rule $\dfrac{C}{\mathsf{inst}(\tau, C)}$ for a (partial) assignment τ to universal variables.

Simulations. Given two proof systems P and Q for the same language (TAUT or QBF), P *p-simulates* Q if each Q-proof can be transformed in polynomial time into a P-proof of the same formula. Two systems are called *p-equivalent* if they p-simulate each other.

In [10] it was shown that IR-calc p-simulates both Q-Res and ∀Exp+Res, while [11] shows that Q-Res and ∀Exp+Res are incomparable, i.e., IR-calc is exponentially stronger than both Q-Res and ∀Exp+Res. However, ∀Exp+Res can p-simulate $\mathsf{Q\text{-}Res}_\mathsf{T}$ [20].

3 Size, Width, and Space in Resolution Calculi

The purpose of the section is twofold: first to review the measures size, width, and space and their relations in classical resolution; and second to explain how to apply these measures to QBF resolution systems. While this is straightforward for size and space, we need a more elaborate discussion on what constitutes a good notion of width for QBF resolution systems.

3.1 Defining Size, Width, and Space for Resolution

For a CNF F, $|F|$ is the number of clauses in it, and $w(F)$ denotes the maximum number of literals in any clause of F. We extend the same notation to QBFs with a CNF matrix.

For P one of the resolution calculi Res, Q-Res, ∀Exp+Res, IR-calc, let $\pi \vdash_P F$ (resp. $\pi \vdash_{P_\mathsf{T}} F$) denote that π is an P-proof (tree-like P-proof, respectively) of the formula F. For a proof π of F in system P, its size $|\pi|$ is defined as the number of clauses in π. The *size complexity* $S(\vdash_P F)$ of deriving F in P is defined as $\min\{|\pi| \ : \ \pi \vdash_P F\}$. The tree-like size complexity, denoted $S(\vdash_{P_\mathsf{T}} F)$, is $\min\{|\pi| \ : \ \pi \vdash_{P_\mathsf{T}} F\}$.

The *width* of a clause C is the number of literals in C, denoted $w(C)$. The width $w(F)$ of a CNF F is the maximum width of a clause in F. The width $w(\pi)$ of a proof π is the maximum width of any clause appearing in π, and the width $w(\vdash_P F)$ of refuting a CNF F in P is defined as $\min\{w(\pi) \ : \ \pi \vdash_P F\}$. Again the same notation extends to quantified CNFs.

Note that for width in any calculus, whether the proof is tree-like or not is immaterial, since a proof can always be made tree-like by duplication without increasing the width. We therefore drop the T subscript when talking about proof width.

The third complexity measure for resolution calculi is *space*[1], first defined in [18]. Informally, it is the minimal number of clauses that must be kept simultaneously to refute a formula. We view a proof as a sequence of CNF formulas $F_0, F_1, \ldots, F_s$, where $F_0 = \emptyset$, $\square \in F_s$, and each F_{i+1} is obtained from F_i by erasing some clause, downloading an axiom, or adding a clause derived by some P-rule from clauses in F_i. In the last case, one of the premises of the inference may also simultaneously be deleted. For such a proof σ, $\mathit{CSpace}(\sigma)$ is the

[1] Also called clause space, to distinguish it from variable space or total space (see for example, [5]). We consider only clause space in this paper, and so we call it just space.

maximum number of clauses in any F_i, $i \in [s]$. The space to refute F, denoted $CSpace(\vdash_{\mathsf{P}} F)$, is the minimum $CSpace(\sigma)$ over all P-refutations σ for F. The same notions apply to QBFs, where $F_0, F_1, \ldots, F_s$ is a sequence of CNF formulas, all with the same quantifier prefix.

If we modify the inference step so that the clause(s) used to obtain the inference are erased in the same step, then any clause can be used at most once and we obtain a tree-like space-oriented P-proof. Correspondingly we can define $CSpace(\vdash_{\mathsf{P_T}} F)$ as the minimum space used by any tree-like proof sequence refuting F.

3.2 Relations in Classical Resolution

We now state some of the main relations between size, width, and space for classical resolution. We start with the foundational size-width relations of Ben-Sasson and Wigderson [7].

▶ **Theorem 1** (Ben-Sasson, Wigderson [7]). *For all unsatisfiable CNFs F in n variables the following holds:* $S(\vdash_{\mathsf{Res_T}} F) \geq 2^{w\left(\vdash_{\mathsf{Res}} F\right) - w(F)}$ *and* $S(\vdash_{\mathsf{Res}} F) = \exp\left(\Omega\left(\frac{\left(w\left(\vdash_{\mathsf{Res}} F\right) - w(F)\right)^2}{n}\right)\right)$.

Space complexity was introduced in [18] and relations between space, size, and width are explored (cf. also [23, 14]), establishing the size-space relation for tree-like resolution:

▶ **Theorem 2** (Esteban, Torán [18]). *For all unsatisfiable CNFs F the following relation holds:* $S(\vdash_{\mathsf{Res_T}} F) \geq 2^{CSpace\left(\vdash_{\mathsf{Res_T}} F\right)} - 1$.

The fundamental relation between space and width for full resolution was obtained in [1]; a more direct proof was given recently in [19].

▶ **Theorem 3** (Atserias, Dalmau [1]). *For all unsatisfiable CNFs F the following relation holds:* $w(\vdash_{\mathsf{Res}} F) \leq CSpace(\vdash_{\mathsf{Res}} F) + w(F) - 1$.

3.3 Existential Width: What Is the Right Width Notion for QBF?

We wish to explore the possibility of a similar approach as in [7] to prove analogues of the classical results above for QBFs. The following simple example shows, however, that the relationships in Theorem 1 and Theorem 3 do not carry over for the system Q-Res.

▶ **Proposition 4.** *For the false QBFs* $\mathcal{F}_n = \forall u_1 \ldots u_n \exists e_0 \exists e_1 \ldots e_n . (e_0) \wedge \bigwedge_{i \in [n]} (\neg e_{i-1} \vee u_i \vee e_i) \wedge (\neg e_n)$ *we have* $S(\vdash_{\mathsf{Q\text{-}Res_T}} \mathcal{F}_n) = O(n)$ *and* $CSpace(\vdash_{\mathsf{Q\text{-}Res_T}} \mathcal{F}_n) = O(1)$*, but* $w(\vdash_{\mathsf{Q\text{-}Res}} \mathcal{F}_n) = \Omega(n)$.

As this example illustrates, it is easy to enforce that universal variables are accumulated in a clause, thus leading to large width. Hence the following question naturally arises: can we obtain size-width or space-width relations by using the tighter measure of only counting existential variables?

This aligns with the situation in the expansion systems ∀Exp+Res and IR-calc, where clauses contain only existential variables. In this respect, it is worth noting that the above example indeed does not demonstrate the failure of the size-width relationship in expansion-based calculi. For instance, in ∀Exp+Res, a tree-like refutation could download the existential variables of axioms annotated with $u_i/0$ for $i \in [n]$, and generate the empty clause in $O(n)$ steps with width just 2 at the leaves and 1 at the internal nodes.

Thus, to get a consistent and interesting width measure for QBF calculi, we consider the notion of *existential width* that just counts the number of existential literals. This approach is justified also for Q-Res as the calculus can only resolve on existential variables, and rules

out the easy counterexamples above. Formally, we define the existential width of a clause C to be the number of existential literals in C, and denote it by $w_\exists(C)$. Using $w_\exists$ instead of w everywhere, we obtain the existential width of a formula $w_\exists(F)$, of a proof $w_\exists(\pi)$, and of refuting a false sentence $w_\exists(\vdash_{\mathsf{P}} \mathcal{F})$.

For the expansion systems $\forall$Exp+Res and IR-calc the notions of existential width and width coincide. (In particular, distinct annotations of the same existential variable are counted as distinct literals.) Hence we can drop the $\exists$ subscript in width of proofs in these systems. For the width of the sentence itself, there is still a difference between w and $w_\exists$.

4 Negative Results: Size-Width, Space-Width Relations Fail in Q-Res

In this section we show that in the Q-Res proof system, even replacing width by existential width, the relations to size or space as in classical resolution (Theorems 1 and 3) no longer hold for both tree-like and general proofs.

Firstly, we point out where the technique of [7] fails. A crucial ingredient of their proof is the following statement: if a clause A can be derived from $F|_{x=1}$ in width w, then the clause $A \vee \neg x$ can be derived from F in width $w+1$ (possibly using a weakening rule at the end). We show that the statement no longer holds in Q-Res.

▶ **Proposition 5.** *There are false sentences ψ_n, with an existential literal b quantified at the innermost level, such that the sentence $\psi_n|_{b=1}$ is false and has a small existential-width proof, but ψ_n itself needs large existential width to refute in Q-Res.*

Proof. The sentence ψ_n is constructed by taking the conjunction of two sentences with distinct variables. The first sentence is a very simple one: $\exists a \forall u \exists b.\, (a \vee u \vee \neg b) \wedge (\neg a)$. It is a true sentence, but if b is set to 1, it becomes false. The second sentence is a false sentence of the form $\exists \vec{x}.\, G_n(\vec{x})$, where G_n is any unsatisfiable CNF formula over the $\vec{x}$ variables, such that G_n needs large width in classical resolution. One such example is the CNF formula described by Bonet and Galesi [16], that we denote as BG_n. BG_n is an unsatisfiable 3-CNF formula over $O(n^2)$ variables with $w(\vdash BG_n) = \Omega(n)$.

Now define ψ_n as $\exists \vec{x} \exists a \forall u \exists b.\, (a \vee u \vee \neg b) \wedge (\neg a) \wedge BG_n(\vec{x})$. Note that the clauses $(a \vee u \vee \neg b) \wedge (\neg a)$ contain a contradiction if and only if $b = 1$. Thus $\psi_n|_{b=1}$ can be refuted with existential width 1 using just these two clauses: a $\forall$-Red on $(a \vee u)$ yields a which can be resolved with $\neg a$. On the other hand, to refute ψ_n, the contradiction in BG_n must be exposed. Since all the variables involved are existential, Q-Res degenerates to classical resolution, requiring (existential) width $\Omega(n)$. ◀

The example in Proposition 5 can be made 'less degenerate' by interleaving more existential and universal variables disjoint from $\vec{x}$ and putting them in the first sentence. All we need is that b is quantified existentially at the end, the first sentence is true as a whole but false if $b = 1$, and this latter sentence can be refuted in Q-Res with small existential width.

We now show that it is not just the technique of [7] that fails for Q-Res. No other technique will work either, because the relation from Theorem 1 between size and existential width itself fails to hold. The same example shows that the relation from Theorem 3 between space and existential width also fails to hold.

We first give an example where the relation for tree-like proofs fails.

▶ **Theorem 6.** *There exist false QBFs CR'_n over $O(n^2)$ variables, such that $S(\vdash_{\text{Q-Res}_T} CR'_n) = n^{O(1)}$, $w_\exists(CR'_n) = 3$, $CSpace(\vdash_{\text{Q-Res}_T} CR'_n) = O(1)$, and $w_\exists(\vdash_{\text{Q-Res}_T} CR'_n) = \Omega(n)$.*

The formulas CR'_n are Tseitin transformations of a natural completion principle formula CR_n from [20]. The proof is similar, but slightly more involved than the proof for our next Theorem 7. Since tree-like space is at least as large as space, Theorem 6 also rules out the space-width relation for general dag-like Q-Res proofs.

However, Theorem 6 cannot be used to show that the size-existential-width relationship for general dag-like proofs fails in Q-Res, because CR'_n have $O(n^2)$ variables. We show via another example that the relation fails to hold in Q-Res as well. This example cannot be used for proving Theorem 6 because it is known to be hard for Q-Res$_\mathsf{T}$ [20].

▶ **Theorem 7.** *There is a family of false QBFs ϕ'_n in $O(n)$ variables such that $S(\vdash_{\text{Q-Res}} \phi'_n) = n^{O(1)}$, $w_\exists(\phi'_n) = 3$, and $w_\exists(\vdash_{\text{Q-Res}} \phi'_n) = \Omega(n)$.*

Proof. Consider the following formulas ϕ_n, introduced by Janota and Marques-Silva [20]: $\exists e_1 \forall u_1 \exists c_1 c_2 \dots \exists e_n \forall u_n \exists c_{2n-1} c_{2n}$.

$$\bigwedge_{i\in[n]} \big((\neg e_i \vee c_{2i-1}) \wedge (\neg u_i \vee c_{2i-1}) \wedge (e_i \vee c_{2i}) \wedge (u_i \vee c_{2i})\big) \wedge \bigvee_{i\in[2n]} \neg c_i.$$

We know from [20] that ϕ_n have polynomial-size proofs in Q-Res (but require exponential-size proofs in Q-Res$_\mathsf{T}$). However, we need a formula with constant initial width. To achieve this we consider quantified Tseitin transformations of ϕ_n, i.e. we introduce $2n+1$ new existential variables x_i at the innermost quantification level in ϕ_n, and replace the only large clause in ϕ_n by any CNF formula that preserves satisfiability. Let ϕ'_n denote the modified formula:

$$\phi'_n = \exists e_1 \forall u_1 \exists c_1 c_2 \dots \exists e_n \forall u_n \exists c_{2n-1} c_{2n} \exists x_0 \dots x_{2n}$$

$$\bigwedge_{i\in[n]} \big((\neg e_i \vee c_{2i-1}) \wedge (\neg u_i \vee c_{2i-1}) \wedge (e_i \vee c_{2i}) \wedge (u_i \vee c_{2i})\big) \wedge \tag{1}$$

$$\neg x_0 \wedge \bigwedge_{i\in[2n]} (x_{i-1} \vee \neg c_i \vee \neg x_i) \wedge x_{2n}. \tag{2}$$

Note that $w_\exists(\phi'_n) = 3$. We refer to the clauses in (2) as x-clauses. It is clear that from the x-clauses, we can derive the large clause of ϕ_n in $2n+1$ resolution steps and get back ϕ_n. Thus $S(\vdash_{\text{Q-Res}} \phi'_n) \le S(\vdash_{\text{Q-Res}} \phi_n) + 2n + 1 \in n^{O(1)}$.

We now show that ϕ'_n needs large existential width. Let π be a proof in Q-Res, $\pi \vdash_{\text{Q-Res}} \phi'_n$. List the clauses of π in sequence, $\pi = \{D_0, D_1, \dots, D_s = \square\}$, where each clause in the sequence is either a clause from ϕ'_n, or is derived from clause(s) preceding it in the sequence using resolution or $\forall$-Red. There must be at least one universal reduction step in π, since all the initial clauses are necessary for refuting ϕ'_n, some of them contain universal variables, and the only way to remove a universal variable in Q-Res is by $\forall$-Red. Let i be the least index such that the clause D_i is obtained by $\forall$-Red on D_j for some $0 < i$. Since all x variables block all u variables, D_j and D_i cannot contain any x variables. We use this fact to show that $w_\exists(D_i) = \Omega(n)$. Our strategy is to associate some set with each clause in π in a specific way, and use the set size to bound existential width.

We associate the following sets with the literals of ϕ'_n and the clauses of π.

$$\begin{array}{lrcl}
 & \sigma(x_0) & = & \emptyset \\
\forall i \in [2n] & \sigma(x_i) & = & [i] = \{1, 2, \dots, i\} \\
 & \sigma(\neg x_0) & = & [2n] \\
\forall i \in [2n] & \sigma(\neg x_i) & = & [2n] \setminus [i] = \{i+1, \dots, 2n\} \\
\forall i \in [n] & \sigma(e_i) = \sigma(u_i) = \sigma(\neg c_{2i}) = \sigma(c_{2i-1}) & = & \{2i\} \\
\forall i \in [n] & \sigma(\neg e_i) = \sigma(\neg u_i) = \sigma(\neg c_{2i-1}) = \sigma(c_{2i}) & = & \{2i-1\} \\
\forall D \in \pi & \sigma(D) & = & \bigcup_{l\in D} \sigma(l).
\end{array}$$

Note that for any literal ℓ, $\sigma(\ell)$ and $\sigma(\neg\ell)$ are disjoint.

For $D \in \pi$, let π_D be the sub-DAG of π, rooted at D.

▶ **Claim 8.** *π_{D_i} contains at least one x-clause (axiom clause of type (2)).*

Proof. The parent D_j of node D_i contains a universal variable which is then removed through $\forall$-Red to get D_i. The universal variables appear only in clauses of type (1), but are blocked by the c-variables in every clause where they appear. Thus, before a reduction is permitted, a c-variable must be eliminated by resolution. Since all c-variables appear only positively in type (1) clauses, some x-clause must be used in the resolution. ◀

We show that all clauses in π_{D_i} that are descendants of some x-clause have large sets associated with them. In particular, we show:

▶ **Claim 9.** *Every clause D in π_{D_i} such that π_D contains an x-clause has $\sigma(D) = [2n]$.*

Deferring the proof briefly, we continue with our argument. From Claim 9 we conclude that $\sigma(D_i) = [2n]$. Recall that none of the x variables belongs to D_i. All other literals are associated with singleton sets, so D_i must contains at least $2n$ literals in order to be associated with the complete set $[2n]$. Since Q-Res proofs prohibit a variable and its negation in the same clause, at most n of the literals in D_i can be universal variables. Thus D_i has at least n existential literals, hence $w_\exists(D_i) = \Omega(n)$.

It remains to establish the claimed set size.

Proof of Claim 9. We proceed by induction on the depth of descendants of x-clauses in π_{D_i}. The base case is an x-clause itself and follows from the definition of σ.

For the inductive step, let D be obtained by resolving $(E \vee z)$ and $(F \vee \neg z)$. There are two cases to consider:

Case 1: Both $(E \vee z)$ and $(F \vee \neg z)$ are descendants of x-clauses (not necessarily the same x-clause). Then by induction, $\sigma(E \vee z) = \sigma(F \vee \neg z) = [2n]$. So $\sigma(E) \supseteq [2n] \setminus \sigma(z)$ and $\sigma(F) \supseteq [2n] \setminus \sigma(\neg z)$. Since $\sigma(z)$ and $\sigma(\neg z)$ are disjoint, $\sigma(E) \cup \sigma(F) = [2n]$. Thus $\sigma(D) = \sigma(E) \cup \sigma(F) = [2n]$ as claimed.

Case 2: Exactly one of $(E \vee z)$ and $(F \vee \neg z)$ is a descendant of an x-clause. Without loss of generality, let $F \vee \neg z$ be the descendant. Then $E \vee z$ is either a type-(1) clause or is derived solely from type-(1) clauses using resolution. However, observe that the only clauses derivable solely from type-(1) clauses via resolution, without creating tautologies as mandated in Q-Res, are of the form $(c_{2i-1} \vee c_{2i})$ for some i. It follows that z is not an x variable. Hence $\sigma(z)$ and $\sigma(\neg z)$ are distinct singleton sets. Further, z cannot be a u variable either, since resolution on universal variables is not permitted in Q-Res.

Now note that for any type-(1) clause C, $\sigma(C) = \{2i-1, 2i\}$ for the appropriate i. Similarly, $\sigma(c_{2i-1} \vee c_{2i}) = \{2i-1, 2i\}$. So if $E \vee z$ is one of these clauses, then $\sigma(E \vee z) = \sigma(z) \cup \sigma(\neg z)$ and $\sigma(E) = \sigma(\neg z)$. Further, as in Case 1, by induction we know that $\sigma(F \vee \neg z) = [2n]$ and $\sigma(F) \supseteq [2n] \setminus \sigma(\neg z)$. Hence, $\sigma(E \vee F) = [2n]$ as claimed. ◀

This completes the proof of the theorem. ◀

The above counterexamples are provided by formulas that require small size, but large existential width. We will now illustrate via another example that also *large size and large width* can occur. These examples are very natural formulas based on the parity function, which have recently been used in [11] to show exponential size lower bounds for Q-Res,

and indeed a separation between Q-Res and ∀Exp+Res. We will later use these formulas in Section 5 to also show a separation for width between Q-Res and ∀Exp+Res.

Let $\mathrm{xor}(o_1, o_2, o)$ be the set of clauses expressing $o \equiv o_1 \oplus o_2$; that is, $\{\neg o_1 \vee \neg o_2 \vee \neg o,\ o_1 \vee o_2 \vee \neg o,\ \neg o_1 \vee o_2 \vee o,\ o_1 \vee \neg o_2 \vee o\}$. In [11], the sentence $\mathrm{QPARITY}_n$ is defined as follows:

$$\exists x_1, \ldots, x_n\, \forall z\, \exists t_2, \ldots, t_n.\ \mathrm{xor}(x_1, x_2, t_2) \cup \bigcup_{i=3}^{n} \mathrm{xor}(t_{i-1}, x_i, t_i) \cup \{z \vee t_n, \neg z \vee \neg t_n\}.$$

The x_i variables act as the input for the parity function, and the t_i variables are defined inductively to calculate $\mathrm{PARITY}(x_1, \ldots, x_i)$.

We now complement the exponential size lower bound from [11] by a width lower bound.

▶ **Theorem 10.** $w_\exists(\vdash_{\text{Q-Res}} \mathrm{QPARITY}_n) \geq n$.

Proof. In the formula $\mathrm{QPARITY}_n$, the contradiction occurs semantically because of the clauses $z \vee t_n$, $\neg z \vee \neg t_n$ asserting $z \neq t_n$ (along with the fact that the values of x variables uniquely determine the values of all t variables, in particular, t_n). Thus, at least one of these clauses must be used in any proof, necessitating a ∀-reduction. In Q-Res we cannot reduce z while any of the t variables are present; and due to the restrictions in Q-Res we cannot resolve any descendants of $z \vee t_n$ with any descendants of $\neg z \vee \neg t_n$ until there is at least one ∀-reduction.

Consider a smallest Q-Res proof, and assume without loss of generality that a first (lowest) ∀ reduction happens on the positive literal z. Therefore before this ∀-reduction step we have essentially a resolution proof π from $\Gamma = \mathrm{xor}(x_1, x_2, t_2) \cup \bigcup_{i=3}^{n} \mathrm{xor}(t_{i-1}, x_i, t_i) \cup \{t_n \vee z\}$. The clause D that occurs in π immediately before the ∀-reduction must only contain variables from $\{x_1, \ldots, x_n\}$ apart from the literal z, else the reduction is blocked.

We now use the following observation.

▶ **Claim 11.** *Suppose $x_1 \oplus \cdots \oplus x_n \vDash C$ for some clause C. Then C is either a tautology or C contains all variables $x_1, \ldots, x_n$.*

Any assignment to the x variables satisfying $x_1 \oplus \cdots \oplus x_n$ has a unique extension to z and the t variables satisfying all clauses of the formula $\mathrm{QPARITY}_n$. This extension necessarily has $t_n = x_1 \oplus \cdots \oplus x_n = 1$ and $z = 0$. Since it satisfies all axioms, by soundness of resolution, it also satisfies D.

This, along with Claim 11, implies that D is either a tautology or has all x variables. Since it cannot be a tautology (it appears in the proof, and besides, at the very least it has the variable z), it must have all x variables, and hence has existential width n. ◀

5 Simulations: Preserving Size, Width, and Space Across Calculi

After these strong negative results, ruling out size-width and space-width relations in Q-Res and Q-Res_T, we aim to determine whether any positive results hold in the expansion systems ∀Exp+Res and IR-calc. Before we can do this we need to relate the measures of size, width, and space across the three calculi Q-Res, ∀Exp+Res, IR-calc. Of course, such a comparison in terms of refined simulations is also interesting in its own as it determines the relative strength of the different proof systems. As size corresponds to running time, and space to memory consumption of QBF solvers, such a comparison yields interesting insights into the power of QBF solvers using CDCL vs. expansion techniques.

It is known that IR-calc p-simulates ∀Exp+Res and Q-Res [10], and that ∀Exp+Res p-simulates Q-Res_T [20]. We revisit these proofs, with special attention to the width parameter, and also obtain simulating proofs that are tree-like if the original proof is tree-like. The relationships we establish are stated in the following theorem:

▶ **Theorem 12.** *For all false QBFs $\mathcal{F}$, the following relations hold:*

1. $\frac{1}{2}S(\vdash_{\mathit{IR}_T\text{-}\mathit{calc}} \mathcal{F}) \leq S\left(\vdash_{\forall \mathit{Exp}+\mathit{Res}_T} \mathcal{F}\right) \leq S(\vdash_{\mathit{IR}_T\text{-}\mathit{calc}} \mathcal{F}) \leq 3S(\vdash_{\mathit{Q}\text{-}\mathit{Res}_T} \mathcal{F})$.
2. $w(\vdash_{\mathit{IR}\text{-}\mathit{calc}} \mathcal{F}) = w(\vdash_{\forall \mathit{Exp}+\mathit{Res}} \mathcal{F}) \leq w_\exists(\vdash_{\mathit{Q}\text{-}\mathit{Res}} \mathcal{F})$.
3. $\mathit{CSpace}(\vdash_{\forall \mathit{Exp}+\mathit{Res}_T} \mathcal{F}) = \mathit{CSpace}(\vdash_{\mathit{IR}_T\text{-}\mathit{calc}} \mathcal{F}) \leq \mathit{CSpace}(\vdash_{\mathit{Q}\text{-}\mathit{Res}_T} \mathcal{F})$.

These results follow from Proposition 13 and Lemmas 14, 15 below. Our first simulation of ∀Exp+Res by IR-calc only needs to complete partial annotations in axioms:

▶ **Proposition 13.** *Any proof in ∀Exp+Res of size S, width W, and space C can be efficiently converted into a proof in IR-calc of size at most $2S$, width W, and space C. If the proof in ∀Exp+Res is tree-like, so is the resulting IR-calc proof.*

▶ **Lemma 14.** *$\forall \mathit{Exp}+\mathit{Res}_T$ p-simulates $\mathit{IR}_T\text{-}\mathit{calc}$ while preserving width, size, and space.*

Proof Sketch. The idea is to systematically transform an IR_T-calc proof, proceeding downwards from the top where we have the empty clause, and modifying annotations as we go down, so that when all leaves have been modified the resulting proof is in fact an $\forall\mathsf{Exp}{+}\mathsf{Res}_\mathsf{T}$ proof. This crucially requires that we start with a tree-like proof; if the underlying graph is not a tree, we cannot always find a way of modifying the annotations that will work for all descendants. ◀

The simulation in Lemma 14 exhibits an interesting phenomenon: while it shows that the tree-like versions of ∀Exp+Res and IR-calc are p-equivalent, it was shown in [11] that in the dag-like versions, IR-calc is exponentially stronger than ∀Exp+Res. Thus ∀Exp+Res and IR-calc provide a rare example in proof complexity of two systems that coincide in the tree-like model, but are separated in the dag-like model.

▶ **Lemma 15.** *$\mathit{IR}_T\text{-}\mathit{calc}$ p-simulates $\mathit{Q}\text{-}\mathit{Res}_T$ while preserving space and existential width exactly and size up to a factor of 3.*

Proof Sketch. We use the same simulation as given in [10]. This simulation was originally for dag-like proof systems, but here we check that it also works for tree-like systems and observe that space and existential width are preserved. ◀

As a by-product, these simulations enable us to give an easy and elementary proof of the simulation of $\mathsf{Q}\text{-}\mathsf{Res}_\mathsf{T}$ by ∀Exp+Res, shown in [20] via a more involved argument.

▶ **Corollary 16** (Janota, Marques-Silva [20])**.** *$\forall \mathit{Exp}+\mathit{Res}_T$ p-simulates $\mathit{Q}\text{-}\mathit{Res}_T$.*

Using again the width lower bound for QPARITY_n (Theorem 10) we can show that item 2 of Theorem 12 cannot be improved, i.e. we obtain an optimal width separation between Q-Res and ∀Exp+Res.

▶ **Theorem 17.** *$w_\exists(\vdash_{\mathit{Q}\text{-}\mathit{Res}} \text{QPARITY}_n) = \Omega(n)$, but $w(\vdash_{\forall \mathit{Exp}+\mathit{Res}} \text{QPARITY}_n) = O(1)$.*

Proof. By Theorem 10, QPARITY_n requires existential width n in Q-Res. To get the separation it remains to show $w(\vdash_{\forall\text{Exp}+\text{Res}} \text{QPARITY}_n) = O(1)$. For this we use the following ∀Exp+Res proofs of QPARITY_n from [11]: the formulas QPARITY_n have exactly one universal variable z, which we expand in both polarities 0 and 1. This does not affect the x_i variables, but creates different copies $t_i^{z/0}$ and $t_i^{z/1}$ of the existential variables right of z. Using the clauses of $\text{xor}(t_{i-1}, x_i, t_i)$, we can inductively derive clauses representing $t_i^{z/0} = t_i^{z/1}$. This lets us derive a contradiction using the clauses $t_n^{z/0}$ and $\neg t_n^{z/1}$.

Clearly, this proof only contains clauses of constant width, giving the result. ◀

6 Positive Results: Size, Width, and Space in Tree-like QBF Calculi

We are now in a position to show positive results on size-width and size-space relations for QBF resolution calculi. However, most of these results only apply to weak tree-like systems.

6.1 Relations in the Expansion Calculi ∀Exp+Res and IR-calc

We first observe that for ∀Exp+Res almost the full spectrum of relations from classical resolution remains valid.

▶ **Theorem 18.** *For all false QBFs $\mathcal{F}$, the following relations hold:*

1. $S\left(\vdash_{\forall Exp+Res_T} \mathcal{F}\right) \geq 2^{w\left(\vdash_{\forall Exp+Res} \mathcal{F}\right) - w_\exists(\mathcal{F})}$.
2. $S\left(\vdash_{\forall Exp+Res_T} \mathcal{F}\right) \geq 2^{CSpace\left(\vdash_{\forall Exp+Res_T} \mathcal{F}\right)} - 1$.
3. $CSpace\left(\vdash_{\forall Exp+Res_T} \mathcal{F}\right) \geq CSpace(\vdash_{\forall Exp+Res} \mathcal{F}) \geq w(\vdash_{\forall Exp+Res} \mathcal{F}) - w_\exists(\mathcal{F}) + 1$.

Proof Sketch. Proofs in ∀Exp+Res first download the axioms, leading to clauses containing only annotated existential literals. After that only classical resolution steps are performed and Theorems 1, 2, and 3 can be applied. ◀

By the equivalence of ∀Exp+Res$_\mathsf{T}$ and IR$_\mathsf{T}$-calc with respect to all three measures size, width, and space (Theorem 12) we can transfer all results from Theorem 18 to IR$_\mathsf{T}$-calc.

▶ **Theorem 19.** *For all false QBFs $\mathcal{F}$, the following relations hold:*

1. $S(\vdash_{IR_T\text{-}calc} \mathcal{F}) \geq 2^{w\left(\vdash_{IR\text{-}calc} \mathcal{F}\right) - w_\exists(\mathcal{F})}$.
2. $S(\vdash_{IR_T\text{-}calc} \mathcal{F}) \geq 2^{CSpace\left(\vdash_{IR_T\text{-}calc} \mathcal{F}\right)} - 1$.
3. $CSpace(\vdash_{IR_T\text{-}calc} \mathcal{F}) \geq w(\vdash_{IR\text{-}calc} \mathcal{F}) - w_\exists(\mathcal{F}) + 1$.

6.2 The Size-Space Relation in Tree-like Q-resolution

We finally return to Q-Res. Most relations were already ruled out in Section 4 for both Q-Res and Q-Res$_\mathsf{T}$. The only relation that we can still show to hold is the classical size-space relation (Theorem 2), which we lift from Res$_\mathsf{T}$ to Q-Res$_\mathsf{T}$.

In classical resolution, this relationship was obtained using pebbling games [18]. We observe that the same approach works for Q-Res$_\mathsf{T}$ as well, giving the analogous relationship.

▶ **Theorem 20.** *For a false QBF sentence $\mathcal{F}$, $S(\vdash_{Q\text{-}Res_T} \mathcal{F}) \geq 2^{CSpace\left(\vdash_{Q\text{-}Res_T} \mathcal{F}\right)} - 1$.*

7 Conclusion

Our results show that the success story of width in resolution needs to be rethought when moving to QBF. Indeed, the question arises: is width a central parameter in QBF resolution? Is there another parameter that plays a similar role as classical width for understanding QBF resolution size and space?

Our findings almost completely uncover the picture for size, space, and width for the most basic and arguably most important QBF resolution systems Q-Res, ∀Exp+Res, and IR-calc. The most immediate open question arising from our investigation is whether size-width relations hold for general dag-like ∀Exp+Res or IR-calc proofs. The issue here is that in the classical size-width relation of [7] the number of variables enters the formula in a crucial way. For the instantiation calculi it is not clear what should qualify as the right count for

this as different annotations of the same existential variable are formally treated as distinct variables (which is also clearly justified by the semantic meaning of expansions).

For further research it will also be interesting whether size-width or space-width relations apply to any of the stronger QBF resolution systems QU-Res [27], LD-Q-Res [2], or IRM-calc [10]. However, we conjecture that the negative picture also prevails for these systems.

Acknowledgements. This work was supported by the EU Marie Curie IRSES grant CORCON, grant no. 48138 from the John Templeton Foundation, EPSRC grant EP/L024233/1, and a Doctoral Training Grant from EPSRC (2nd author).

References

1 Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. *Journal of Computer and System Sciences*, 74(3):323–334, 2008.

2 Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF certification and its applications. *Form. Methods Syst. Des.*, 41(1):45–65, August 2012.

3 Valeriy Balabanov, Magdalena Widl, and Jie-Hong R. Jiang. QBF resolution systems and their proof complexities. In *Proc. Theory and Applications of Satisfiability Testing (SAT'14)*, pages 154–169, 2014.

4 Paul Beame, Christopher Beck, and Russell Impagliazzo. Time-space tradeoffs in resolution: superpolynomial lower bounds for superlinear space. In *Proc. ACM Symposium on Theory of Computing (STOC'12)*, pages 213–232, 2012.

5 Eli Ben-Sasson. Size space tradeoffs for resolution. In *Proc. Annual ACM Symposium on Theory of Computing (STOC'02)*, pages 457–464, 2002.

6 Eli Ben-Sasson and Jakob Nordström. Understanding space in proof complexity: Separations and trade-offs via substitutions. In *Proc. Innovations in Computer Science (ICS'11)*, pages 401–416, 2011.

7 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow – resolution made simple. *Journal of the ACM*, 48(2):149–169, 2001.

8 Marco Benedetti and Hratch Mangassarian. QBF-based formal verification: Experience and perspectives. *JSAT*, 5(1-4):133–191, 2008.

9 Olaf Beyersdorff, Ilario Bonacina, and Leroy Chew. Lower bounds: from circuits to QBF proof systems. In *Proc. Innovations in Theoretical Computer Science (ITCS'16)*, 2016.

10 Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. On unification of QBF resolution-based calculi. In *Proc. Mathematical Foundations of Computer Science (MFCS'14)*, pages 81–93, 2014.

11 Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. Proof complexity of resolution-based QBF calculi. In *Proc. Symposium on Theoretical Aspects of Computer Science(STACS'15)*, pages 76–89, 2015.

12 Olaf Beyersdorff, Leroy Chew, Meena Mahajan, and Anil Shukla. Feasible interpolation for QBF resolution calculi. In *Proc. Automata, Languages, and Programming – 42nd International Colloquium (ICALP'15)*, pages 180–192, 2015.

13 Olaf Beyersdorff, Leroy Chew, and Karteek Sreenivasaiah. A game characterisation of tree-like Q-resolution size. In *Proc. International Conference on Language and Automata Theory and Applications(LATA'15)*, pages 486–498, 2015.

14 Olaf Beyersdorff and Oliver Kullmann. Unified characterisations of resolution hardness measures. In *Proc. Theory and Applications of Satisfiability Testing (SAT'14)*, pages 170–187, 2014.

15 Archie Blake. *Canonical expressions in boolean algebra.* PhD thesis, University of Chicago, 1937.

16 Maria Luisa Bonet and Nicola Galesi. A study of proof search algorithms for resolution and polynomial calculus. In *Proc. Annual Symposium on Foundations of Computer Science (FOCS'99)*, 1999.

17 Uwe Egly, Martin Kronegger, Florian Lonsing, and Andreas Pfandler. Conformant planning as a case study of incremental QBF solving. In *Proc. Artificial Intelligence and Symbolic Computation (AISC'14)*, pages 120–131, 2014.

18 Juan Luis Esteban and Jacobo Torán. Space bounds for resolution. *Information and Computation*, 171(1):84–97, 2001.

19 Yuval Filmus, Massimo Lauria, Mladen Miksa, Jakob Nordström, and Marc Vinyals. From small space to small width in resolution. In *International Symposium on Theoretical Aspects of Computer Science (STACS'14)*, pages 300–311, 2014.

20 Mikolás Janota and Joao Marques-Silva. Expansion-based QBF solving versus Q-resolution. *Theor. Comput. Sci.*, 577:25–42, 2015.

21 Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified Boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995.

22 Jan Krajíček. Interpolation theorems, lower bounds for proof systems and independence results for bounded arithmetic. *Journal of Symbolic Logic*, 62(2):457–486, 1997.

23 Oliver Kullmann. Investigating a general hierarchy of polynomially decidable classes of CNF's based on short tree-like resolution proofs. *Electronic Colloquium on Computational Complexity (ECCC)*, 99(41), 1999.

24 Jakob Nordström. Pebble games, proof complexity, and time-space trade-offs. *Logical Methods in Computer Science*, 9(3), 2013.

25 Jussi Rintanen. Asymptotically optimal encodings of conformant planning in QBF. In *AAAI*, pages 1045–1050. AAAI Press, 2007.

26 John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

27 Allen Van Gelder. Contributions to the theory of practical quantified Boolean formula solving. In *Proc. Principles and Practice of Constraint Programming (CP'12)*, pages 647–663, 2012.

28 Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *Proc. IEEE/ACM International Conference on Computer-aided Design (ICCAD'02)*, pages 442–449, 2002.

Faster Algorithms for the Constrained k-Means Problem

Anup Bhattacharya[1], Ragesh Jaiswal[2], and Amit Kumar[3]

1 **Department of Computer Science and Engineering, Indian Institute of Technology Delhi, New Delhi, India**
anupb@cse.iitd.ernet.in

2 **Department of Computer Science and Engineering, Indian Institute of Technology Delhi, New Delhi, India**
rjaiswal@cse.iitd.ernet.in

3 **Department of Computer Science and Engineering, Indian Institute of Technology Delhi, New Delhi, India**
amitk@cse.iitd.ernet.in

Abstract

The classical center based clustering problems such as k-means/median/center assume that the optimal clusters satisfy the locality property that the points in the same cluster are close to each other. A number of clustering problems arise in machine learning where the optimal clusters do not follow such a locality property. For instance, consider the *r-gather clustering problem* where there is an additional constraint that each of the clusters should have at least r points or the *capacitated clustering problem* where there is an upper bound on the cluster sizes. Consider a variant of the k-means problem that may be regarded as a general version of such problems. Here, the optimal clusters $O_1, ..., O_k$ are an *arbitrary* partition of the dataset and the goal is to output k-centers $c_1, ..., c_k$ such that the objective function $\sum_{i=1}^{k} \sum_{x \in O_i} ||x - c_i||^2$ is minimized. It is not difficult to argue that any algorithm (without knowing the optimal clusters) that outputs a single set of k centers, will not behave well as far as optimizing the above objective function is concerned. However, this does not rule out the existence of algorithms that output a *list* of such k centers such that at least one of these k centers behaves well. Given an error parameter $\varepsilon > 0$, let ℓ denote the size of the smallest list of k-centers such that at least one of the k-centers gives a $(1 + \varepsilon)$ approximation w.r.t. the objective function above. In this paper, we show an upper bound on ℓ by giving a randomized algorithm that outputs a list of $2^{\tilde{O}(k/\varepsilon)}$ k-centers. We also give a closely matching lower bound of $2^{\tilde{\Omega}(k/\sqrt{\varepsilon})}$. Moreover, our algorithm runs in time $O\left(nd \cdot 2^{\tilde{O}(k/\varepsilon)}\right)$. This is a significant improvement over the previous result of Ding and Xu who gave an algorithm with running time $O\left(nd \cdot (\log n)^k \cdot 2^{poly(k/\varepsilon)}\right)$ and output a list of size $O\left((\log n)^k \cdot 2^{poly(k/\varepsilon)}\right)$. Our techniques generalize for the k-median problem and for many other settings where non-Euclidean distance measures are involved.

1998 ACM Subject Classification I.5.3 Clustering

Keywords and phrases k-means, k-median, approximation algorithm, sampling

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.16

1 Introduction

Clustering problems intend to classify high dimensional data based on the proximity of points to each other. There is an inherent assumption that the clusters satisfy *locality* property – points close to each other (in a geometric sense) should belong to the same category. Often,

© A. Bhattacharya, R. Jaiswal, and A. Kumar;
licensed under Creative Commons License CC-BY

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 16; pp. 16:1–16:13

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

we model such problems by the notion of a center based clustering problem. We would like to identify a set of centers, one for each cluster, and then the clustering is obtained by assigning each point to the nearest center. For example, the k-means problem is defined in the following manner: given a dataset $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ and an integer k, output a set of k centers $\{c_1, \ldots, c_k\} \subset \mathbb{R}^d$ such that the objective function $\sum_{x \in X} \min_{c \in \{c_1, \ldots, c_k\}} ||x - c||^2$ is minimized. The k-median and the k-center problems are defined in a similar manner by defining a suitable objective function.

However, often such clustering problems entail several *side constraints*. Such constraints limit the set of feasible clusterings. For example, the r-gather k-means clustering problem is defined in the same manner as the k-means problem, but has the additional constraint that each cluster must have at least r points in it. In such settings, it is no longer true that the clustering is obtained from the set of centers by the Voronoi partition. Ding and Xu [5] began a systematic study of such problems, and this is the starting point of our work as well. They defined the so-called *constrained k-means* problem. An instance of such a problem is specified by a set of points X, a parameter k, and a set $\mathbb{C}$, where each element of $\mathbb{C}$ is a partitioning of X into k disjoint subsets (or clusters). Since the set $\mathbb{C}$ may be exponentially large, we will assume that it is specified in a succinct manner by an efficient algorithm which decides membership in this set. A solution needs to output an element $\mathbb{O} = \{O_1, \ldots, O_k\}$ of $\mathbb{C}$, and a set of k centers, $c_1, \ldots, c_k$, one for each cluster in $\mathbb{O}$. The goal is to minimize $\sum_{i=1}^{k} \sum_{x \in O_i} ||x - c_i||^2$. It is easy to check that the center c_i must be the mean of the corresponding cluster O_i. Note that the k-means problem is a special case of this problem where the set $\mathbb{C}$ contains all possible ways of partitioning X into k subsets. The constrained k-median problem can be defined similarly. We will make the natural assumption (which is made by Ding and Xu as well) that it suffices to find a set of k centers. In other words, there is an (efficient) algorithm $A^{\mathbb{C}}$, which given a set of k centers $c_1, \ldots, c_k$, outputs the clustering $\{O_1, \ldots, O_k\} \in \mathbb{C}$ such that $\sum_{i=1}^{k} \sum_{x \in O_i} ||c_i - x||^2$ is minimized. Such an algorithm is called a *partition algorithm* by Ding and Xu [5] [1]. For the case of the k-means problem, this algorithm will just give the Voronoi partition with respect to $c_1, \ldots, c_k$, whereas in the case of the r-gather k-means clustering problem, the algorithm $A^{\mathbb{C}}$ will be given by a suitable min-cost flow computation (see section 4.1 in [5]).

Ding and Xu [5] considered several natural problems arising in diverse areas, e.g. machine learning, which can be stated in this framework. These included the so-called r-gather k-means, r-capacity k-means and l-diversity k-means problems. Their approach for solving such problems was to output a list of candidate sets of centers (of size k) such that at least one of these were close to the optimal centers. We formalize this approach and show that if k is small, then one can obtain a PTAS for the constrained k-means (and the constrained k-median) problems whose running time is linear plus a constant number of calls to $A^{\mathbb{C}}$.

We define the *list k-means* problem. Given a set of points X and parameters k and ε, we want to output a list $\mathcal{L}$ of sets of k points (or centers). The list $\mathcal{L}$ should have the following property: for *any* partitioning $\mathbb{O} = \{O_1, \ldots, O_k\}$ of X into k clusters, there exists a set $c_1, \ldots, c_k$ in the list $\mathcal{L}$ such that (up-to reordering of these centers)

$$\sum_{i=1}^{k} \sum_{x \in O_i} ||c_i - x||^2 \leq (1 + \varepsilon) \sum_{i=1}^{k} \sum_{x \in O_i} ||x - m_i||^2, \tag{1}$$

[1] [5] also gave a discussion on such partition algorithms for a number of clustering problems with side constraints.

where $m_i = \frac{\sum_{x \in O_i} x}{|O_i|}$ denotes the mean of O_i. Note that the latter quantity is the k-means cost of the clustering $\mathbb{O}$, and so we require $c_1, \ldots, c_k$ to be such that the cost of assigning to these centers is close to the optimal k-means cost of this clustering. We shall use $\mathtt{opt}_k(\mathbb{O})$ to denote the optimal k-means cost of $\mathbb{O}$.

Although such an oblivious approach to clustering may appear too optimistic, we show that it is possible to obtain such a list $\mathcal{L}$ of size $2^{\tilde{O}(k/\varepsilon)}$ in $O\left(nd \cdot 2^{\tilde{O}(k/\varepsilon)}\right)$ time [2]. This improves the result of Ding and Xu [5], where they gave an algorithm which outputs a list of size $O\left((\log n)^k \cdot 2^{\text{poly}(k/\varepsilon)}\right)$. Observe that we address a question which is both algorithmic and existential : how small can the size of $\mathcal{L}$ be, and how efficiently can we find it ? We also give almost matching lower bounds on the size of such a list $\mathcal{L}$. Our algorithm for finding $\mathcal{L}$ relies on the D^2-sampling idea – iteratively find the centers by picking the next one to be *far* from the current set of centers. Although these ideas have been used for the k-means problems (see e.g. [9]), they rely heavily on the fact that given a set of centers, the corresponding clustering is obtained by the corresponding Voronoi partition. Our approach relies in showing that there is a small sized list $\mathcal{L}$ which works well for all possible clusterings.

It is not hard to show that a result for the list k-means problem implies a corresponding result for the constrained k-means problem with the number of calls to $A^{\mathbb{C}}$ being equal to the size of the list $\mathcal{L}$. Therefore, we obtain as corollary of our main result efficient algorithms for the constrained k-means (and the constrained k-median) problems.

1.1 Related Work

The classical k-means problem is one of the most well-studied clustering problems. There is a long sequence of work on obtaining fast PTAS for the k-means and the k-median problems (see e.g., [12, 2, 4, 7, 11, 1, 3, 9, 6] and references therein). Some of these works implicitly maintain a list of centers of size k such that the condition (1) is satisfied for all clusterings $\mathbb{O}$ which correspond to a Voronoi partition (with respect to a set of k centers) of the input set of points, and one picks the best possible set of centers from this list (see e.g., [11, 1, 9]). The list has at most $2^{\text{poly}(k/\varepsilon)}$ elements, and from this, one can recover a $(1+\varepsilon)$-approximation algorithm for the k-means problem with running time $O\left(nd \cdot 2^{\text{poly}(k/\varepsilon)}\right)$.

The more general case of the constrained k-means problem was studied by Ding and Xu [5] who also gave an algorithm that outputs a list of size $O\left((\log n)^k \cdot 2^{\text{poly}(k/\varepsilon)}\right)$. Our work improves upon this result. Moreover, we consider the formulation of the list k-means problem as an important contribution, and feel that similar formulations in other classification settings would be useful.

1.2 Preliminaries

We formally define the problems considered in this paper. The centroid or mean of a finite set of points $X \subset \mathbb{R}^d$ is denoted by $\Gamma(X) = \frac{\sum_{x \in X} x}{|X|}$. Let $\Delta(X)$ denote the 1-means cost of these set of points, i.e., $\sum_{x \in X} ||x - \Gamma(X)||^2$.

An input instance $\mathcal{I}$ for the list k-means (or the list k-median) problem consists of a set of points X, a positive integer k and a positive parameter ε. A partition of X into disjoint subsets $O_1, \ldots, O_k$ will be called a *clustering* of X. Given a clustering $\mathbb{O}^\star = \{O_1^\star, \ldots, O_k^\star\}$

[2] $\tilde{O}$ notation hides a $O(\log \frac{k}{\varepsilon})$ factor.

of X and a set of k centers $C = \{c_1, \ldots, c_k\}$, define $\texttt{cost}_C(\mathbb{O}^\star)$ as the minimum, over all permutations π of C, of $\sum_{i=1}^{k} \sum_{x \in O_i^\star} ||x - c_{\pi(i)}||^2$. Recall that $\texttt{opt}_k(\mathbb{O}^\star)$ denotes the optimal k-means cost of $\mathbb{O}^\star$, i.e., $\sum_{i=1}^{k} \sum_{x \in O_i^\star} ||x - \Gamma(O_i^\star)||^2$.

For a set of points X and a set of points C (of size at most k), define $\Phi_C(X)$ as $\sum_{x \in X} \min_{c \in C} ||x - c||^2$, i.e., we consider the Voronoi partition of X induced by C, and consider the k-means cost of X with respect to this partition. When considering the list k-median problem, we will use the same notation, except that we will consider the Euclidean norm instead of the square of the Euclidean norm. When C is a singleton set $\{c\}$, we shall abuse notation by using $\Phi_c(X)$ instead of $\Phi_{\{c\}}(X)$.

As mentioned in the introduction, the constrained k-means problem is specified by a set of points X, a positive integer k, and a set $\mathbb{C}$ of feasible clusterings of X. Further, we are given an algorithm $A^{\mathbb{C}}$, which given a set of k centers C, outputs the clustering $\mathbb{O}$ in $\mathbb{C}$ which minimizes $\texttt{cost}_C(\mathbb{O})$. The goal is to find a clustering $\mathbb{O} \in \mathbb{C}$ and a set C of size k which minimizes $\texttt{cost}_C(\mathbb{O})$. Note that the centers in C should just be the mean of each cluster in $\mathbb{O}$. On the other hand, if we know C, then we can find the best clustering in $\mathbb{C}$ by calling $A^{\mathbb{C}}$. We use the same notation for the constrained k-median problem.

We now mention a few results which will be used in our analysis. The following fact is well known.

▶ **Fact 1.** *For any $X \subset \mathbb{R}^d$ and $c \in \mathbb{R}^d$ we have $\sum_{x \in X} ||x - c||^2 = \sum_{x \in X} ||x - \Gamma(X)||^2 + |X| \cdot ||c - \Gamma(X)||^2$.*

We next define the notion of D^2-sampling.

▶ **Definition 2** (D^2-sampling)**.** Given a set of points $X \subset \mathbb{R}^d$ and another set of points $C \subset \mathbb{R}^d$, D^2-sampling from X w.r.t. C samples a point $x \in X$ with probability $\frac{\Phi_C(\{x\})}{\Phi_C(X)}$. For the case $C = \emptyset$, D^2-sampling is the same as uniform sampling from X.

The following result of Inaba et al. [8] shows that a constant size random sample is a good enough approximation of a set of points X as far as the 1-means objective is concerned.

▶ **Lemma 3** ([8])**.** *Let S be a set of points obtained by independently sampling M points with replacement uniformly at random from a point set $X \subset \mathbb{R}^d$. Then for any $\delta > 0$,*

$$\mathbf{Pr}\left[\Phi_{\Gamma(S)}(X) \leq \left(1 + \frac{1}{\delta M}\right) \cdot \Delta(X)\right] \geq (1 - \delta).$$

We will also use the following simple fact that may be interpreted as approximate version of the triangle inequality for squared Euclidean distance.

▶ **Fact 4** (Approximate triangle inequality)**.** *For any $x, y, z \in \mathbb{R}^d$, we have $||x - z||^2 \leq 2 \cdot ||x - y||^2 + 2 \cdot ||y - z||^2$.*

1.3 Our Results

We now state our results for the list k-means and the list k-median problems.

▶ **Theorem 5.** *Given a set of n points $X \subset \mathbb{R}^d$, parameters k and ε, there is a randomized algorithm which outputs a list $\mathcal{L}$ of $2^{\tilde{O}(k/\varepsilon)}$ sets of centers of size k such that for any clustering $\mathbb{O}^\star = \{O_1^\star, ..., O_k^\star\}$ of X, the following event happens with probability at least $1/2$: there is a set $C \in \mathcal{L}$ such that*

$$\texttt{cost}_C(\mathbb{O}^\star) \leq (1 + \varepsilon) \cdot \texttt{opt}_k(\mathbb{O}^\star).$$

Moreover, the running time of our algorithm is $O\left(nd \cdot 2^{\tilde{O}(k/\varepsilon)}\right)$. *The same statement holds for the list k-median problem as well, except that the size of the list* $\mathcal{L}$ *becomes* $2^{\tilde{O}(k/\varepsilon^{O(1)})}$ *and the running time of our algorithm becomes* $O\left(nd \cdot 2^{\tilde{O}(k/\varepsilon^{O(1)})}\right)$.

As a corollary of this result we get PTAS for the constrained k-means problem (and similarly for the constrained k-median problem). The proof may be found in the full version of this paper. [3]

▶ **Corollary 6.** *There is a randomized algorithm which given an instance of the constrained k-means problem and parameter* $\varepsilon > 0$, *outputs a solution of cost at most* $(1+\varepsilon)$*-times the optimal cost with probability at least* $1/2$. *Further, the time taken by this algorithm is* $O\left(nd \cdot 2^{\tilde{O}(k/\varepsilon)}\right) + 2^{\tilde{O}(k/\varepsilon)} \cdot T$, *where* T *denotes the time taken by* $A^{\mathbb{C}}$ *on this instance.*

Proof. We use the algorithm in Theorem 5 to get a list $\mathcal{L}$ for this data-set. For each set $C \in \mathcal{L}$, we invoke $A^{\mathbb{C}}$ with C as the set of centers – let $\mathbb{O}(C)$ denote the clustering produced by $A^{\mathbb{C}}$. We output the clustering for which $\texttt{cost}_C(\mathbb{O}(C))$ is minimum. Let $\mathbb{O}^\star$ be the optimal clustering, i.e., the clustering in $\mathbb{C}$ for which $\texttt{opt}_k(\mathbb{O}^\star)$ is minimum. We know that with probability at least $1/2$, there is set $C \in \mathcal{L}$ for which $\texttt{cost}_C(\mathbb{O}^\star) \leq (1+\varepsilon)\texttt{opt}_k(\mathbb{O}^\star)$. Now, the solution produced by our algorithm has cost at most $\texttt{cost}_C(\mathbb{O}(C))$, which by definition of $A^{\mathbb{C}}$, is at most $\texttt{cost}_C(\mathbb{O}^\star)$. ◀

We also give a nearly matching lower bound on the size of $\mathcal{L}$. The following result along with Yao's Lemma shows that one cannot reduce the size of $\mathcal{L}$ to less than $2^{\tilde{\Omega}\left(\frac{k}{\sqrt{\varepsilon}}\right)}$.

▶ **Theorem 7.** *Given a parameter* k *and a small enough positive constant* ε, *there exists a set* X *of points in* $\mathbb{R}^d$ *and a set* $\mathbb{C}$ *of clusterings of* X *such that any list* $\mathcal{L}$ *of* k*-centers with the following property must have size at least* $2^{\tilde{\Omega}\left(\frac{k}{\sqrt{\varepsilon}}\right)}$ *: for at least half of the clusterings* $\mathbb{O} \in \mathbb{C}$, *there exists a set* C *in* $\mathcal{L}$ *such that* $\texttt{cost}_C(\mathbb{O}) \leq (1+\varepsilon)\texttt{opt}_k(\mathbb{O})$.

Our techniques also extend to settings involving many other "approximate" metric spaces (see the discussion in the full version of this paper). Another important observation is that in the lower bound result above, the clusterings in $\mathbb{C}$ correspond to Voronoi partitions of X. This throws light on the previous works [11, 1, 6, 9, 10] as to why the running time of all the algorithms was proportional to $2^{\text{poly}(k/\varepsilon)}$: they were implicitly maintaining a list which satisfied (1) for all Voronoi partitions of X, and therefore, our lower bound result applies to their algorithms as well.

1.4 Our Techniques

Our techniques are based on the idea of D^2-sampling that was used by Jaiswal et al. [9] to give a $(1+\varepsilon)$-approximation algorithm for the k-means problem. Our ideas also have similarities to the ideas of Ding and Xu [5]. We discuss these similarities towards the end of this subsection.

One of the crucial ingredients that is used in most of the $(1+\varepsilon)$-approximation algorithms for k-means is Lemma 3. This result essentially states that given a set of points P, if we are able to uniformly sample $O(1/\varepsilon)$ points from it, then the mean of these sampled points

[3] The full version of this paper may be found on Arxiv. Here is the link: `http://arxiv.org/abs/1504.02564`.

will be a good substitute for the mean of P. Consider an optimal clustering $O_1^\star, \ldots, O_k^\star$ for a set of points X. If we could uniformly sample from each of the clusters $O_i^\star$, then by the argument above, we will be done. The first problem one encounters is that one can only sample from the input set of points, and so, if we sample sufficiently many points from X, we need to somehow distinguish the points which belong to $O_i^\star$ in this sample. This can be dealt with using the following argument: suppose we manage to get a small sample S of points (say of size $O(\text{poly}(k/\varepsilon))$) that contain at least $\Omega(1/\varepsilon)$ points uniformly distributed in $O_i^\star$, then we can try all possible subsets of S of size $O(1/\varepsilon)$ and ensure that at least one of the subsets is a uniform sample of appropriate size from $O_i^\star$. Another issue is – how do we ensure that the sample S has sufficient representation from $O_i^\star$? Uniform sampling from the input X will not work since $|O_i^\star|$ might be really small compared to the size of $|X|$. This is where D^2-sampling plays a crucial role and we discuss this next.

Given a set of points $X \subseteq \mathbb{R}^d$ and candidate centers $c_1, ..., c_i \in \mathbb{R}^d$, D^2-sampling with respect to the centers $c_1, ..., c_i$ samples a point $x \in X$ with probability proportional to $\min_{c \in \{c_1, \ldots, c_i\}} ||x - c||^2$. Note that this process "boosts" the probability of a cluster $O_j^\star$ that has many points far from the set $\{c_1, \ldots, c_i\}$. Therefore, even if a cluster $O_j^\star$ has a small size, we will have a good chance of sampling points from it (if it is far from the current set of centers). However, this nonuniform sampling technique gives rise to another issue. The points being sampled are no longer uniform samples from the optimal clusters. Depending on the current set of centers, different points in a cluster $O_j^\star$ have different probability of getting sampled. This issue is not that grave for the k-means problem where the optimal clusters are Voronoi regions since we can argue that the probabilities are not very different. However, for the constrained k-means problem where the optimal clusters are allowed to be arbitrary partition of the input points, this problem becomes more serious. This can be illustrated using the following example. Suppose we have managed to pick centers $c_1, \ldots, c_i$ that are good (in terms of cluster cost) for the optimal clusters $O_1^\star, \ldots, O_i^\star$. At this point let $O_j^\star$ denote the cluster other than $O_1^\star, \ldots, O_i^\star$, such that a point sampled using D^2 sampling w.r.t. $c_1, \ldots, c_i$ is most likely to be from $O_j^\star$. Suppose we sample a set S of $O(k/\varepsilon)$ points using D^2-sampling. Are we guaranteed (w.h.p.) to have a subset in S that is a uniform sample from $O_j^\star$? The answer is no (actually quite far from it). This is because the optimal clusters may form an arbitrary partition of the data-set and it is possible that most of the points in $O_j^\star$ might be very close to the centers $c_1, \ldots, c_i$. In this case the probability of sampling such points will be close to 0. The way we deal with this scenario is that we consider a multi-set S' that is the union of the set of samples S and $O(1/\varepsilon)$ copies of each of $c_1, \ldots, c_i$. We then argue that all the points in $O_j^\star$ that are far from $c_1, \ldots, c_i$ will have a good chance of being represented in S (and hence in S'). On the other hand, even though the points that are close to one of $c_1, \ldots, c_i$ will not be represented in S (and hence S'), the center (among $c_1, \ldots, c_i$) that is close to these points have good representation in S' and these centers may be regarded as "proxy" for the points in $O_j^\star$.

Ding and Xu [5], instead of using the idea of D^2-sampling, rely on the ideas of Kumar et al. [11] which involves uniform sampling of points and then pruning the data-set by removing the points that are close to centers that are currently being considered. In their work, they also encounter the problem that points from some optimal cluster might be close to the current set of good centers (and hence will be removed before uniform sampling). Ding and Xu [5] deal with this issue using what they call a "simplex lemma". Consider the same scenario as in the previous paragraph. At a very high level, they consider grids inside several simplices defined by the current centers $c_1, \ldots, c_i$ and the sampled points. Using the simplex lemma, they argue that one of the points inside these grids will be a good center for the cluster $O_j^\star$.

We now give an overview of the paper. In Section 2, we give the algorithm for generating the list of sets of centers for an instance of the list k-means problem. The algorithm is analyzed in Section 3. Details about the lower bound construction (Theorem 7 and extensions to the k-median problem, and other distance metric settings, are discussed in the full version.)

2 The Algorithm

Consider an instance of the list k-means problem. Let X denote the set of points, and ε be a positive parameter. The algorithm **List-k-means** is described in Algorithm 1. It maintains a set C of centers, which is initially empty. Each recursive call to the function **Sample-centers** increases the size of C by one. In Step 2 of this function, the algorithm tries out various candidates which can be added to C (to increase its size by 1). First, it builds a multi-set S as follows: it independently samples (with replacement) $O(k/\varepsilon^3)$ points using D^2-sampling from X w.r.t. the set C. Further, it adds $O(1/\varepsilon)$ copies of each of the centers in C to the set S. Having constructed S, we consider all subsets of size $O(1/\varepsilon)$ of S – for each such subset we try adding the mean of this set to C. Thus, each invocation of **Sample-centers** makes multiple recursive calls to itself ($\binom{|S|}{M}$ to be precise). It will be useful to think of the execution of this algorithm as a tree $\mathcal{T}$ of depth k. Each node in the tree can be labeled with a set C – it corresponds to the invocation of **Sample-centers** with this set as C (and i being the depth of this node). The children of a node denote the recursive function calls by the corresponding invocation of **Sample-centers**. Finally, the leaves denote the set of candidate centers produced by the algorithm.

List-k-means(X, k, ε)
- Let $N = \frac{136448 \cdot k}{\varepsilon^3}$, $M = \frac{100}{\varepsilon}$
- Initialize $\mathcal{L}$ to $\emptyset$.
- Repeat 2^k times:
  - Make a call to **Sample-centers**$(X, k, \varepsilon, 0, \{\})$.
- Return $\mathcal{L}$.

Sample-centers$(X, k, \varepsilon, i, C)$
(1) If $(i = k)$ then add C to the set $\mathcal{L}$.
(2) else
  (a) Sample a multi-set S of N points with D^2-sampling (w.r.t. centers C)
  (b) $S' \leftarrow S$
  (c) For all $c \in C$: $S' \leftarrow S' \cup \{M \text{ copies of } c\}$
  (d) For all subsets $T \subset S'$ of size M:
    (i) $C \leftarrow C \cup \{\Gamma(T)\}$.
    (ii) **Sample-centers**$(X, k, \varepsilon, i+1, C)$

Algorithm 1 Algorithm for list k-means.

3 Analysis

In this section we prove Theorem 5 for the list k-means problem. Let $\mathcal{L}$ denote the set of candidate solutions produced by **List-k-means**, where a solution corresponds to a set of centers C of size k. These solutions are output at the leaves of the execution tree $\mathcal{T}$. Fix a

clustering $\mathbb{O}^\star = \{O_1^\star, \ldots, O_k^\star\}$ of X. Recall that a node v at depth i in the execution tree $\mathcal{T}$ corresponds to a set C of size i – call this set C_v. Our proof will argue inductively that for each i, there will be a node v at depth i such that the centers chosen so far in C_v are *good* with respect to a subset of i clusters in $O_1^\star, \ldots, O_k^\star$. We will argue that the following invariant $P(i)$ is maintained during the recursive calls to **Sample-centers**:

***P*(*i*)**: With probability at least $\frac{1}{2^{i-1}}$, there is a node v_i at depth $(i-1)$ in the tree $\mathcal{T}$ and a set of $(i-1)$ distinct clusters $O_{j_1}^\star, O_{j_2}^\star, ..., O_{j_{i-1}}^\star$ such that

$$\forall l \in \{1, ..., i-1\}, \Phi_{c_l}(O_{j_l}^\star) \leq \left(1 + \frac{\varepsilon}{2}\right) \cdot \Delta(O_{j_l}^\star) + \frac{\varepsilon}{2k} \cdot \mathtt{opt}_k(\mathbb{O}^\star), \tag{2}$$

where $c_1, \ldots, c_{i-1}$ are the centers in the set C_{v_i} corresponding to v_i. Recall that $\Delta(O_{j_l}^\star)$ refers to the optimal 1-means cost of $O_{j_l}^\star$.

The proof of the main theorem follows easily from this invariant property – indeed, the statement $P(k)$ holds with probability at least $1/2^k$. Since the algorithm **List-k-means** invokes **Sample-centers** 2^k times, the probability of the statement in $P(k)$ being true in at least one of these invocations is at least a constant. We now prove the invariant by induction on i. The base case for $i = 1$ follows trivially: the vertex v_1 is the root of the tree $\mathcal{T}$ and C_{v_1} is empty. Now assume that $P(i)$ holds for some $i \geq 1$. We will prove that $P(i+1)$ also holds. We first condition on the event in $P(i)$ (which happens with probability at least $\frac{1}{2^{i-1}}$). Let v_i and $O_{j_1}^\star, \ldots, O_{j_{i-1}}^\star$ be as guaranteed by the invariant $P(i)$. Let $C_{v_i} = \{c_1, \ldots, c_{i-1}\}$ (as in the statement $P(i)$). For sake of ease of notation, we assume without loss of generality that the index j_i is i, and we shall use C_i to denote C_{v_i}. Thus, the center c_l corresponds to the cluster $O_l^\star$, $1 \leq l \leq i-1$. Note that for a cluster $O_{i'}^\star, i' \geq i$, $\Phi_{C_i}(O_{i'}^\star)$ is proportional to the probability that a point sampled from X using D^2-sampling w.r.t. C_i comes from the set $O_{i'}^\star$ – let $\bar{i} \in \{i, \ldots, k\}$ be the index i' for which $\Phi_{C_i}(O_{i'}^\star)$ is maximum. We will argue that the invocation of **Sample-centers** corresponding to v_i will try out a point c_i (in Step 2(d)(i)) such that the following property will hold with probability at least 1/2: $\Phi_{c_i}(O_{\bar{i}}^\star) \leq (1 + \varepsilon/2) \cdot \Delta(O_{\bar{i}}^\star) + (\varepsilon/2k) \cdot \mathtt{opt}_k(\mathbb{O}^\star)$. For doing this, we break the analysis into the following two parts. These two parts are discussed in the next two subsections that follow.

Case I. $\left(\frac{\Phi_{C_i}(O_{\bar{i}}^\star)}{\sum_{j=1}^k \Phi_{C_i}(O_j^\star)} < \frac{\varepsilon}{13k}\right)$**:** This captures the scenario where the probability of sampling from any of the uncovered clusters is very small. Note that for the classical k-means problem, this is not an issue because in this case we can argue that the current set of centers C already provides a good approximation for the entire set of data points and we are done. However, for us this is an issue — for example, assuming $i > 2$, it is possible that some of the points in $O_{\bar{i}}^\star$ are close to c_1, whereas the remaining points of this cluster are close to c_2. Still we need to output a center for $O_{\bar{i}}^\star$. In this case we argue that it will be sufficient to output a suitable convex combination of c_1 and c_2.

Case II. $\left(\frac{\Phi_{C_i}(O_{\bar{i}}^\star)}{\sum_{j=1}^k \Phi_{C_i}(O_j^\star)} \geq \frac{\varepsilon}{13k}\right)$**:** In this case, we argue that with good probability we will sample sufficient points from $O_{\bar{i}}^\star$ during Step 2(a) of **Sample-centers**. Further, we will show that a suitable combination of such points along with centers in C_i will be a good center for $O_{\bar{i}}^\star$.

3.1 Case I: $\left(\frac{\Phi_{C_i}(O_i^\star)}{\sum_{j=1}^{k}\Phi_{C_i}(O_j^\star)} < \frac{\varepsilon}{13k}\right)$

In this case we argue that a convex combination of the centers in C_i provides a good approximation to $\Delta(O_i^\star)$. Intuitively, this is because the points in $O_i^\star$ are close to the points in the set C_i. This convex combination is essentially "simulated" by taking $O(1/\varepsilon)$ copies of each of the centers $c_1, ..., c_{i-1}$ in the multi-set S and then trying all possible subsets of size $O(1/\varepsilon)$. The formal analysis follows. First, we note that $\Phi_{C_i}(O_i^\star)$ should be small compared to $\mathtt{opt}_k(\mathbb{O}^\star)$. The proof is deferred to the full version of the paper.

▶ **Lemma 8.** $\Phi_{C_i}(O_i^\star) \leq \frac{\varepsilon}{6k} \cdot \mathtt{opt}_k(\mathbb{O}^\star)$.

For each point $p \in O_i^\star$, let $c(p)$ denote the closest center in C_i. We now define a multi-set $O_i^{'}$ as $\{c(p) : p \in O_i^\star\}$. Note that $O_i^{'}$ is obtained by taking multiple copies of points in C_i. The remaining part of the proof proceeds in two steps. Let $m^\star$ and m' denote the mean of $O_i^\star$ and $O_i^{'}$ respectively. We first show that $m^\star$ and m' are close, and so, assigning all the points of $O_i^\star$ to m' will have cost close to $\Delta(O_i^\star)$. Secondly, we show that if we have a good approximation m'' to m', then assigning all the points of $O_i^\star$ to m'' will also incur small cost (comparable to $\Delta(O_i^\star)$). We now carry out these steps in detail. Observe that

$$\sum_{p \in O_i^\star} ||p - c(p)||^2 = \Phi_{C_i}(O_i^\star). \tag{3}$$

▶ **Lemma 9.** $||m^\star - m'||^2 \leq \frac{\Phi_{C_i}(O_i^\star)}{|O_i^\star|}$.

Proof. Let n denote $|O_i^\star|$. Then,

$$||m^\star - m'||^2 = \frac{1}{n^2}\left|\left|\sum_{p \in O_i^\star}(p - c(p))\right|\right|^2 \leq \frac{1}{n}\sum_{p \in O_i^\star}||p - c(p)||^2 = \frac{\Phi_{C_i}(O_i^\star)}{n},$$

where the second last inequality follows from Cauchy-Schwartz [4]. ◀

Now we show that $\Delta(O_i^\star)$ and $\Delta(O_i^{'})$ are close.

▶ **Lemma 10.** $\Delta(O_i^{'}) \leq 2 \cdot \Phi_{C_i}(O_i^\star) + 2 \cdot \Delta(O_i^\star)$.

Proof. The lemma follows by the following inequalities:

$$\begin{aligned}\Delta(O_i^{'}) &= \sum_{p \in O_i^\star}||c(p) - m'||^2 \overset{\text{Fact 1}}{\leq} \sum_{p \in O_i^\star}||c(p) - m^\star||^2 \\ &\overset{\text{Fact 4}}{\leq} 2 \cdot \sum_{p \in O_i^\star}\left(||c(p) - p||^2 + ||p - m^\star||^2\right) = 2 \cdot \Phi_{C_i}(O_i^\star) + 2 \cdot \Delta(O_i^\star).\end{aligned}$$

This completes the proof of the lemma. ◀

Finally, we argue that a good center for $O_i^{'}$ will also serve as a good center for $O_i^\star$.

▶ **Lemma 11.** *Let m'' be a point such that $\Phi_{m''}(O_i^{'}) \leq \left(1 + \frac{\varepsilon}{8}\right) \cdot \Delta(O_i^{'})$. Then $\Phi_{m''}(O_i^\star) \leq \left(1 + \frac{\varepsilon}{2}\right) \cdot \Delta(O_i^\star) + \frac{\varepsilon}{2k} \cdot \mathtt{opt}_k(\mathbb{O}^\star)$.*

[4] For any real numbers $a_1, ..., a_m$, $(\sum_r a_r)^2/m \leq \sum_r a_r^2$.

Proof. Let $n^\star$ denote $|O_{\tilde{i}}^\star|$. Observe that

$$\begin{aligned}
\Phi_{m''}(O_{\tilde{i}}^\star) &= \sum_{p\in O_{\tilde{i}}^\star} ||m''-p||^2 \\
&\overset{\text{Fact 1}}{=} \sum_{p\in O_{\tilde{i}}^\star} ||m^\star-p||^2 + n^\star\cdot||m^\star-m''||^2 \\
&\overset{\text{Fact 4}}{\leq} \Delta(O_{\tilde{i}}^\star) + 2n^\star\left(||m^\star-m'||^2+||m'-m''||^2\right) \\
&\overset{\text{Lemma 9}}{\leq} \Delta(O_{\tilde{i}}^\star) + 2\cdot\Phi_{C_i}(O_{\tilde{i}}^\star) + 2n^\star||m'-m''||^2 \\
&\overset{\text{Fact 1}}{\leq} \Delta(O_{\tilde{i}}^\star) + 2\cdot\Phi_{C_i}(O_{\tilde{i}}^\star) + 2\left(\Phi_{m''}(O_{\tilde{i}}^{'}) - \Delta(O_{\tilde{i}}^{'})\right) \\
&\leq \Delta(O_{\tilde{i}}^\star) + 2\cdot\Phi_{C_i}(O_{\tilde{i}}^\star) + \frac{\varepsilon}{4}\cdot\Delta(O_{\tilde{i}}^{'}) \\
&\overset{\text{Lemma 10}}{\leq} \Delta(O_{\tilde{i}}^\star) + 2\cdot\Phi_{C_i}(O_{\tilde{i}}^\star) + \frac{\varepsilon}{2}\cdot\left(\Phi_{C_i}(O_{\tilde{i}}^\star)+\Delta(O_{\tilde{i}}^\star)\right) \\
&\overset{\text{Lemma 8}}{\leq} \left(1+\frac{\varepsilon}{2}\right)\cdot\Delta(O_{\tilde{i}}^\star) + \frac{\varepsilon}{2k}\cdot\mathtt{opt}_k(\mathbb{O}^\star)
\end{aligned}$$

This completes the proof of the lemma. ◀

The above lemma tells us that it will be sufficient to obtain a $(1+\varepsilon/8)$-approximation to the 1-means problem for the dataset $O_{\tilde{i}}^{'}$. Now, Lemma 3 tells us that there is a subset (again as a multi-set) O'' of size $\frac{16}{\varepsilon}$ of $O_{\tilde{i}}^{'}$ such that the mean m'' of these points satisfies the conditions of Lemma 11. Now, observe that O'' will be a subset of the set S constructed in Step 2 of the algorithm **Sample-center** – indeed, in Step 2(c), we add more than $\frac{16}{\varepsilon}$ copies of *each* point in C_i to S. Now, in Step 2(d), we will try out all subsets of size $\frac{16}{\varepsilon}$ of S and for each such subset, we will try adding its mean to C_i. In particular, there will be a recursive call of this function, where we will have $C_{i+1} = C_i \cup \{m''\}$ as the set of centers. Lemma 11 now implies that C_{i+1} will satisfy the invariant $P(i+1)$. Thus, we are done in this case.

3.2 Case II: $\left(\frac{\Phi_{C_i}(O_{\tilde{i}}^\star)}{\sum_j \Phi_{C_i}(O_j^\star)} \geq \frac{\varepsilon}{13k}\right)$

In this case, we would like to prove that we add a good approximation to the mean of $O_{\tilde{i}}^\star$ to the set C_i. Again, consider the invocation of **Sample-centers** corresponding to C_i. We want the multi-set S to contain a good representation from points in the set $O_{\tilde{i}}^\star$. Secondly, in order to apply Lemma 3, we will need this representation to be a uniform sample from $O_{\tilde{i}}^\star$. Since $\Phi_{C_i}(O_{\tilde{i}}^\star) \geq \frac{\varepsilon}{13k}\cdot\sum_j \Phi_{C_i}(O_j^\star)$, the probability that a point sampled using D^2 sampling w.r.t. C_i is from $O_{\tilde{i}}^\star$ is not too small. So, the multi-set S will have non-negligible representation from the set $O_{\tilde{i}}^\star$. However the points from $O_{\tilde{i}}^\star$ in S may not be a uniform sample from $O_{\tilde{i}}^\star$. Indeed, suppose there is a good fraction of points of $O_{\tilde{i}}^\star$ which are close to C_i, and remaining points of $O_{\tilde{i}}^\star$ are quite far from C_i. Then, D^2-sampling w.r.t. to C_i will not give us a uniform sample from $O_{\tilde{i}}^\star$. To alleviate this problem, we take sufficiently many copies of points in C_i and add them to the multi-set S. In some sense, these copies act as proxy for points in $O_{\tilde{i}}^\star$ that are too close to C_i. Finally, we argue that one of the subsets of S "simulates" a uniform sample from $O_{\tilde{i}}^\star$ and the mean of this subset provides a good approximation for the mean of $O_{\tilde{i}}^\star$. The formal analysis follows.

We divide the points in $O_i^\star$ into two parts – points which are close to a center in C_i, and the remaining points. More formally, let the radius R be given by

$$R^2 = \frac{\varepsilon^2}{41} \cdot \frac{\Phi_{C_i}(O_i^\star)}{|O_i^\star|} \tag{4}$$

Define O_i^n as the points in $O_i^\star$ which are within distance R of a center in C_i, and O_i^f be the rest of the points in $O_i^\star$. As in Case I, we define a new set $O_i^{'}$ where each point in O_i^n is replaced by a copy of the corresponding point in C_i. For a point $p \in O_i^n$, define $c(p)$ as the closest center in C_i to p. Now define a multi-set $O_i^{'}$ as $O_i^f \cup \{c(p) : p \in O_i^n\}$. Intuitively, $O_i^{'}$ denotes the set of points that are same as $O_i^\star$ except that points close to centers in C_i have been "collapsed" to these centers by taking appropriate number of copies. Clearly, $|O_i^{'}| = |O_i^\star|$. At a high level, we will argue that any center that provides a good 1-means approximation for $O_i^{'}$ also provides a good approximation for $O_i^\star$. We will then focus on analyzing whether the invocation of **Sample-centers** tries out a good center for $O_i^{'}$.

We give some more notation. Let $m^\star$ and m' denote the mean of $O_i^\star$ and $O_i^{'}$ respectively. Let $n^\star$ and n denote the size of the sets $O_i^\star$ and O_i^n respectively. First, we show that $\Delta(O_i^\star)$ is large with respect to R.

▶ **Lemma 12.** $\Delta(O_i^\star) = \Phi_{m^\star}(O_i^\star) \geq \frac{16n}{\varepsilon^2} R^2$.

Proof. Let c be the center in C_i which is closest to $m^\star$. We divide the proof into two cases:

(i) $||m^\star - c|| \geq \frac{5}{\varepsilon} \cdot R$: For any point $p \in O_i^n$, triangle inequality implies that

$$||p - m^\star|| \geq ||c(p) - m^\star|| - ||c(p) - p|| \geq \frac{5}{\varepsilon} \cdot R - R \geq \frac{4}{\varepsilon} \cdot R.$$

Therefore, $\Delta(O_i^\star) \geq \sum_{p \in O_i^n} ||p - m^\star||^2 \geq \frac{16n}{\varepsilon^2} R^2$.

(ii) $||m^\star - c|| < \frac{5}{\varepsilon} \cdot R$: In this case, we have

$$\begin{aligned} \Phi_{m^\star}(O_i^\star) &\overset{\text{Fact 1}}{=} \Phi_c(O_i^\star) - n^\star \cdot ||m^\star - c||^2 \geq \Phi_{C_i}(O_i^\star) - n^\star \cdot ||m^\star - c||^2 \\ &\overset{(4)}{\geq} \frac{41n^\star}{\varepsilon^2} \cdot R^2 - \frac{25n^\star}{\varepsilon^2} \cdot R^2 \geq \frac{16n}{\varepsilon^2} R^2. \end{aligned}$$

This completes the proof of the lemma. ◀

The proofs of the following two lemmas are similar to those of Lemma 9 and Lemma 10 respectively, and are deferred to the full version of the paper.

▶ **Lemma 13.** $||m^\star - m'||^2 \leq \frac{n}{n^\star} \cdot R^2$

▶ **Lemma 14.** $\Delta(O_i^{'}) \leq 4nR^2 + 2 \cdot \Delta(O_i^\star)$.

We now argue that any center that is good for $O_i^{'}$ is also good for $O_i^\star$.

▶ **Lemma 15.** *Let m'' be such that $\Phi_{m''}(O_i^{'}) \leq \left(1 + \frac{\varepsilon}{16}\right) \cdot \Delta(O_i^{'})$. Then $\Phi_{m''}(O_i^\star) \leq \left(1 + \frac{\varepsilon}{2}\right) \cdot \Delta(O_i^\star)$.*

Proof. The lemma follows from the following inequalities:

$$\begin{aligned} \Phi_{m''}(O_i^\star) &= \sum_{p \in O_i^\star} ||m'' - p||^2 \\ &\overset{\text{Fact 1}}{=} \sum_{p \in O_i^\star} ||m^\star - p||^2 + n^\star \cdot ||m^\star - m''||^2 \end{aligned}$$

$$\begin{aligned}
&\overset{\text{Fact 4}}{\leq} && \Delta(O_{\bar{i}}^{\star}) + 2n^{\star}\left(||m^{\star} - m'||^2 + ||m' - m''||^2\right)\\
&\overset{\text{Lemma 13}}{\leq} && \Delta(O_{\bar{i}}^{\star}) + 2nR^2 + 2n^{\star} \cdot ||m' - m''||^2\\
&\overset{\text{Fact 1}}{\leq} && \Delta(O_{\bar{i}}^{\star}) + 2nR^2 + 2 \cdot \left(\Phi_{m''}(O_{\bar{i}}^{'}) - \Delta(O_{\bar{i}}^{'})\right)\\
&\leq && \Delta(O_{\bar{i}}^{\star}) + 2nR^2 + \frac{\varepsilon}{8} \cdot \Delta(O_{\bar{i}}^{'})\\
&\overset{\text{Lemma 14}}{\leq} && \Delta(O_{\bar{i}}^{\star}) + 2nR^2 + \frac{\varepsilon}{2} \cdot nR^2 + \frac{\varepsilon}{4} \cdot \Delta(O_{\bar{i}}^{\star})\\
&\overset{\text{Lemma 12}}{\leq} && \left(1 + \frac{\varepsilon}{2}\right) \cdot \Delta(O_{\bar{i}}^{\star}).
\end{aligned}$$

This completes the proof of the lemma. ◀

Given the above lemma, all we need to argue is that our algorithm indeed considers a center m'' such that $\Phi_{m''}(O_{\bar{i}}^{'}) \leq (1 + \varepsilon/16) \cdot \Delta(O_{\bar{i}}^{'})$. For this we would need about $O(1/\varepsilon)$ uniform samples from $O_{\bar{i}}^{'}$. However, our algorithm can only sample using D^2-sampling w.r.t. C_i. For ease of notation, let $c(O_{\bar{i}}^{n})$ denote the multi-set $\{c(p) : p \in O_{\bar{i}}^{n}\}$. Recall that $O_{\bar{i}}^{'}$ consists of $O_{\bar{i}}^{f}$ and $c(O_{\bar{i}}^{n})$. The first observation is that the probability of sampling an element from $O_{\bar{i}}^{f}$ is reasonably large (proportional to ε/k). Using this fact, we show how to sample from $O_{\bar{i}}^{'}$ (almost uniformly). Finally, we show how to convert this almost uniform sampling to uniform sampling (at the cost of increasing the size of sample). We defer the proof of the following lemma to the full version of the paper.

▶ **Lemma 16.** *Let x be a sample from D^2-sampling w.r.t. C_i. Then, $\mathbf{Pr}[x \in O_{\bar{i}}^{f}] \geq \frac{\varepsilon}{15k}$. Further, for any point $p \in O_{\bar{i}}^{f}$, $\mathbf{Pr}[x = p] \geq \frac{\gamma}{|O_{\bar{i}}^{\star}|}$, where γ denotes $\frac{\varepsilon^2}{533k}$.*

Let $X_1, \ldots X_l$ be l points sampled independently using D^2-sampling w.r.t. C_i. We construct a new set of random variables $Y_1, \ldots, Y_l$. Each variable Y_u will depend on X_u only, and will take values either in $O_{\bar{i}}^{'}$ or will be $\perp$. These variables are defined as follows: if $X_u \notin O_{\bar{i}}^{f}$, we set Y_u to $\perp$. Otherwise, we assign Y_u to one of the following random variables with equal probability: (i) X_u or (ii) a random element of the multi-set $c(O_{\bar{i}}^{n})$. The following observation follows from Lemma 16, and its proof is deferred to the full version of the paper.

▶ **Corollary 17.** *For a fixed index u, and an element $x \in O_{\bar{i}}^{'}$, $\mathbf{Pr}[Y_u = x] \geq \frac{\gamma'}{|O_{\bar{i}}^{'}|}$, where $\gamma' = \gamma/2$.*

Corollary 17 shows that we can obtain samples from $O_{\bar{i}}^{'}$ which are nearly uniform (up to a constant factor). To convert this to a set of uniform samples, we use the idea of [9]. For an element $x \in O_{\bar{i}}^{'}$, let γ_x be such that $\frac{\gamma_x}{|O_{\bar{i}}^{'}|}$ denotes the probability that the random variable Y_u is equal to x (note that this is independent of u). Corollary 17 implies that $\gamma_x \geq \gamma'$. We define a new set of independent random variables $Z_1, \ldots, Z_l$. The random variable Z_u will depend on Y_u only. If Y_u is $\perp$, Z_u is also $\perp$. If Y_u is equal to $x \in O_{\bar{i}}^{'}$, then Z_u takes the value x with probability $\frac{\gamma'}{\gamma_x}$, and $\perp$ with the remaining probability. Note that Z_u is either $\perp$ or one of the elements of $O_{\bar{i}}^{'}$. Further, conditioned on the latter event, it is a uniform sample from $O_{\bar{i}}^{'}$. We can now give the key lemma (proof is deferred to the full version).

▶ **Lemma 18.** *Let l be $\frac{128}{\gamma' \cdot \varepsilon}$, and m'' denote the mean of the non-null samples from $Z_1, \ldots, Z_l$. Then, with probability at least $1/2$, $\Phi_{m''}(O_{\bar{i}}^{'}) \leq (1 + \varepsilon/16) \cdot \Delta(O_{\bar{i}}^{'})$.*

Let $C_i^{(l)}$ denote the multi-set obtained by taking l copies of each of the centers in C_i. Now observe that all the non-$\perp$ elements among $Y_1, \ldots, Y_l$ are elements of $\{X_1, \ldots, X_l\} \cup C_i^{(l)}$, and so the same must hold for $Z_1, \ldots, Z_l$. This implies that in Step 2(d) of the algorithm **Sample-centers**, we would have tried adding the point m'' as described in Lemma 18. Therefore, the induction hypothesis continues to hold with probability at least $1/2$. This concludes the proof of Theorem 5.

References

1. Marcel R. Ackermann, Johannes Blömer, and Christian Sohler. Clustering for metric and nonmetric distance measures. *ACM Trans. Algorithms*, 6:59:1–59:26, September 2010. URL: http://doi.acm.org/10.1145/1824777.1824779.
2. Mihai Bādoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proc. of the 34th Annual ACM Symp. on Theory of Computing*, STOC'02, pages 250–257, New York, NY, USA, 2002. ACM. doi:10.1145/509907.509947.
3. Ke Chen. On k-median clustering in high dimensions. In *Proc. of the 17th Annual ACM-SIAM Symp. on Discrete Algorithm*, SODA'06, pages 1177–1185, New York, NY, USA, 2006. ACM. doi:10.1145/1109557.1109687.
4. W. Fernandez de la Vega, Marek Karpinski, Claire Kenyon, and Yuval Rabani. Approximation schemes for clustering problems. In *Proc. of the 35th Annual ACM Symp. on Theory of Computing*, STOC'03, pages 50–58, New York, NY, USA, 2003. ACM. doi:10.1145/780542.780550.
5. Hu Ding and Jinhui Xu. A unified framework for clustering constrained data without locality property. In *Proc. of the 26th Annual ACM-SIAM Symp. on Discrete Algorithms*, SODA'15, pages 1471–1490, 2015. doi:10.1137/1.9781611973730.97.
6. Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A PTAS for k-means clustering based on weak coresets. In *Proc. of the 23rd Annual Symp. on Computational Geometry*, SoCG'07, pages 11–18, New York, NY, USA, 2007. ACM. doi:10.1145/1247069.1247072.
7. Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proc. of the 36th Annual ACM Symp. on Theory of Computing*, STOC'04, pages 291–300, New York, NY, USA, 2004. ACM. doi:10.1145/1007352.1007400.
8. Mary Inaba, Naoki Katoh, and Hiroshi Imai. Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering: (extended abstract). In *Proc. of the 10th Annual Symp. on Computational Geometry*, SoCG'94, pages 332–339, New York, NY, USA, 1994. ACM. doi:10.1145/177424.178042.
9. Ragesh Jaiswal, Amit Kumar, and Sandeep Sen. A simple D^2-sampling based PTAS for k-means and other clustering problems. *Algorithmica*, 70(1):22–46, 2014. doi:10.1007/s00453-013-9833-9.
10. Ragesh Jaiswal, Mehul Kumar, and Pulkit Yadav. Improved analysis of D^2-sampling based PTAS for k-means and other clustering problems. *Information Processing Letters*, 115(2):100–103, 2015. doi:http://dx.doi.org/10.1016/j.ipl.2014.07.009.
11. Amit Kumar, Yogish Sabharwal, and Sandeep Sen. Linear-time approximation schemes for clustering problems in any dimensions. *J. ACM*, 57(2):5:1–5:32, February 2010. doi:10.1145/1667053.1667054.
12. J. Matoušek. On approximate geometric k-clustering. *Discrete and Computational Geometry*, 24(1):61–84, 2000. doi:10.1007/s004540010019.

A Catalog of $\exists\mathbb{R}$-Complete Decision Problems About Nash Equilibria in Multi-Player Games

Vittorio Bilò[1] **and Marios Mavronicolas**[2]

1 **Department of Mathematics and Physics, University of Salento, 73100 Lecce, Italy**
vittorio.bilo@unisalento.it

2 **Department of Computer Science, University of Cyprus, CY-1678 Nicosia, Cyprus**
mavronic@cs.ucy.ac.cy

Abstract

[Schaefer and Štefankovič, *Theory of Computing Systems, 2015*] provided an explicit formulation of $\exists\mathbb{R}$ as the class capturing the complexity of deciding the *Existential Theory of the Reals,* and established that deciding, given a 3-player *game,* whether or not it has a *Nash equilibrium* with no probability exceeding a given rational is $\exists\mathbb{R}$-complete. Four more decision problems about Nash equilibria for 3-player games were very recently shown $\exists\mathbb{R}$-complete via a chain of individual, problem-specific reductions in [Garg et al., *Proceedings of ICALP* 2015]; determining more such $\exists\mathbb{R}$-complete problems was posed there as an open problem. In this work, we deliver an extensive catalog of $\exists\mathbb{R}$-complete decision problems about Nash equilibria in 3-player games, thus resolving completely the open problem from [Garg et al., *Proceedings of ICALP* 2015]. Towards this end, we present a single and very simple, unifying reduction from the $\exists\mathbb{R}$-complete decision problem from [Schaefer and Štefankovič, *Theory of Computing Systems, 2015*] to (almost) *all* the decision problems about Nash equilibria that were before shown $\mathcal{NP}$-complete for 2-player games in [Bilò and Mavronicolas, *Proceedings of SAGT* 2012; Conitzer and Sandholm, *Games and Economic Behavior*, 2008; Gilboa and Zemel, *Games and Economic Behavior*, 1989]. Encompassed in the catalog are the four decision problems shown $\exists\mathbb{R}$-complete in [Garg et al., *Proceedings of ICALP* 2015].

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Nash equilibrium, complexity of equilibria, $\exists\mathbb{R}$-completeness

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.17

1 Introduction

1.1 Framework, Motivation and Contribution

The *Existential Theory of the Reals,* denoted as ETR, is the set of existential first-order sentences over the real numbers. In 1948, Alfred Tarski used his method of quantifier elimination [18] to show that the entire *First Order Theory of the Reals,* encompassing ETR, is decidable, albeit without an elementary bound on its complexity. To date the best known upper bound to decide ETR is $\mathcal{PSPACE}$, coming from the seminal work of Canny [4].Many geometric, graph-drawing and topological problems have been recognized to have the same complexity as ETR. Some of them concern recognizing *intersection graphs* of a certain type – see, e.g., [15, 16]; others concern deciding the stretchability of pseudolines [11]: given a family of plane curves, are they homeomorphic to a line arrangement? Based on these, the

© Vittorio Bilò and Marios Mavronicolas;
licensed under Creative Commons License CC-BY

complexity class $\exists\mathbb{R}$ was defined by Schaefer and Štefankovič [17] as the set of problems with a polynomial-time, many-to-one reduction to ETR. Decision variants of fixed-point problems, including the *Brouwer fixed-point* problem and *Nash equilibria,* were shown $\exists\mathbb{R}$-complete in [17]; Nash equilibrium [12, 13] is undoubtedly the most influential solution concept in Game Theory, representing a state of a *game* where no *player* could unilaterally switch her *strategy* to improve her *payoff*. Both *search* and *decision problems* about Nash equilibria have been studied extensively in *Algorithmic Game Theory*; by the seminal results in [5, 7], their search problem is $\mathcal{PPAD}$-complete [14] even for 2-player games.

More specifically, Schaefer and Štefankovič [17, Corollary 3.5] identified the *first* $\exists\mathbb{R}$-complete decision problem about Nash equilibria in multi-player games: this is $\exists$ NASH IN A BALL, which asks, given an *r-player game* with $r \geq 3$ and a rational ϱ, whether or not it has a Nash equilibrium with no probability exceeding ϱ. The proof employed a reduction from BROUWER, another decision problem shown $\exists\mathbb{R}$-complete in [17], which asks whether or not a function, represented by a given straight-line program, has a fixed point in a specified ball. Very recently, Garg *et al.* [9] used a chain of *problem-specific* reductions, starting from $\exists$ NASH IN A BALL [17], to prove that four among the $\mathcal{NP}$-complete problems for 2-player games [1, 6, 10] are $\exists\mathbb{R}$-complete for r-player games with $r \geq 3$. Garg *et al.* [9, Appendix H] posed as an open problem the enlargement of the class of such $\exists\mathbb{R}$-complete problems.

A full story is known for decision problems about Nash equilibria for 2-player games; they are $\mathcal{NP}$-complete; see [1, 6, 10] for an extensive catalog. Their membership in $\mathcal{NP}$ is due to the fact that the Nash equilibria for a 2-player game involve rational probabilities; this allows, given the supports, polynomial time verification of the Nash equilibrium property. This is no longer the case for r-player games with $r \geq 3$, which may have Nash equilibria with irrational probabilities. Hence, these decision problems are only known to be $\mathcal{NP}$-hard over r-player games with $r \geq 3$, and their precise complexity characterization has remained elusive. (Two notable exceptions are the problems of deciding the existence of a *rational* Nash equilibrium [2] and a *uniform* Nash equilibrium [3], which belong to $\mathcal{NP}$ for r-player games with $r \geq 3$, and this finalizes their complexity classification.) *In this work, we show that they are (almost) all $\exists\mathbb{R}$-complete, delivering an extended catalog of $\exists\mathbb{R}$-complete decision problems about Nash equilibria for r-player games with $r \geq 3$* (Theorem 10).

1.2 Techniques and Significance

We employ a *game reduction* (Section 3) that maps, given an arbitrary number $\delta > 0$, a pair of 3-player games $\widetilde{\mathsf{G}}$ and $\widehat{\mathsf{G}}$, called the *subgames,* to a 3-player game G with a larger set of strategies for each player; both games $\widetilde{\mathsf{G}}$ (with δ added to each utility) are "embedded" in G as subgames. The reduction guarantees certain correspondences between the Nash equilibria for $\widetilde{\mathsf{G}}$ and $\widehat{\mathsf{G}}$, respectively, and those for G. Specifically, a Nash equilibrium for G subsumes either a Nash equilibrium for $\widetilde{\mathsf{G}}$ or one for $\widehat{\mathsf{G}}$ (Lemma 7); in the other direction, a Nash equilibrium for $\widehat{\mathsf{G}}$ always induces one for G (Lemma 8); but a Nash equilibrium for $\widetilde{\mathsf{G}}$ induces one for G if and only if none of its probabilities exceeds $\frac{1}{2}$ (Lemma 9).

We proceed to embed the game reduction into a polynomial time many-to-one reduction from $\exists$ NASH IN A BALL to a catalog of decision problems about Nash equilibria for r-player games with $r \geq 3$, thus establishing their $\exists\mathbb{R}$-hardness (Section 4). We are given an instance $\widetilde{\mathsf{G}}$ of $\exists$ NASH IN A BALL, called the *inbox game*. We construct a game $\widehat{\mathsf{G}}$, called the *gadget game,* which may depend on $\widetilde{\mathsf{G}}$. Finally, we apply the game reduction on $\widetilde{\mathsf{G}}$ and $\widehat{\mathsf{G}}$ to get the game G. The correspondences between the Nash equilibria for $\widetilde{\mathsf{G}}$ and $\widehat{\mathsf{G}}$, respectively, and those for G are used to deduce the properties of the Nash equilibria for G, which are found to depend on whether or not the inbox game $\widetilde{\mathsf{G}}$ is a positive instance for $\exists$ NASH IN A

BALL. The established equivalence between $\widetilde{\mathsf{G}}$ being a positive instance for $\exists$ NASH IN A BALL and the induced properties of G imply the $\exists\mathbb{R}$-hardness of the properties. The *single, unifying reduction* we employ to establish the $\exists\mathbb{R}$-hardness of all decision problems in the catalog (Sections 3 and 4) is extremely simple, as well as its corresponding proof; thus, it simplifies tremendously the corresponding chain of (problem-specific) reductions in [9], which had involved proofs but only yielded *four* $\exists\mathbb{R}$-hard problems, which are encompassed in the catalog we present. The catalog includes (almost) *all* the decision problems about Nash equilibria for 2-player games shown $\mathcal{NP}$-complete in [1, 6, 10].

2 Background and Preliminaries

2.1 The Class $\exists\mathbb{R}$

The *Existential Theory of the Reals,* denoted as ETR, is the set of true sentences of the form $(\exists x_1, \ldots, x_n)(\varphi(x_1, \ldots, x_n)$, where φ is a quantifier-free $(\vee, \wedge, \neg)$-boolean formula over the signature $(0, 1, +, *, <, \leq, =)$ interpreted over the real numbers. $\exists\mathbb{R}$ is the complexity class associated with ETR: A decision problem *belongs to* $\exists\mathbb{R}$ if there is a polynomial-time, many-to-one reduction from it to ETR, and it is $\exists\mathbb{R}$*-hard* if there is a polynomial-time many-to-one reduction from each problem in $\exists\mathbb{R}$ to it; it is $\exists\mathbb{R}$*-complete* if it belongs to $\exists\mathbb{R}$ and it is $\exists\mathbb{R}$-hard. Since satisfiability of a propositional boolean formula (SAT) can be expressed in ETR, $\mathcal{NP} \subseteq \exists\mathbb{R}$; so, ETR is for $\exists\mathbb{R}$ what SAT is for $\mathcal{NP}$. Thus, an $\exists\mathbb{R}$-complete problem is decided in $\mathcal{NP}$ if and only if ETR in $\mathcal{NP}$. By Canny's result [4], $\exists\mathbb{R} \subseteq \mathcal{PSPACE}$. We refer the reader to [17] for more background on the class $\exists\mathbb{R}$.

2.2 Games and Nash Equilibria

A *game* is a triple $\mathsf{G} = \left\langle [r], \{\Sigma_i\}_{i\in[r]}, \{\mathsf{U}_i\}_{i\in[r]} \right\rangle$, where *(i)* $[r] = \{1, \ldots, r\}$ is a finite set of *players* with $r \geq 2$, and *(ii)* for each player $i \in [r]$, Σ_i is the set of *strategies* for player i, and U_i is the *payoff function* $\mathsf{U}_i : \times_{k\in[r]}\Sigma_k \to \mathbb{R}$ for player i. Denote as $\underline{u}(\mathsf{G}) = \min_{\mathbf{s}\in\Sigma, i\in[r]} \{\mathsf{U}_i(\mathbf{s})\}$ the *minimum payoff* for G. The game G is *win-lose* if for each player $i \in [r]$, U_i is a function $\mathsf{U}_i : \times_{k\in[r]}\Sigma_k \to \{0, 1\}$. For each player $i \in [r]$, denote $\Sigma_{-i} = \times_{k\in[r]\setminus\{i\}}\Sigma_k$; denote $\Sigma = \times_{k\in[r]}\Sigma_k$. A *profile* is a tuple $\mathbf{s} \in \Sigma$ of r strategies, one per player. The vector $\mathsf{U}(\mathbf{s}) = \langle \mathsf{U}_1(\mathbf{s}), \ldots, \mathsf{U}_r(\mathbf{s})\rangle$ is the *payoff vector* for $\mathbf{s}$. A *partial profile* $\mathbf{s}_{-i}$ is a tuple of $r-1$ strategies, one for each player other than i; so $\mathbf{s}_{-i} \in \Sigma_{-i}$. For a profile $\mathbf{s}$ and a strategy $t \in \Sigma_i$, denote as $\mathbf{s}_{-i} \diamond t$ the profile obtained by substituting strategy t for s_i in $\mathbf{s}$. The game G has the *positive payoff property* [1] if for each player $i \in [r]$ and each partial profile $\mathbf{s}_{-i} \in \Sigma_{-i}$, there is a strategy $t = t(\mathbf{s}_{-i})$ such that $\mathsf{U}_i(\mathbf{s}_{-i} \diamond t) > 0$.

A *mixed strategy* for player $i \in [r]$ is a probability distribution σ_i on her strategy set Σ_i: a function $\sigma_i : \Sigma_i \to [0, 1]$ such that $\sum_{s\in\Sigma_i} \sigma_i(s) = 1$. Denote as $\mathsf{Support}(\sigma_i)$ the set of strategies $s \in \Sigma_i$ with $\sigma_i(s) > 0$. A *mixed profile* $\boldsymbol{\sigma} = \{\sigma_i\}_{i\in[r]}$ is a tuple of mixed strategies, one per player. So, a profile is the degenerate case of a mixed profile where all probabilities are either 0 or 1. A *partial mixed profile* $\boldsymbol{\sigma}_{-i}$ is a tuple of $r-1$ mixed strategies, one per player other than i. For a mixed profile $\boldsymbol{\sigma}$ and a mixed strategy τ_i of player $i \in [r]$, denote as $\boldsymbol{\sigma}_{-i} \diamond \tau_i$ the mixed profile obtained by substituting τ_i for σ_i in the mixed profile $\boldsymbol{\sigma}$.

A mixed profile $\boldsymbol{\sigma}$ induces a probability measure $\mathbb{P}_{\boldsymbol{\sigma}}$ on Σ in the natural way; so, for a profile $\mathbf{s}$, $\mathbb{P}_{\boldsymbol{\sigma}}(\mathbf{s}) = \prod_{k\in[r]} \sigma_k(s_k)$. Say that the profile $\mathbf{s} \in \Sigma$ is *supported* in the mixed profile $\boldsymbol{\sigma}$ if $\mathbb{P}_{\boldsymbol{\sigma}}(\mathbf{s}) > 0$. Under the mixed profile $\boldsymbol{\sigma}$, the payoff of each player becomes a random variable. So, associated with $\boldsymbol{\sigma}$ is the *expected payoff* for each player $i \in [r]$,

denoted as $\mathsf{U}_i(\boldsymbol{\sigma})$, which is the expectation according to $\mathbb{P}_{\boldsymbol{\sigma}}$ of her payoff; so, clearly, $\mathsf{U}_i(\boldsymbol{\sigma}) = \sum_{\mathbf{s}\in\Sigma}\left(\prod_{k\in[r]}\sigma_k(s_k)\right)\cdot\mathsf{U}_i(\mathbf{s})$.

A *pure Nash equilibrium* is a profile $\mathbf{s}\in\Sigma$ such that for each player $i\in[r]$ and for each strategy $t\in\Sigma_i$, $\mathsf{U}_i(\mathbf{s}) \geq \mathsf{U}_i(\mathbf{s}_{-i}\diamond t)$. A *mixed Nash equilibrium,* or *Nash equilibrium* for short, is a mixed profile $\boldsymbol{\sigma}$ such that for each player $i\in[r]$ and for each mixed strategy τ_i, $\mathsf{U}_i(\boldsymbol{\sigma}) \geq \mathsf{U}_i(\boldsymbol{\sigma}_{-i}\diamond\tau_i)$. A Nash equilibrium $\boldsymbol{\sigma}$ is *fully mixed* if $\mathsf{Support}(\sigma_i)=\Sigma_i$ for each player $i\in[r]$. Denote as $\mathcal{NE}(\mathsf{G})$ the set of Nash equilibria for G. We shall make extensive use of the following characterization of Nash equilibria.

▶ **Lemma 1.** *A mixed profile $\boldsymbol{\sigma}$ is a Nash equilibrium if and only if for each player $i\in[r]$, (1) for each strategy $t\in\mathsf{Support}(\sigma_i)$, $\mathsf{U}_i(\boldsymbol{\sigma}) = \mathsf{U}_i(\boldsymbol{\sigma}_{-i}\diamond t)$, and (2) for each strategy $t\notin\mathsf{Support}(\sigma_i)$, $\mathsf{U}_i(\boldsymbol{\sigma}) \geq \mathsf{U}_i(\boldsymbol{\sigma}_{-i}\diamond t)$.*

We also recall a simple technical fact from [1, Lemma 2.2].

▶ **Lemma 2.** *Fix a win-lose game G with the positive payoff property. Then, in a Nash equilibrium $\boldsymbol{\sigma}$, for each player $i\in[r]$, $\mathsf{U}_i(\boldsymbol{\sigma})>0$.*

For an arbitrary number δ, we denote as $\mathsf{G}+\delta$ the game obtained from G by adding δ to each possible value of the payoff function. We recall a very simple, well-known fact:

▶ **Lemma 3.** *Consider the r-player games G and $\widehat{\mathsf{G}}=\mathsf{G}+\delta$, for some number δ. Then, $\mathcal{NE}(\mathsf{G}) = \mathcal{NE}(\widehat{\mathsf{G}})$. Moreover, for every Nash equilibrium $\boldsymbol{\sigma}\in\mathcal{NE}(\widehat{\mathsf{G}})$ and player $i\in[r]$, $\widehat{\mathsf{U}}_i(\boldsymbol{\sigma}) = \mathsf{U}_i(\boldsymbol{\sigma})+\delta$.*

2.3 Decision Problems about Nash Equilibria

Here are the formal statements of the decision problems we shall consider, in the style of Garey and Johnson [8], where I. and Q. stand for Instance and Question, respectively.

∃ NASH IN A BALL

I.:	A game G and a rational number $k\in(0,1)$.
Q.:	Is there a Nash equilibrium $\boldsymbol{\sigma}$ such that for each player $i\in[r]$, $\max_{s\in\Sigma_i}\sigma_i(s)\leq k$?

∃ SECOND NASH

I.:	A game G.
Q.:	Is there a second Nash equilibrium?

∃ NASH WITH LARGE PAYOFFS

I.:	A game G and a number u.
Q.:	Is there a Nash equilibrium $\boldsymbol{\sigma}$ such that for each player $i\in[r]$, $\mathsf{U}_i(\boldsymbol{\sigma})\geq u$?

∃ NASH WITH SMALL PAYOFFS

I.:	A game G and a number u.
Q.:	Is there a Nash equilibrium $\boldsymbol{\sigma}$ such that for each player $i\in[r]$, $\mathsf{U}_i(\boldsymbol{\sigma})\leq u$?

∃ NASH WITH LARGE TOTAL PAYOFF

I.:	A game G and a number u.
Q.:	Is there a Nash equilibrium $\boldsymbol{\sigma}$ such that $\sum_{i\in[r]}\mathsf{U}_i(\boldsymbol{\sigma})\geq u$?

∃ NASH WITH SMALL TOTAL PAYOFF

I.:	A game G and a number u.
Q.:	Is there a Nash equilibrium $\boldsymbol{\sigma}$ such that $\sum_{i\in[r]} \mathsf{U}_i(\boldsymbol{\sigma}) \leq u$?

∃ NASH WITH LARGE SUPPORTS

I.:	A game G and an integer $k \geq 1$.
Q.:	Is there a Nash equilibrium $\boldsymbol{\sigma}$ such that for each player $i \in [r]$, $\lvert\mathsf{Support}(\sigma_i)\rvert \geq k$?

∃ NASH WITH SMALL SUPPORTS

I.:	A game G and an integer $k \geq 1$.
Q.:	Is there a Nash equilibrium $\boldsymbol{\sigma}$ such that for each player $i \in [r]$, $\lvert\mathsf{Support}(\sigma_i)\rvert \leq k$?

∃ NASH WITH RESTRICTING SUPPORTS

I.:	A game G and a subset of strategies $\mathsf{T}_i \subseteq \mathbf{\Sigma}_i$ for each player $i \in [r]$.
Q.:	Is there a Nash equilibrium $\boldsymbol{\sigma}$ such that for each player $i \in [r]$, $\mathsf{T}_i \subseteq \mathsf{Support}(\sigma_i)$?

∃ NASH WITH RESTRICTED SUPPORTS

I.:	A game G and a subset of strategies $\mathsf{T}_i \subseteq \mathbf{\Sigma}_i$ for each player $i \in [r]$.
Q.:	Is there a Nash equilibrium $\boldsymbol{\sigma}$ such that for each player $i \in [r]$, $\mathsf{Support}(\sigma_i) \subseteq \mathsf{T}_i$?

Given two mixed profiles $\boldsymbol{\sigma}$ and $\widehat{\boldsymbol{\sigma}}$, denote as $\mathsf{Diff}(\boldsymbol{\sigma}, \widehat{\boldsymbol{\sigma}}) := \{i \in [r] : \sigma_i \neq \widehat{\sigma}_i\}$ the set of players with different mixed strategies in $\boldsymbol{\sigma}$ and $\widehat{\boldsymbol{\sigma}}$. A Nash equilibrium $\boldsymbol{\sigma}$ is *Strongly Pareto-Optimal* if for each mixed profile $\widehat{\boldsymbol{\sigma}}$ where there is player $i \in [r]$ with $\mathsf{U}_i(\widehat{\boldsymbol{\sigma}}) > \mathsf{U}_i(\boldsymbol{\sigma})$ for some player $i \in [r]$, there is a player $j \in \mathsf{Diff}(\boldsymbol{\sigma}, \widehat{\boldsymbol{\sigma}})$ such that $\mathsf{U}_j(\widehat{\boldsymbol{\sigma}}) \leq \mathsf{U}_j(\boldsymbol{\sigma})$; so, there is no other profile where at least one player is strictly better off and every player using a different strategy is strictly better off. We have two additional decision problems.

∃ NON-PARETO-OPTIMAL NASH

I.:	A game G.
Q.:	Is there a Nash equilibrium which is not Pareto-Optimal?

∃ NON-STRONGLY PARETO-OPTIMAL NASH

I.:	A game G.
Q.:	Is there a Nash equilibrium which is not Strongly Pareto-Optimal?

Restricted to 2-player games with rational utilities, all these problems are $\mathcal{NP}$-complete. ∃ NASH IN A BALL was the first problem shown $\exists\mathbb{R}$-complete. The problems ∃ SECOND NASH, ∃ NASH WITH LARGE PAYOFFS, ∃ NASH WITH RESTRICTING SUPPORTS and ∃ NASH WITH RESTRICTED SUPPORTS are $\exists\mathbb{R}$-complete for r-player games with $r \geq 3$ [9].

2.4 The Game $\widehat{\mathsf{G}}[m]$

For an integer $m \geq 2$, define the 3-player win-lose game $\widehat{\mathsf{G}}[m]$ as follows: For each player $i \in [3]$, $\widehat{\mathbf{\Sigma}}_i = [m]$, and the strategy sets are *cyclic* so that strategy 0 coincides with strategy m; the payoff functions are: (1) $\widehat{\mathsf{U}}_1(\mathbf{s}) = 1$ if and only if $s_1 = s_2$; (2) $\widehat{\mathsf{U}}_2(\mathbf{s}) = 1$ if and only if $s_2 = s_3 - 1$; (3) $\widehat{\mathsf{U}}_3(\mathbf{s}) = 1$ if and only if $s_3 = s_1 - 1$. Note that $\widehat{\mathsf{G}}[m]$ has the positive payoff property. We prove:

▶ **Lemma 4.** *Fix an odd integer $m \geq 3$. Then, $\widehat{\mathsf{G}}[m]$ has a unique Nash equilibrium $\boldsymbol{\sigma}$, which is fully mixed and has $\widehat{\mathsf{U}}_i(\boldsymbol{\sigma}) = \frac{1}{m}$ for each player $i \in [3]$.*

Proof. Fix a Nash equilibrium $\boldsymbol{\sigma}$ for $\widehat{\mathsf{G}}[m]$. To prove that $\boldsymbol{\sigma}$ is fully mixed, assume, by way of contradiction, that for a strategy $j \in [m]$, $\sigma_1(j) = 0$. This implies that $\sigma_3(j-1) = 0$ since otherwise ($\sigma_3(j-1) > 0$), Lemma 1 (Condition (1)) implies that $\widehat{\mathsf{U}}_3(\boldsymbol{\sigma}) = \widehat{\mathsf{U}}_3(\boldsymbol{\sigma}_{-3} \diamond (j-1)) = 0$, which contradicts Lemma 2. This implies that $\sigma_2(j-2) = 0$ since otherwise ($\sigma_2(j-2) > 0$), Lemma 1 (Condition (1)) implies that $\widehat{\mathsf{U}}_2(\boldsymbol{\sigma}) = \widehat{\mathsf{U}}_2(\boldsymbol{\sigma}_{-2} \diamond (j-2)) = 0$, which contradicts Lemma 2. This implies that $\sigma_1(j-2) = 0$ since otherwise ($\sigma_1(j-2) > 0$), Lemma 1 (Condition (1)) implies that $\widehat{\mathsf{U}}_1(\boldsymbol{\sigma}) = \widehat{\mathsf{U}}_1(\boldsymbol{\sigma}_{-1} \diamond (j-2)) = 0$, which contradicts Lemma 2. Since m is odd, a repeated application of the implication yields that $\sigma_1(1) = \ldots = \sigma_1(m) = 0$, $\sigma_2(1) = \ldots = \sigma_2(m) = 0$ and $\sigma_3(1) = \ldots = \sigma_3(m) = 0$. A contradiction. Hence, for each player $i \in [3]$, for each strategy $j \in [m]$, $\sigma_i(j) > 0$, or $\boldsymbol{\sigma}$ is fully mixed. By Lemma 1 (Condition (1)), this implies that for each player $i \in [3]$, for each strategy $j \in [m]$, $\widehat{\mathsf{U}}_i(\boldsymbol{\sigma}) = \widehat{\mathsf{U}}_i(\boldsymbol{\sigma}_{-i} \diamond j)$. By the definition of the payoff functions, for each strategy $j \in [m]$, $\widehat{\mathsf{U}}_1(\boldsymbol{\sigma}_{-1} \diamond j) = \sigma_2(j)$. $\widehat{\mathsf{U}}_2(\boldsymbol{\sigma}_{-2} \diamond j) = \sigma_3(j+1)$ and $\widehat{\mathsf{U}}_3(\boldsymbol{\sigma}_{-3} \diamond j) = \sigma_1(j+1)$. Hence, it follows that for each player $i \in [3]$ and for each strategy $j \in [m]$, $\sigma_i(j)$ is independent of j, which implies that $\sigma_i(j) = \frac{1}{m}$, so that $\boldsymbol{\sigma}$ is unique with $\mathsf{U}_i(\boldsymbol{\sigma}) = \frac{1}{m}$ for each player $i \in [3]$. ◀

3 The Game Reduction

Fix an arbitrary number $\delta > 0$. The game reduction takes as input a pair of games:

- A 3-player game $\widetilde{\mathsf{G}}$ with $\Sigma_i = [n]$ for each player $i \in [3]$, the *inbox game*.
- A 3-player game $\widehat{\mathsf{G}}$ with $\Sigma_i = [m]$ for each player $i \in [3]$, with $m \geq n$, the *gadget game*.

The game reduction constructs a 3-player game $\mathsf{G} = \mathsf{G}\left(\widetilde{\mathsf{G}}, \widehat{\mathsf{G}}\right)$; $\widetilde{\mathsf{G}}$ and $\widehat{\mathsf{G}}$ are the *subgames*.

3.1 Definition and Some Notation

Set $\phi^* := \min\left\{\underline{u}(\widetilde{\mathsf{G}}), \underline{u}(\widehat{\mathsf{G}})\right\} - 1$ and $\phi := \phi^* - 1$. We construct the game G as follows:

- For each player $i \in [r]$, $\Sigma_i = [p]$, with $p = n(n+1)+m$; $[p]$ is partitioned into $n+2$ blocks $\mathsf{B}_0, \mathsf{B}_1, \ldots, \mathsf{B}_{n+1}$, where for each index h with $0 \leq h \leq n$, $\mathsf{B}_h := \{hn+1, \ldots, (h+1)n\}$ and $\mathsf{B}_{n+1} := \{n(n+1)+1, \ldots, n(n+1)+m\}$; thus, $|\mathsf{B}_{n+1}| = m$, while $|\mathsf{B}_h| = n$ for each index h with $0 \leq h \leq n$. For each index h with $0 \leq h \leq n+1$ and $k \in [|\mathsf{B}_h|]$, denote as $\mathsf{B}_h(k)$ the order k strategy in B_h; thus, $\mathsf{B}_h(k)$ is the order $(hn+k)$ strategy in Σ_i.
- The payoff functions for G are given in Figure 1.

Clearly, G is constructed in time polynomial in the sizes of $\widetilde{\mathsf{G}}$ and $\widehat{\mathsf{G}}$. Note that by Case (1), $\widetilde{\mathsf{G}} + \delta$ is a subgame of G; by Case (2), $\widehat{\mathsf{G}}$ is a subgame of G. So, the blocks B_0 and B_{n+1} correspond to the input games $\widetilde{\mathsf{G}}$ and $\widehat{\mathsf{G}}$, respectively.

For an n-dimensional vector $\boldsymbol{x} \in \mathbb{R}^n$, denote as $\overrightarrow{\boldsymbol{x}} \in \mathbb{R}^p$ the p-dimensional vector with $\overrightarrow{x}_j = x_j$ for $j \in [n]$ and $\overrightarrow{x}_j = 0$ for $n < j \leq p$. Similarly, for an m-dimensional vector $\boldsymbol{x} \in \mathbb{R}^m$, denote as $\overleftarrow{\boldsymbol{x}} \in \mathbb{R}^p$ the p-dimensional vector with $\overleftarrow{x}_j = 0$ for $i \in [n(n+1)]$ and $\overleftarrow{x}_j = x_j$ for $n(n+1) < j \leq p$. Hence, for a mixed profile $\boldsymbol{\sigma} \in \widetilde{\Sigma}$, $(\overrightarrow{\sigma_1}, \overrightarrow{\sigma_2}, \overrightarrow{\sigma_3}) \in \Sigma$; for a mixed profile $\boldsymbol{\sigma} \in \widehat{\Sigma}$, $(\overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \overleftarrow{\sigma_3}) \in \Sigma$. For a given multidimensional space, $\mathcal{B}_\varrho$ denote the ball of radius ϱ. For a mixed profile $\boldsymbol{\sigma}$, we write $\boldsymbol{\sigma} \in \mathcal{B}_\varrho$ when $\sigma_i \in \mathcal{B}_\varrho$ for each player $i \in [3]$.

Case	Condition on the profile $\mathbf{s}$	Payoff vector $\mathsf{U}(\mathbf{s})$
(1)	$s_i \in \mathsf{B}_0$ for each player $i \in [3]$	$\left\langle \widetilde{\mathsf{U}}_1(\mathbf{s})+\delta, \widetilde{\mathsf{U}}_2(\mathbf{s})+\delta, \widetilde{\mathsf{U}}_3(\mathbf{s})+\delta \right\rangle$
(2)	$s_i \in \mathsf{B}_{n+1}$ for each player $i \in [3]$	$\left\langle \widehat{\mathsf{U}}_1(\mathbf{s}), \widehat{\mathsf{U}}_2(\mathbf{s}), \widehat{\mathsf{U}}_3(\mathbf{s}) \right\rangle$
(3)	$s_i = \mathsf{B}_{n+1}(k)$ with $k \in [n]$ & $s_j \in \mathsf{B}_0$ for $j \neq i$	$\mathsf{U}_i(\mathbf{s}) = \mathsf{U}_i(\mathbf{s}_{-i} \diamond k)$ $\mathsf{U}_j(\mathbf{s}) = \phi$ for $j \neq i$
(4)	$s_i = \mathsf{B}_h(k)$ with $h, k \in [n]$ & $s_j \in \mathsf{B}_0$ for $j \neq i$ with $s_{i+1} = h$	$\mathsf{U}_i(\mathbf{s}) = \widetilde{\mathsf{U}}_1(\mathbf{s}_{-i} \diamond k) + 2\delta$ $\mathsf{U}_j(\mathbf{s}) = \phi$ for $j \neq i$
(5)	$s_i = \mathsf{B}_h(k)$ with $h, k \in [n]$ & $s_j \in \mathsf{B}_0$ for $j \neq i$ with $s_{i+1} \neq h$	$\mathsf{U}_i(\mathbf{s}) = \widetilde{\mathsf{U}}_i(\mathbf{s}_{-i} \diamond k)$ $\mathsf{U}_j(\mathbf{s}) = \phi$ for $j \neq i$
(6)	$\mathsf{P}(\mathbf{s}) \neq \emptyset$ & $\mathsf{P}(\mathbf{s}) \neq [3]$, with $\mathbf{s}$ not falling in Case (3)	$\mathsf{U}_i(\mathbf{s}) = \phi^*$ if $i \in \mathsf{P}(\mathbf{s})$ $\mathsf{U}_i(\mathbf{s}) = \phi$, if $i \notin \mathsf{P}(\mathbf{s})$
(7)	None of the above	$\langle \phi, \phi, \phi \rangle$

Figure 1 The payoff functions for the game G. Here $\mathsf{P}(\mathbf{s}) := \{i \in [3] \mid s_i \in \mathsf{B}_{n+1}\}$, the set of players choosing strategies from the block B_{n+1} in the profile $\mathbf{s}$.

3.2 Correspondences Between Nash Equilibria

We now establish certain correspondences between the Nash equilibria for the subgames $\widetilde{\mathsf{G}}$ and $\widehat{\mathsf{G}}$, respectively, and the Nash equilibria for the constructed game G.

3.2.1 Backward Correspondence: From the Game G to the Subgames

We shall prove that a Nash equilibrium for G is induced by a Nash equilibrium for either $\widetilde{\mathsf{G}}$ or $\widehat{\mathsf{G}}$. We start with two technical claims about a Nash equilibrium for G (Lemmas 5 and 6). We first prove that if some player is playing some strategy outside B_0, then none of the other two players is playing a strategy in B_0.

▶ **Lemma 5.** *Fix a Nash equilibrium $\boldsymbol{\sigma} \in \mathcal{NE}(\mathsf{G})$ for which there is a player i' such that $\mathsf{Support}(\sigma_{i'}) \setminus \mathsf{B}_0 \neq \emptyset$. Then, for every player $i \neq i'$, $\mathsf{Support}(\sigma_i) \cap \mathsf{B}_0 = \emptyset$.*

Proof. Assume, by way of contradiction, that there is a player $i \neq i'$ with $\mathsf{Support}(\sigma_i) \cap \mathsf{B}_0 \neq \emptyset$. Choose an arbitrary strategy $k \in \mathsf{Support}(\sigma_i) \cap \mathsf{B}_0$. Since $k \in \mathsf{Support}(\sigma_i)$, Lemma 1 (Condition (1)) implies that $\mathsf{U}_i(\boldsymbol{\sigma}) = \sum_{\mathbf{s}_{-i} \in \Sigma_{-i}} \mathsf{U}_i(\mathbf{s}_{-i} \diamond k) \cdot \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i})$. Lemma 1 (Condition (2)) implies that

$$\mathsf{U}_i(\boldsymbol{\sigma}_{-i} \diamond \mathsf{B}_{n+1}(k)) \quad = \quad \sum_{\mathbf{s}_{-i} \in \Sigma_{-i}} \mathsf{U}_i(\mathbf{s}_{-i} \diamond \mathsf{B}_{n+1}(k)) \cdot \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i}) \quad \leq \quad \mathsf{U}_i(\boldsymbol{\sigma})\,.$$

Fix a partial profile $\mathbf{s}_{-i} \in \Sigma_{-i}$. There are six possible cases for the two players other than i.

1. *Both players choose a strategy from B_0.* Then, $\mathbf{s}_{-i} \diamond k$ falls into Case (1) of the payoff table, while $\mathbf{s}_{-i} \diamond \mathsf{B}_{n+1}(k)$ falls into Case (3), so that $\mathsf{U}_i(\mathbf{s}_{-i} \diamond k) = \mathsf{U}_i(\mathbf{s}_{-i} \diamond \mathsf{B}_{n+1}(k))$.
2. *Both players choose a strategy from B_{n+1}.* Then, $\mathbf{s}_{-i} \diamond k$ falls into Case (6) of the payoff table with $\mathsf{U}_i(\mathbf{s}_{-i} \diamond k) = \phi$, while $\mathbf{s}_{-i} \diamond \mathsf{B}_{n+1}(k)$ falls into Case (2) with $\mathsf{U}_i(\mathbf{s}_{-i} \diamond \mathsf{B}_{n+1}(k)) \geq \underline{u}(\widehat{\mathsf{G}}) > \phi$.
3. *Both players choose a strategy outside $\mathsf{B}_0 \cup \mathsf{B}_{n+1}$.* Then, $\mathbf{s}_{-i} \diamond k$ falls into Case (7) of the payoff table with $\mathsf{U}_i(\mathbf{s}_{-i} \diamond k) = \phi$, while $\mathbf{s}_{-i} \diamond \mathsf{B}_{n+1}(k)$ falls into Case (6) with $\mathsf{U}_i(\mathbf{s}_{-i} \diamond \mathsf{B}_{n+1}(k)) = \phi^* > \phi$.
4. *One player chooses a strategy from B_0 and the other chooses one from B_{n+1}.* Then, $\mathbf{s}_{-i} \diamond k$ falls into Case (3) of the payoff table with $\mathsf{U}_i(\mathbf{s}_{-i} \diamond k) = \phi$, while $\mathbf{s}_{-i} \diamond \mathsf{B}_{n+1}(k)$ falls into Case (6) with $\mathsf{U}_i(\mathbf{s}_{-i} \diamond \mathsf{B}_{n+1}(k)) = \phi^* > \phi$.

5. *One player chooses a strategy from* B_0 *and the other chooses one outside* $\mathsf{B}_0 \cup \mathsf{B}_{n+1}$.
Then, $\mathbf{s}_{-i} \diamond k$ falls into either Case (4) or (5) of the payoff table with $\mathsf{U}_i(\mathbf{s}_{-i} \diamond k) = \phi$, while $\mathbf{s}_{-i} \diamond \mathsf{B}_{n+1}(k)$ falls into Case (6) with $\mathsf{U}_i(\mathbf{s}_{-i} \diamond \mathsf{B}_{n+1}(k)) = \phi^* > \phi$.
6. *One player chooses a strategy from* B_{n+1} *and the other chooses one outside* $\mathsf{B}_0 \cup \mathsf{B}_{n+1}$.
Then, both $\mathbf{s}_{-i} \diamond k$ and $\mathbf{s}_{-i} \diamond \mathsf{B}_{n+1}(k)$ fall into Case (6) of the payoff table with $\mathsf{U}_i(\mathbf{s}_{-i} \diamond k) = \phi$ and $\mathsf{U}_i(\mathbf{s}_{-i} \diamond \mathsf{B}_{n+1}(k)) = \phi^* > \phi$.

Note that in the first of the six cases, player i has the same payoff for the two strategies k and $\mathsf{B}_{n+1}(k)$, while in all other cases, player i improves her payoff when switching to $\mathsf{B}_{n+1}(k)$. Hence, there is no profile supported in $\boldsymbol{\sigma}_{-i} \diamond \mathsf{B}_{n+1}(k)$ in which player i has a smaller payoff. The assumption implies that $\mathsf{Support}(\sigma_j) \setminus \mathsf{B}_0 \neq \emptyset$; it follows that there is at least one profile supported in $\boldsymbol{\sigma}_{-i} \diamond \mathsf{B}_{n+1}(k)$ for which player i has a larger payoff. A contradiction. ◀

We continue to prove that if some player is playing no strategy from B_0, then the other two players are playing only strategies from B_{n+1}.

▶ **Lemma 6.** *Fix a Nash equilibrium* $\boldsymbol{\sigma} \in \mathcal{NE}(\mathsf{G})$ *for which there is a player* i' *with* $\mathsf{Support}(\sigma_{i'}) \cap \mathsf{B}_0 = \emptyset$. *Then, for each player* $i \neq i'$, $\mathsf{Support}(\sigma_i) \setminus \mathsf{B}_{n+1} = \emptyset$.

Proof. Assume, by way of contradiction, that there is a player $i \neq i'$ with $\mathsf{Support}(\sigma_i) \setminus \mathsf{B}_{n+1} \neq \emptyset$. Choose an arbitrary strategy $k \in \mathsf{Support}(\sigma_i) \setminus \mathsf{B}_{n+1}$ and an arbitrary strategy $h \in \mathsf{B}_{n+1}$. Since $k \in \mathsf{Support}(\sigma_i)$, Lemma 1 (Condition (1)) implies that $\mathsf{U}_i(\boldsymbol{\sigma}) = \sum_{\mathbf{s}_{-i} \in \boldsymbol{\Sigma}_{-i}} \mathsf{U}_i(\mathbf{s}_{-i} \diamond k) \cdot \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i})$. Lemma 1 (Condition (2)) implies that

$$\mathsf{U}_i(\boldsymbol{\sigma}_{-i} \diamond h) \quad = \quad \sum_{\mathbf{s}_{-i} \in \boldsymbol{\Sigma}_{-i}} \mathsf{U}_i(\mathbf{s}_{-i} \diamond h) \cdot \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i}) \quad \leq \quad \mathsf{U}_i(\boldsymbol{\sigma}) \,.$$

Fix now a profile $\mathbf{s}_{-i} \in \boldsymbol{\Sigma}_{-i}$. There are five possible cases for the two players other than i.

1. *Both players play a strategy from* B_{n+1}. Then, $\mathbf{s}_{-i} \diamond k$ falls into Case (6) of the payoff table with $\mathsf{U}_i(\mathbf{s}_{-i} \diamond k) = \phi$, while $\mathbf{s}_{-i} \diamond h$ falls into Case (2) with $\mathsf{U}_i(\mathbf{s}_{-i} \diamond h) \geq \underline{u}(\widehat{\mathsf{G}}) > \phi$.
2. *Both players choose a strategy outside* $\mathsf{B}_0 \cup \mathsf{B}_{n+1}$. Then, $\mathbf{s}_{-i} \diamond k$ falls into Case (7) of the payoff table with $\mathsf{U}_i(\mathbf{s}_{-i} \diamond k) = \phi$, while $\mathbf{s}_{-i} \diamond h$ falls into Case (6) with $\mathsf{U}_i(\mathbf{s}_{-i} \diamond h) = \phi^* > \phi$.
3. *One player chooses a strategy from* B_0 *and the other chooses one from* B_{n+1}.
Then, $\mathbf{s}_{-i} \diamond k$ falls into either Case (3) or (7) of the payoff table with $\mathsf{U}_i(\mathbf{s}_{-i} \diamond k) = \phi$, while $\mathbf{s}_{-i} \diamond h$ falls into Case (6) with $\mathsf{U}_i(\mathbf{s}_{-i} \diamond h) = \phi^* > \phi$.
4. *One player chooses a strategy from* B_0 *and the other chooses one outside* $\mathsf{B}_0 \cup \mathsf{B}_{n+1}$.
Then, $\mathbf{s}_{-i} \diamond k$ falls into either Case (4) or (5) or (7) of the payoff table with $\mathsf{U}_i(\mathbf{s}_{-i} \diamond k) = \phi$, while $\mathbf{s}_{-i} \diamond h$ falls into Case (6) with $\mathsf{U}_i(\mathbf{s}_{-i} \diamond h) = \phi^* > \phi$.
5. *One player chooses a strategy from* B_{n+1} *and the other chooses one outside* $\mathsf{B}_0 \cup \mathsf{B}_{n+1}$.
Then, both $\mathbf{s}_{-i} \diamond k$ and $\mathbf{s}_{-i} \diamond h$ fall into Case (6) of the payoff table, so that $\mathsf{U}_i(\mathbf{s}_{-i} \diamond k) = \phi$ and $\mathsf{U}_i(\mathbf{s}_{-i} \diamond h) = \phi^* > \phi$.

Thus, in all profiles supported in $\boldsymbol{\sigma}_{-i} \diamond \mathsf{B}_{n+1}(k)$, player i improves her payoff by switching to strategy h. A contradiction. ◀

We are now ready to prove:

▶ **Lemma 7.** *Fix a Nash equilibrium* $\boldsymbol{\sigma} \in \mathcal{NE}(\mathsf{G})$. *Then, there are only two possible cases:*

- $\boldsymbol{\sigma} = (\overrightarrow{\tau_1}, \overrightarrow{\tau_2}, \overrightarrow{\tau_3})$ *for some Nash equilibrium* $\boldsymbol{\tau} \in \mathcal{NE}(\widetilde{\mathsf{G}})$.
- $\boldsymbol{\sigma} = (\overleftarrow{\tau_1}, \overleftarrow{\tau_2}, \overleftarrow{\tau_3})$ *for some Nash equilibrium* $\boldsymbol{\tau} \in \mathcal{NE}(\widehat{\mathsf{G}})$.

Proof. By Lemmas 5 and 6, either *(i)* $\sigma_1 = \overrightarrow{\tau_1}$, $\sigma_2 = \overrightarrow{\tau_2}$ and $\sigma_3 = \overrightarrow{\tau_3}$ for some mixed profile $\boldsymbol{\tau} \in \widetilde{\boldsymbol{\Sigma}}$ or *(ii)* $\sigma_1 = \overleftarrow{\tau_1}$, $\sigma_2 = \overleftarrow{\tau_2}$, $\sigma_3 = \overleftarrow{\tau_3}$ for some mixed profile $\boldsymbol{\tau} \in \widehat{\boldsymbol{\Sigma}}$. The two properties that $\boldsymbol{\tau} \in \mathcal{NE}(\widetilde{\mathsf{G}})$ and $\boldsymbol{\tau} \in \mathcal{NE}(\widehat{\mathsf{G}})$ follow from the facts that both $\widetilde{\mathsf{G}} + \delta$ and $\widehat{\mathsf{G}}$ are subgames of G, and from Lemma 3, by which $\widetilde{\mathsf{G}}$ and $\widetilde{\mathsf{G}} + \delta$ have the same set of Nash equilibria. ◀

3.2.2 Forward Correspondence: From the Subgames to the Game G

We now characterize the Nash equilibria for the two subgames $\widetilde{\mathsf{G}}$ and $\widehat{\mathsf{G}}$ that induce corresponding Nash equilibria for G. Specifically, these are all the Nash equilibria for $\widehat{\mathsf{G}}$ (Lemma 8) and every Nash equilibrium in $\mathcal{B}_{1/2}$ for $\widetilde{\mathsf{G}}$ (Lemma 9), respectively. We first prove:

▶ **Lemma 8.** *Fix a Nash equilibrium $\boldsymbol{\sigma} \in \mathcal{NE}(\widehat{\mathsf{G}})$. Then, $(\overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \overleftarrow{\sigma_3}) \in \mathcal{NE}(\mathsf{G})$.*

Proof. By Case (2) of the payoff table, for each player $i \in [3]$, $\mathsf{U}_i\left(\overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \overleftarrow{\sigma_3}\right) = \widehat{\mathsf{U}}_i(\boldsymbol{\sigma}) \geq \underline{u}(\widehat{\mathsf{G}})$.

Since $\boldsymbol{\sigma} \in \mathcal{NE}(\widehat{\mathsf{G}})$, no player could improve her payoff by deviating to a strategy from B_{n+1}. A deviation to a strategy outside B_{n+1} gives rise to a mixed profile for which only profiles from Case (6) are supported, and the expected payoff of the deviating player is $\phi < \underline{u}(\widehat{\mathsf{G}})$. ◀

We continue to prove:

▶ **Lemma 9.** *Fix a Nash equilibrium $\boldsymbol{\sigma} \in \mathcal{NE}(\widetilde{\mathsf{G}})$. Then, $(\overrightarrow{\sigma_1}, \overrightarrow{\sigma_2}, \overrightarrow{\sigma_3}) \in \mathcal{NE}(\mathsf{G})$ if and only if $\boldsymbol{\sigma} \in \mathcal{B}_{1/2}$.*

Proof. Fix a player $i \in [3]$. By Lemma 1 (Condition (1)), for each strategy $j \in \mathsf{Support}(\sigma_i)$,

$$\begin{aligned}
\mathsf{U}_i\left(\overrightarrow{\sigma_1}, \overrightarrow{\sigma_2}, \overrightarrow{\sigma_3}\right) &= \sum_{\mathbf{s}_{-i} \in \widetilde{\boldsymbol{\Sigma}}_{-i}} \mathsf{U}_i\left(\mathbf{s}_{-i} \diamond j\right) \cdot \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i}) \\
&= \sum_{\mathbf{s}_{-i} \in \widetilde{\boldsymbol{\Sigma}}_{-i}} \mathsf{U}_i(\mathbf{s}_{-i} \diamond j) \cdot \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i}) \\
&= \sum_{\mathbf{s}_{-i} \in \widetilde{\boldsymbol{\Sigma}}_{-i}} \left(\widetilde{\mathsf{U}}_i\left(\mathbf{s}_{-i} \diamond j\right) + \delta\right) \cdot \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i}) \\
&= \delta + \sum_{\mathbf{s}_{-i} \in \widetilde{\boldsymbol{\Sigma}}_{-i}} \widetilde{\mathsf{U}}_i\left(\mathbf{s}_{-i} \diamond j\right) \cdot \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i}) \\
&> \underline{u}(\widetilde{\mathsf{G}})\,.
\end{aligned}$$

By Lemma 3, $\mathcal{NE}(\widetilde{\mathsf{G}}) = \mathcal{NE}(\widetilde{\mathsf{G}} + \delta)$; thus, player i cannot improve her payoff by switching to a strategy from B_0. Also, by Case (3), player i cannot improve her payoff by switching to any of the first n strategies in B_{n+1}; by Case (6), player i cannot improve her payoff by switching to any of the last $m - n$ strategies in B_{n+1} since $\underline{u}(\widetilde{\mathsf{G}}) > \phi^*$. So, it remains to consider deviations to strategies outside $\mathsf{B}_0 \cup \mathsf{B}_{n+1}$. We proceed by case analysis.

1. Assume first that $\boldsymbol{\sigma} \notin \mathcal{B}_{1/2}$. Hence, there is a player $h = i + 1$ with $\sigma_h \notin \mathcal{B}_{1/2}$. (This assumption is without loss of generality since i is arbitrary.) Consider the third player $h' = i + 2$. Denote as $k \in \mathsf{B}_0$ the strategy with $\sigma_h(k) > \frac{1}{2}$. Consider player i switching to the strategy $\mathsf{B}_k(j)$. By Cases (4) and (5) of the payoff table, her expected payoff is

$$\begin{aligned}
&\sum_{\mathbf{s}_{-i} \in \widetilde{\boldsymbol{\Sigma}}_{-i}} \mathsf{U}_i\left(\mathbf{s}_{-i} \diamond \mathsf{B}_k(j)\right) \cdot \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i}) \\
= &\sum_{\mathbf{s}_{-i} \in \widetilde{\boldsymbol{\Sigma}}_{-i}} \mathsf{U}_i\left(\mathbf{s}_{-i} \diamond \mathsf{B}_k(j)\right) \cdot \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i}) \\
= &\sum_{\mathbf{s}_{-i} \in \widetilde{\boldsymbol{\Sigma}}_{-i} : s_h = k} \left(\widetilde{\mathsf{U}}_i(\mathbf{s}_{-i} \diamond j) + 2\delta\right) \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i}) + \sum_{\mathbf{s}_{-i} \in \widetilde{\boldsymbol{\Sigma}}_{-i} : s_h \neq k} \widetilde{\mathsf{U}}_i(\mathbf{s}_{-i} \diamond j) \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i}) \\
= &\; 2\delta \sum_{\mathbf{s}_{-i} \in \widetilde{\boldsymbol{\Sigma}}_{-i} : s_h = k} \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i}) + \sum_{\mathbf{s}_{-i} \in \widetilde{\boldsymbol{\Sigma}}_{-i}} \widetilde{\mathsf{U}}_i(\mathbf{s}_{-i} \diamond j) \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i})
\end{aligned}$$

$$> \quad \delta + \sum_{\mathbf{s}_{-i} \in \widetilde{\Sigma}_{-i}} \widetilde{\mathsf{U}}_i(\mathbf{s}_{-i} \diamond j) \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i})$$
$$= \quad \mathsf{U}_i\left(\overrightarrow{\sigma_1}, \overrightarrow{\sigma_2}, \overrightarrow{\sigma_3}\right) ,$$

which implies that $(\overrightarrow{\sigma_1}, \overrightarrow{\sigma_2}, \overrightarrow{\sigma_3}) \notin \mathcal{NE}(\mathsf{G})$.

2. Assume now that $\boldsymbol{\sigma} \in \mathcal{B}_{1/2}$. Consider player i switching to an arbitrary strategy $\mathsf{B}_k(j)$ outside $\mathsf{B}_0 \cup \mathsf{B}_{n+1}$. Set $h := i+1$ and $h' := i+2$. By Cases (4) and (5), her expected payoff is

$$\sum_{\mathbf{s}_{-i} \in \boldsymbol{\Sigma}_{-i}} \mathsf{U}_i\left(\mathbf{s}_{-i} \diamond \mathsf{B}_k(j)\right) \cdot \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i})$$
$$= \sum_{\mathbf{s}_{-i} \in \widetilde{\Sigma}_{-i}} \mathsf{U}_i\left(\mathbf{s}_{-i} \diamond \mathsf{B}_k(j)\right) \cdot \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i})$$
$$= \sum_{\mathbf{s}_{-i} \in \widetilde{\Sigma}_{-i} | s_h = k} \left(\widetilde{\mathsf{U}}_i\left(\mathbf{s}_{-i} \diamond j\right) + 2\,\delta\right) \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i}) + \sum_{\mathbf{s}_{-i} \in \widetilde{\Sigma}_{-i} | s_h \neq k} \widetilde{\mathsf{U}}_i\left(\mathbf{s}_{-i} \diamond j\right) \cdot \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i})$$
$$= 2\,\delta \sum_{\mathbf{s}_{-i} \in \widetilde{\Sigma}_{-i} | s_h = k} \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i}) + \sum_{\mathbf{s}_{-i} \in \widetilde{\Sigma}_{-i}} \widetilde{\mathsf{U}}_i\left(\mathbf{s}_{-i} \diamond j\right) \cdot \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i})$$
$$\leq \quad \delta + \sum_{\mathbf{s}_{-i} \in \widetilde{\Sigma}_{-i}} \widetilde{\mathsf{U}}_i\left(\mathbf{s}_{-i} \diamond j\right) \cdot \mathbb{P}_{\boldsymbol{\sigma}_{-i}}(\mathbf{s}_{-i})$$
$$= \quad \mathsf{U}_i\left(\overrightarrow{\sigma_1}, \overrightarrow{\sigma_2}, \overrightarrow{\sigma_3}\right) ,$$

which implies that $(\overrightarrow{\sigma_1}, \overrightarrow{\sigma_2}, \overrightarrow{\sigma_3}) \in \mathcal{NE}(\mathsf{G})$.

The proof is now complete. ◀

4 The ∃ℝ-Complete Decision Problems

We now present the ∃ℝ-completeness results. We show:

▶ **Theorem 10.** *Restricted to r-player games with $r \geq 3$, the following decision problems are ∃ℝ-complete:*

Group I	*Group II*
∃ SECOND NASH	∃ NASH WITH LARGE PAYOFFS
∃ NASH WITH SMALL PAYOFFS	∃ NASH WITH LARGE TOTAL PAYOFF
∃ NASH WITH SMALL TOTAL PAYOFF	∃ WITH SMALL SUPPORTS
∃ NASH WITH LARGE SUPPORTS	
∃ NASH WITH RESTRICTING SUPPORTS	
∃ NASH WITH RESTRICTED SUPPORTS	
∃ NON-PARETO OPTIMAL NASH	
∃ NON-STRONGLY-PARETO-OPTIMAL NASH	

Membership of the decision problems (for r-player games with $r \geq 3$) in ∃ℝ is established with standard techniques, employing simple formulas to define their properties (cf. [9]).

Proof. Assume first that $r = 3$. We use a polynomial time, many-to-one reduction from ∃ NASH IN A BALL. Consider an instance $\widetilde{\mathsf{G}}$ of ∃ NASH IN A BALL with $\varrho = \frac{1}{2}$, called the *inbox game*. Assume, without loss of generality, that $\widetilde{\Sigma}_i = [n]$ for each player $i \in [3]$. We start with an outline of the proof. The reduction will be the composition of the construction of a gadget game and the game reduction from Section 3. For each of *Group I* and *Group II,* we shall employ a suitable game $\widehat{\mathsf{G}}$, called the *gadget game,* which may be constructed from the inbox game $\widetilde{\mathsf{G}}$. Then, we shall apply the game reduction from Section 3 with $\widetilde{\mathsf{G}}$

Case	Condition on the profile $\mathbf{s}$	Payoff vector $\widehat{\mathsf{U}}(\mathbf{s})$
(1)	$\mathsf{P}(\mathbf{s}) = [3]$	$\langle u^*, u^*, u^* \rangle$
(2)	$\mathsf{P}(\mathbf{s}) \neq \emptyset \wedge \mathsf{P}(\mathbf{s}) \neq [3]$	$\widehat{\mathsf{U}}_i(\mathbf{s}) = u^*$ if $i \in \mathsf{P}(\mathbf{s})$ $\widehat{\mathsf{U}}_i(\mathbf{s}) = u^* - 1$ if $i \notin \mathsf{P}(\mathbf{s})$
(3)	$\mathsf{P}(\mathbf{s}) = \emptyset$	$\langle u^* - 2, u^* - 2, u^* 2 \rangle$

Figure 2 The payoff functions for the game $\widehat{\mathsf{G}}$. Here, $\mathsf{P}(\mathbf{s}) := \{i \in [3] \mid s_i = 1\}$, and $u^* := \overline{u}(\widetilde{\mathsf{G}}) + 1$.

and $\widehat{\mathsf{G}}$ as the subgames to obtain the game $\mathsf{G} = \mathsf{G}\left\langle \widetilde{\mathsf{G}}, \widehat{\mathsf{G}} \right\rangle$; G is the instance of the decision problem (from the corresponding *Group*) associated with some particular property of Nash equilibria; to prove that the decision problem is $\exists\mathbb{R}$-hard, we need to establish: The game $\widetilde{\mathsf{G}}$ has a Nash equilibrium in the ball $\mathcal{B}_{1/2}$ if and only if the game G has a Nash equilibrium with the property (respectively, the set of Nash equilibria for G has the property, as for the decision problem $\exists$ SECOND NASH).

We continue with the formal proof. We treat separately each of *Group I* and *Group II*. *Group I*: Construct the 3-player gadget game $\widehat{\mathsf{G}}$ where for each player $i \in [3]$, $\widehat{\Sigma}_i = [n]$. The payoff functions are given in Figure 2: Clearly, the gadget game $\widehat{\mathsf{G}}$ is constructed in time polynomial in the size of the inbox game $\widetilde{\mathsf{G}}$. Note that $\widehat{\mathsf{G}}$ has a unique Nash equilibrium which is the profile $\mathbf{s} = (1, 1, 1)$, in which each player has payoff u^*.

Apply now the game reduction from Section 3 to construct the game G from the subgames $\widetilde{\mathsf{G}}$ and $\widehat{\mathsf{G}}$ (and an arbitrary $\delta > 0$). Since *(i)* G is constructed in time polynomial in the sizes of $\widetilde{\mathsf{G}}$ and $\widehat{\mathsf{G}}$, and *(ii)* $\widehat{\mathsf{G}}$ is constructed in time polynomial in the size of $\widetilde{\mathsf{G}}$, it follows that G is constructed in time polynomial in the size of $\widetilde{\mathsf{G}}$. Lemmas 7, 8 and 9 immediately imply:

▶ **Lemma 11.** *Assume that the inbox game $\widetilde{\mathsf{G}}$ has no Nash equilibrium in $\mathcal{B}_{1/2}$. Then, G has a unique Nash equilibrium $(\overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \overleftarrow{\sigma_3})$ with the following properties:*

1. *For each player $i \in [3]$, $\mathsf{U}_i(\overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \overleftarrow{\sigma_3}) = u^*$.*
2. *For each player $i \in [3]$, $\mathsf{Support}(\overleftarrow{\sigma_i}) = \{n(n+1)+1\}$.*
3. *$(\overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \overleftarrow{\sigma_3})$ is Pareto-Optimal.*
4. *$(\overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \overleftarrow{\sigma_3})$ is Strongly-Pareto-Optimal.*

On the other hand, Lemma 9 immediately implies:

▶ **Lemma 12.** *Assume that the inbox game $\widetilde{\mathsf{G}}$ has a Nash equilibrium in $\mathcal{B}_{1/2}$. Then, G has a Nash equilibrium $\boldsymbol{\tau}$ with the following properties:*

1. *For each player $i \in [3]$, $\mathsf{U}_i(\boldsymbol{\tau}) \leq \overline{u}(\widetilde{\mathsf{G}}) < u^*$.*
2. *For each player $i \in [3]$, $\mathsf{Support}(\tau_i) \in [n]$ with $|\mathsf{Support}(\tau_i)| \geq h$ for some integer $h \geq 2$,*
3. *$\boldsymbol{\tau}$ is not Pareto-Optimal.*
4. *$\boldsymbol{\tau}$ is not Strongly Pareto-Optimal.*

Now, combining the two families of properties in Lemmas 11 and 12 immediately yields the $\exists\mathbb{R}$-hardness of the following decision problems:

- $\exists$ SECOND NASH;
- $\exists$ NASH WITH SMALL PAYOFFS, taking u with $\overline{u}(\widetilde{\mathsf{G}}) < u \leq u^*$;
- $\exists$ NASH WITH SMALL TOTAL PAYOFF, taking u with $r \cdot \overline{u}(\widetilde{\mathsf{G}}) < u \leq r \cdot u^*$;
- $\exists$ NASH WITH LARGE SUPPORTS, taking k with $2 \leq k \leq h$;
- $\exists$ NASH WITH RESTRICTING SUPPORTS, taking a triple $(\mathsf{T}_1, \mathsf{T}_2, \mathsf{T}_3) = (\{i\}, \{j\}, \{k\})$ with arbitrary strategies $i, j, k \in [n]$;
- $\exists$ NASH WITH RESTRICTED SUPPORTS, taking, for each player $i \in [3]$, T_i such that $[n] \subseteq \mathsf{T}_i \subseteq [p] \setminus \{n(n+1)+1\}$;

- ∃ NON-PARETO-OPTIMAL NASH;
- ∃ NON-STRONGLY PARETO-OPTIMAL NASH.

Group II: Construct the 3-player gadget game $\widehat{\mathsf{G}} := \mathsf{G}[m] + \underline{u}(\widetilde{\mathsf{G}}) - 1$, where $m \geq 3$ is an odd integer with size polynomial in the size of n, and $\mathsf{G}[m]$ is the gadget game from Section 2.4. Clearly, the game $\widehat{\mathsf{G}}$ is constructed in time polynomial in the size of $\widetilde{\mathsf{G}}$. By Lemmas 3 and 4, $\widehat{\mathsf{G}}$ has a unique Nash equilibrium $\boldsymbol{\sigma}$ which is fully mixed and has $\widehat{\mathsf{U}}_i(\boldsymbol{\sigma}) = \underline{u}(\widetilde{\mathsf{G}}) - 1 + \frac{1}{m}$. Apply now the game reduction from Section 3 to construct the game G from the subgames $\widetilde{\mathsf{G}}$ and $\widehat{\mathsf{G}}$. As in *Group I,* and using the fact that m has size polynomial in that of n, G is constructed in time polynomial in the size of $\widetilde{\mathsf{G}}$. Lemmas 7, 8 and 9 immediately imply:

▶ **Lemma 13.** *Assume that the inbox game $\widetilde{\mathsf{G}}$ has no Nash equilibria in $\mathcal{B}_{1/2}$. Then, G has a unique Nash equilibrium $(\overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \overleftarrow{\sigma_3})$ with the following properties:*

1. *For each player $i \in [3]$, $\mathsf{U}_i\left(\overleftarrow{\sigma_1}, \overleftarrow{\sigma_2}, \overleftarrow{\sigma_3}\right) = \underline{u}\left(\widetilde{\mathsf{G}}\right) - 1 + \frac{1}{m}$.*
2. *For each player $i \in [3]$, $|\mathsf{Support}(\overleftarrow{\sigma_i})| = m$.*

On the other hand, Lemma 9, immediately implies:

▶ **Lemma 14.** *Assume that the inbox game $\widetilde{\mathsf{G}}$ has a Nash equilibrium in $\mathcal{B}_{1/2}$. Then, G has a Nash equilibrium $\boldsymbol{\tau}$ with the following properties:*

1. *For each player $i \in [3]$, $\mathsf{U}_i(\boldsymbol{\tau}) \geq \underline{u}(\widetilde{\mathsf{G}}) - 1 + \frac{1}{m}$.*
2. *For each player $i \in [3]$, $|\mathsf{Support}(\tau_i)| \leq n$.*

Now, combining the two families of properties in Lemmas 13 and 14 immediately yields the ∃ℝ-hardness of the following decision problems:

- ∃ NASH WITH LARGE PAYOFFS, taking u with $\underline{u}(\widetilde{\mathsf{G}}) - 1 + \frac{1}{m} < u \leq \underline{u}(\widetilde{\mathsf{G}})$.
- ∃ NASH WITH LARGE TOTAL PAYOFF, taking u with $r \cdot \left(\underline{u}(\widetilde{\mathsf{G}}) - 1 + \frac{r}{m}, \right) < u \leq r \cdot \underline{u}(\widetilde{\mathsf{G}})$.
- ∃ NASH WITH LARGE SUPPORTS, taking k with $n \leq k < m$.

Each ∃ℝ-hardness result can be extended to r-player games with $r > 3$ using a trivial technique, also used in [9]. Specifically, to prove that a particular decision problem is ∃ℝ-hard for r-player games with $r > 3$ given that it is ∃ℝ-hard for 3-player games, we reduce from 3-player games to r-player games: We add $r - 3$ dummy players; each comes with a suitable payoff function so that the property associated with the decision problem is either satisfied vacuously by the dummy players, or its satisfaction is not affected by the dummy players. For example, for ∃ NASH WITH LARGE PAYOFFS, each dummy player comes with a single strategy 1 and the payoff of each dummy player is always u no matter what the other players choose; for ∃ NASH WITH RESTRICTED SUPPORTS, each dummy player comes with a single strategy 1 and the set T_i for each dummy player $i \in [r] \setminus [3]$ is taken as $\{1\}$. For ∃ NASH WITH LARGE SUPPORTS, each dummy player comes with k strategies, each yielding the same payoff no matter what the other players choose; thus, a mixed profile where each dummy player plays each of her k strategies with probability $\frac{1}{k}$ is a Nash equilibrium when restricted to the dummy players. This implies that there is a Nash equilibrium such that for each player $i \in [r]$, $|\mathsf{Support}(\sigma_i)| \geq k$ if and only if there is a Nash equilibrium such that for each non-dummy player i, $|\mathsf{Support}(\sigma_i)| \geq k$. Further details are omitted as trivial. ◀

5 Epilogue

The extensive catalog of ∃ℝ-complete decision problems about Nash equilibria in r-player games with $r \geq 3$ we presented extends significantly the corresponding ∃ℝ-completeness results from [9, 17] and completes the picture for the complexity characterization of such

problems. Deciding any of these problems in $\mathcal{NP}$ (for r-player games with $r \geq 3$) is as hard as deciding ETR in $\mathcal{NP}$, which is considered very unlikely. The presented catalog seconds the corresponding one of $\mathcal{NP}$-complete decision problems about Nash equilibria in 2-player games from [1, 6, 10]. It remains open whether or not the established $\exists\mathbb{R}$-hardness survives the restriction to r-player *win-lose* games with $r \geq 3$; we note that the corresponding $\mathcal{NP}$-hardness for 2-player games [6, 10] was extended to 2-player win-lose games in [1].

Acknowledgements. This work was partially supported by the PRIN 2010–2011 research project ARS TechnoMedia: "Algorithmics for Social Technological Networks" funded by the Italian Ministry of Education, Universities and Research and by research funds at the University of Cyprus.

References

1 V. Bilò and M. Mavronicolas. The complexity of decision problems about Nash equilibria in win-lose games. In *Proc. of the 5th Inte'l Symposium on Algorithmic Game Theory*, volume 7615 of *LNCS*, pages 37–48, 2012.

2 V. Bilò and M. Mavronicolas. Complexity of rational and irrational Nash equilibria. *Theory of Computing Systems*, 54(3):491–527, 2014.

3 V. Bonifaci, U. Di Orio, and L. Laura. The complexity of uniform Nash equilibria and related subgraph problems. *Theoretical Computer Science*, 401(1–3):144–152, 2008.

4 J. Canny. Some algebraic and geometric computations in $\mathcal{PSPACE}$. In *Proc. of the 20th Annual ACM Symp. on Theory of Computing*, pages 460–467, 1988.

5 X. Chen, X. Deng, and S.-H. Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56(3), 2009.

6 V. Conitzer and T. Sandholm. New complexity results about Nash equilibria. *Games and Economic Behavior*, 63(2):621–641, 2008.

7 C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

8 M. J. Garey and D. J. Johnson. *Computers and Intractability – A Guide to the Theory of $\mathcal{NP}$-Completeness*. W. H. Freeman, 1979.

9 J. Garg, R. Mehta, V. V. Vazirani, and S. Yazdanbod. $\mathcal{ETR}$-completeness for decision versions of multi-player (symmetric) Nash equilibria. In *Proc. of the 42nd Int'l Colloquium on Automata, Languages and Programming*, volume 9134 of *LNCS*, pages 554–566, 2015.

10 I. Gilboa and E. Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1(1):80–93, 1989.

11 N. E. Mnev. The universality theorems on the classification problem of configuration varieties and convex polytopes varieties. In *Topology and Geometry – Rohlin Seminar*, number 1346 in LNM, pages 527–543, 1988.

12 J. F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36:48–49, 1950.

13 J. F. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.

14 C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994.

15 M. Schaefer. Complexity of some geometric and topological problems. In *Proc. of the 17th Int'l Symp. on Graph Drawing*, volume 5849 of *LNCS*, pages 334–344, 2010.

16 M. Schaefer. Realizability of graphs and linkages. In *Thirty Essays on Geometric Graph Theory*, 2013.

17 M. Schaefer and D. Štefankovič. Fixed points, Nash equilibria and the existential theory of the reals. *Theory of Computing Systems*, First online: 04 November 2015.

18 A. Tarski. A decision method for elementary algebra and geometry. *RAND Corporation*, 1948.

Multiple-Edge-Fault-Tolerant Approximate Shortest-Path Trees*

Davide Bilò[1], Luciano Gualà[2], Stefano Leucci[3], and Guido Proietti[4]

1 **DUMAS, Università di Sassari, Italy**
davide.bilo@uniss.it
2 **DII, Università di Roma "Tor Vergata", Italy**
guala@mat.uniroma2.it
3 **DISIM, Università degli Studi dell'Aquila, Italy; and DI, "Sapienza" Università di Roma, Italy**
Stefano.leucci@univaq.it
4 **DISIM, Università degli Studi dell'Aquila, Italy; and IASI, CNR, Roma, Italy**
guido.proietti@univaq.it

Abstract

Let G be an n-node and m-edge positively real-weighted undirected graph. For any given integer $f \geq 1$, we study the problem of designing a sparse *f-edge-fault-tolerant* (f-EFT) *σ-approximate single-source shortest-path tree* (σ-ASPT), namely a subgraph of G having as few edges as possible and which, following the failure of a set F of at most f edges in G, contains paths from a fixed source that are stretched at most by a factor of σ. To this respect, we provide an algorithm that efficiently computes an f-EFT $(2|F|+1)$-ASPT of size $O(fn)$. Our structure improves on a previous related construction designed for *unweighted* graphs, having the same size but guaranteeing a larger stretch factor of $3(f+1)$, plus an additive term of $(f+1)\log n$.

Then, we show how to convert our structure into an efficient f-EFT *single-source distance oracle* (SSDO), that can be built in $\widetilde{O}(fm)$ time, has size $O(fn\log^2 n)$, and is able to report, after the failure of the edge set F, in $O(|F|^2\log^2 n)$ time a $(2|F|+1)$-approximate distance from the source to any node, and a corresponding approximate path in the same amount of time plus the path's size. Such an oracle is obtained by handling another fundamental problem, namely that of updating a *minimum spanning forest* (MSF) of G after that a *batch* of k simultaneous edge modifications (i.e., edge insertions, deletions and weight changes) is performed. For this problem, we build in $O(m\log^3 n)$ time a *sensitivity* oracle of size $O(m\log^2 n)$, that reports in $O(k^2\log^2 n)$ time the (at most $2k$) edges either exiting from or entering into the MSF. As a result of independent interest, it is worth noticing that our MSF oracle can be employed to handle arbitrary sequences of $o(\sqrt[4]{n}/\log n)$ (non-simultaneous) updates with a worst-case time per update of $o(\sqrt{n})$. Thus, for relatively short sequences of updates, our oracle should be preferred w.r.t. the best-known (in a worst-case sense) MSF *fully-dynamic* algorithm, requiring $O(\sqrt{n})$ time per update.

1998 ACM Subject Classification G.2.2 [Graph Theory] Graph algorithms, Trees

Keywords and phrases fault-tolerant shortest-path tree, distance oracle, minimum spanning tree

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.18

* This work was partially supported by the Research Grant PRIN 2010 "ARS TechnoMedia", funded by the Italian Ministry of Education, University, and Research.

© Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti;
licensed under Creative Commons License CC-BY

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 18; pp. 18:1–18:14

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

1 Introduction

Let $G = (V(G), E(G), w)$ be a positively real-weighted undirected graph of n nodes and m edges. A *shortest-path tree* (SPT) of G rooted at a distinguished source vertex, say s, is one of the most popular structures in communication networks. For example, it can be used for implementing the fundamental *broadcasting* operation. However, the SPT, as any tree-based topology, is highly sensitive to edge/vertex failures, which cause the undesired effect of disconnecting sets of vertices from the source.

Therefore, a general approach to cope with this scenario is to make the SPT resistant against a given number of component failures, by adding to it a set of suitably selected edges from the underlying graph, so that the resulting structure will still contain an SPT of the surviving network. If we prepare ourselves to resist against a set of at most f failing edges in G, then the corresponding structure will be named an *f-edge-fault-tolerant* (f-EFT) SPT. Unfortunately, it can be seen that even if $f = 1$ and $m = \Theta(n^2)$, then $\Theta(m)$ additional edges may be needed, as will be shown in the full version of this paper. Thus, to sparsify such a structure, it makes sense to resort to *approximate* shortest paths from the source, that are stretched at most by a factor $\sigma > 1$, for any possible set of failures that has to be handled.

In this paper, we show how to build[1] an efficient structure of this sort. Moreover, we show that it is possible to transform such a structure into an efficient *oracle* that will allow to quickly switch to the alternative paths in case a set of failures will take place.

1.1 Related Work

In the recent past, several single and multiple edge/vertex-fault-tolerant *approximate* SPT (ASPT) structures have been devised. More formally, we say that a spanning subgraph H of G is an f-EFT σ-ASPT if it satisfies the following condition: For each set of edges $F \subseteq E(G)$ of size at most f, all the distances from the source s in the subgraph $H-F = (V(G), E(H)\setminus F, w)$ are at most σ times longer than the corresponding distances in $G - F$. Similar definitions can be given for the *vertex-fault-tolerant* (VFT) case.

A natural counterpart of fault-tolerant SPT structures are fault-tolerant *σ-stretched single-source distance oracles* (σ-SSDO in the following), i.e., *compact* data structures that can be built with a *low* preprocessing time, and that are able to *quickly* return σ-approximate distances/paths from the source following a set of failures. Converting a fault-tolerant SPT into a corresponding SSDO with the very same stretch, and additionally having a small size and a fast query time, is a quite natural process, because of its practical usage: computing the alternative post-failure distances/paths on the structure may indeed be very time consuming. However, such a conversion process is not straightforward, in general, since it requires to exploit distance-related information that are instead implicit in the underlying structure, and this has to be done by optimizing the trade-off between the size and the query time of the oracle.

Turning back our attention to fault-tolerant SPT structures, their study originated in [4], where the authors built in $O(m \log n + n \log^2 n)$ time a 1-VFT 3-SSDO of size $O(n \log n)$, and, for *unweighted* graphs, in $O(m\sqrt{n/\varepsilon})$ time a 1-VFT $(1+\varepsilon)$-SSDO of size $O(\frac{n}{\varepsilon^3} + n \log n)$, for any $\varepsilon > 0$, both having a distance (resp., path) query time of $O(1)$ (resp., proportional to the path's size). In such a paper, the authors observe explicitly that, as a result of

[1] Throughout this introduction, all the discussed structures are poly-time computable, even if we may omit to specify the actual running time.

independent interest, the latter oracle (but actually the former as well) can be converted into a corresponding structure (i.e., a spanning subgraph), having the same size and stretch. For the weighted case, the obtained 1-VFT 3-ASPT of size $O(n \log n)$ was then substantially improved in [8], where the authors showed the existence of a 1-E/VFT $(1+\varepsilon)$-ASPT of size $O(\frac{n \log n}{\varepsilon^2})$, for any $\varepsilon > 0$ (without providing a corresponding oracle).

Concerning unweighted graphs, Parter and Peleg in [30] presented a 1-E/VFT *Breadth-First Search tree* (BFS) of size $O(n \cdot \min\{ecc(s), \sqrt{n}\})$, where $ecc(s)$ denotes the eccentricity of the source vertex s in G, namely a structure containing *exact* shortest paths from the source after a single edge/vertex failure. In the same paper, the authors also exhibit a corresponding lower bound of $\Omega(n^{3/2})$ for the size of a 1-E/VFT BFS. Then, in [31], the same authors presented a set of lower and upper bounds to the size of fault-tolerant (σ, β)-ABFS, where a further additive distortion β is allowed to the distances. More precisely, they showed that for every $\beta \in [1, O(\log n)]$, there exists a graph G and a source vertex $s \in V(G)$ such that a corresponding 1-EFT $(1, \beta)$-ABFS requires $\Omega(n^{1+\varepsilon(\beta)})$ edges, for some function $\varepsilon(\beta) \in (0, 1)$. Moreover, they also constructed a 1-EFT $(1, 4)$-ABFS of size $O(n^{4/3})$. Finally, assuming at most $f = O(1)$ edge failures can take place, they showed the existence of (i) an f-EFT $(3(f+1), (f+1)\log n)$-ABFS of size $O(fn)$, and (ii) an f-EFT $(3f+4)$-ABFS of size $O(fn \log^{f+1} n)$. These structures will be exactly our touchstone in this paper, since they are the only ones concerned with multiple-edge-failure single-source shortest paths.

1.2 Our Results

In this paper, we present the following main results:

- An f-EFT $(2|F|+1)$-ASPT of size $O(fn)$ that is able to handle the failure of any set $F \subseteq E(G)$ of at most f edges. This considerably improves w.r.t. to its direct competitors, namely the structures presented in [31]: our structure has a size that is never worse, a lower stretch, works on weighted graphs, and handles an arbitrary (i.e., even non-constant) number of failures. Moreover, our construction is simpler and can be computed quickly in $O(fm\,\alpha(m, n))$ time, where α is the inverse of the Ackermann's function.
- A corresponding f-EFT $(2|F|+1)$-SSDO of size $O(\min\{m, fn\} \log^2 n)$, that has a query time for a post-failure distance from the source of $O(|F|^2 \log^2 n)$, and is also able to report the corresponding path in the same time plus the path size. The preprocessing time is $O(fm\,\alpha(m, n) + fn \log^3 n)$. Notice that if one is willing to use $O(m \log^2 n)$ space, then our oracle will be prepared to handle any number of edge failures (i.e., up to m).

Interestingly enough, the former result is obtained by posing a simple yet surprising relationship between the structure of the replacement paths and the *minimum spanning forest* (MSF) of an ad-hoc auxiliary graph. This approach is also useful to develop the latter result, that is indeed obtained through an efficient updating of an MSF after that a *batch* of k edge modifications (i.e., edge insertions, deletions and weight changes) are simultaneously performed. For this problem indeed we provide the following result:

- a *sensitivity oracle*[2] of size $O(m \log^2 n)$, that can be built in $O(m \log^3 n)$ time, and is able to report in $O(k^2 \log^2 n)$ time the (at most $2k$) edges either exiting from or entering into the MSF. As a result of independent interest, it is worth noticing that our oracle can be used to efficiently maintain an MSF under relatively short sequences of *non-simultaneous* updates. Indeed, observe that a sequence $\lambda = \langle \lambda_1, \ldots, \lambda_h \rangle$ of updates can

[2] We use this noun for the oracle in accordance with its functionality of only reporting the updates in the MSF.

be managed through h sequential queries to the oracle, where the i-th query will involve the modifications to the starting MSF induced by the batch of the first i updates. This way, we spend $O(h^2 \log^2 n)$ time to handle each single update. Hence, as the fastest long-standing algorithm for the classic (and clearly more general) *fully-dynamic* MSF problem has a worst-case cost of $O(\sqrt{n})$ per update [20], it follows that for $h = o(\sqrt[4]{n}/\log n)$, our oracle should be preferred, since it will manage each update in $o(\sqrt{n})$ time. Notice also that a comparison with other known online/offline algorithms for maintaining an MSF that are more efficient in an amortized sense, like for instance those given in [19, 25, 26], is unfeasible, since they need to start from an empty graph to guarantee their bounds (or, they need long sequences of updates to become efficient). Thus, when starting from an arbitrary graph, as it happens in our setting, a single update operation could even cost them $\Theta(n)$ time!

Finally, we point out that we are also able to prove a lower bound of $\Omega(n^{1+\frac{1}{k}})$ on the size of any f-EFT σ-ASPT with $f \geq \log n$ and $\sigma < \frac{3k+1}{k+1}$, that holds if the Erdős' girth conjecture is true. Our lower bound shows that, in contrast to the single-edge failure case, it is not possible to obtain a stretch arbitrary close to 1 with size $\widetilde{O}(n)$ when the number of faults is more than $\log n$. We look at the problem of understanding if this can be done for constant $f > 1$ as an interesting open problem. Due to space limitations, this result will be given in the full version of the paper.

1.3 Other Related Work on Fault-Tolerant Single/Multiple-Source Structures/Oracles

Besides the papers mentioned before, several other research efforts have been devoted to structures and oracles for tolerating single/multiple failures in single-source shortest paths. An early work on the topic is [27], where the authors were concerned with the computation of *best swap edges* (w.r.t. several swap functions) for the failure of each and every edge in an SPT. As a by-product of their results, it can be easily seen that by adding to an SPT the (at most) $n-1$ best swap edges w.r.t. to the new distance from s to the root of the subtree disconnected from s after an edge failure, then a 1-EFT 3-ASPT is obtained. Interestingly, such a structure can be easily converted into a 1-EFT 3-SSDO of size $O(n)$ and query time $O(1)$. Recently, in [15], the authors faced the special case of *shortest-path failures*, in which the failure of a set F of at most f adjacent edges along any source-leaf path has to be tolerated. They proposed an f-EFT $(2k-1)(2|F|+1)$-ASPT of size $O(kn\, f^{1+1/k})$, where $|F|$ denotes the size of the actual failing path, and $k \geq 1$ is a parameter of choice. Notice that this result is subsumed by ours. Moreover, they also provided a conversion to a corresponding oracle, and for the special case of $f = 2$, they gave an ad-hoc solution of size $O(n \log n)$ and with stretch 3. For directed graphs with integer positive edge weights bounded by M, in [23] the authors showed how to build efficiently in $\widetilde{O}(Mn^{\omega})$ time a randomized 1-EFT 1-SSDO of size $\Theta(n^2)$ and with $O(1)$ query time, where returned distances are exact w.h.p., and $\omega < 2.373$ denotes the matrix multiplication exponent.

Concerning unweighted graphs, in [8] the authors showed that an *ordinary* (i.e., non fault-tolerant) (σ, β)-*spanner* (i.e., where distances/paths between arbitrary pairs of nodes are at most (σ, β)-stretched) of size $O(g(n))$ can be used to build a 1-EFT (resp., VFT) (σ, β)-ABFS of the same size (resp., of size $O(g(n) + n \log n)$). This result is useful for building sparse 1-VFT $(1, \beta)$-ABFS structures by making use of the vast literature on *additive* $(1, \beta)$-spanners (e.g., [5, 11]). Finally, Parter in [29] presented a 2-EFT BFS having $O(n^{5/3})$ edges, which is tight.

Another research stream related to our work is that on *multi-source* (MS) fault-tolerant structures, for which we look at distances/paths from a set $S \subseteq V(G)$ of sources. Here, results are known only for unweighted graphs. In [30] the authors gave an algorithm to compute a 1-EFT MSBFS of size $O(\sqrt{|S|}\, n^{3/2})$, which is tight. Then, in [8] it was shown that an ordinary (σ, β)-spanner of size $O(g(n))$ can be used to build a 1-EFT (σ, β)-AMSBFS of size $O(g(n) + n\,|S|)$, and similarly for the vertex case of size $O(g(n) + n \log n|S|)$.

1.4 More Related Work on (Fault-Tolerant) Spanners/Oracles

For the sake of completeness, we also give some hints on the large body of literature on the related topic of (fault-tolerant) spanners and distance oracles.

On weighted graphs, the currently best known construction is, for any $f \geq 1$ and any integer parameter $k \geq 1$, the f-EFT (resp., VFT) $(2k-1)$-spanner of size $O(f\, n^{1+1/k})$ (resp., $\widetilde{O}(f^2\, k^{f+1}\, n^{1+1/k})$) given in [13]. For the vertex-failure case, this has been then improved in a randomized sense in [16], where the expected size was reduced to $\widetilde{O}(f^{2-1/k}\, n^{1+1/k})$. For a comparison, the sparsest known $(2k-1)$-multiplicative ordinary spanner has size $O(n^{1+1/k})$ [2], and this is believed to be asymptotically tight due to the girth conjecture of Erdős [21]. Then, in [3] it was introduced the resembling concept of 1-EFT *resilient* spanners, i.e., spanners such that whenever any edge in G fails, then the relative distance increases in the spanner are very close to those in G.

Ordinary (i.e., fault-free) *all-pairs distance oracles* (APDO) on weigthed graphs were introduced in a seminal work by Thorup and Zwick [32] (who also coined the term *oracle*), followed by a sequel of papers (among the others, we mention [12, 18] for the currently best bounds). In a fault-tolerant setting, in [6] the authors built (on directed graphs) a 1-E/VFT 1-APDO of size $\widetilde{O}(n^2)$ and with query time $O(1)$. For two failures, in [17] the authors built, still on directed graphs, a 2-E/VFT 1-APDO of size $\widetilde{O}(n^2)$ and with query time $O(\log n)$. Concerning multiple-edge failures, in [14] the authors built, for any integer $k \geq 1$, an f-EFT $(8k-2)(f+1)$-APDO of size $O(fk\, n^{1+1/k} \log(nW))$, where W is the ratio of the maximum to the minimum edge weight in G, and with a query time of $\widetilde{O}(|F| \log\log d)$, where F is the actual set of failing edges, and d is the distance between the queried pair of nodes in $G - F$.

On unweighted graphs, it makes instead sense to study fault-tolerant additive spanners. In particular, Braunshvig et al. [9] proposed the following general approach to build an f-EFT additive spanner: Let A be an f-EFT σ-spanner, and let B be an ordinary $(1, \beta)$-spanner. Then $H = A \cup B$ is an f-EFT $(1, 2f(2\beta + \sigma - 1) + \beta)$-spanner. Recently, in [7] the corresponding analysis has been refined yielding a better additive bound of $2f(\beta + \sigma - 1) + \beta$. Finally, for other results on single edge/vertex failures spanners/oracles on unweighted graphs, we refer the reader to [4, 28, 7].

2 An f-EFT $(2|F|+1)$-ASPT and a Corresponding Oracle

In this section we show how to compute an f-EFT $(2|F|+1)$-ASPT H of G. When up to f edges can fail, it is easy to see that whenever G is $(f+1)$-edge-connected, H must contain $\Omega(fn)$ edges even if we are only interested in preserving the connectivity of G, since the degree of each vertex must be at least equal to $f+1$. Here we show that $|E(H)| = O(fn)$ edges also suffice if we seek to preserve distances that are at most $(2f+1)$-stretched w.r.t. the surviving part of G.

Let $d_X(u, u')$ and $\pi_X(u, u')$ denote the distance and the shortest path between nodes u and u' in any subgraph X of G, respectively. When $u = s$, we will simply write $d_X(u')$ and $\pi_X(u')$. If π is a path, $\pi[u, u']$ will denote the subpath of π between $u, u' \in V(\pi)$.

Algorithm 1: Algorithm for computing an f-EFT $(2|F|+1)$-ASPT of G.

```
T ← compute an SPT of G
for (u, v) ∈ E(G) do
    if (u, v) ∈ E(T) then w'(u, v) ← 0
    else w'(u, v) ← d_T(u) + w(u, v) + d_T(v)
G' ← (V(G), E(G), w')
G_0 ← G'
for i = 0, ..., f do
    M_i ← edges of an MSF of G_i (w.r.t. w')
    G_{i+1} ← G_i \ M_i
H ← subgraph of G containing the edges in ⋃_{i=0}^{f} M_i
return H
```

For any given integer f, Algorithm 1 returns an f-EFT $(2|F|+1)$-ASPT of G. First, it computes an SPT T of G that is used to assign a weight to the edges of an auxiliary graph $G' = (V(G), E(G), w')$. More precisely, the weight of an edge e of G' is 0 if e is also in T, otherwise it is equal to the sum of the corresponding edge weight in G and the distances in T between s and the endpoints of e. Then, $f+1$ MSFs $M_0, \ldots, M_f$ of G' are iteratively computed: when we compute the i-th forest, we remove its edges M_i from G' before computing the $(i+1)$-th forest, so that the sets M_i are pairwise disjoint. The sought subgraph H contains all the edges of the sets M_i. Notice that M_0 coincides with $E(T)$.

We now argue that H is indeed an f-EFT $(2|F|+1)$-ASPT of G. Fix a vertex t and let $\pi = \pi_{G-F}(t)$ be the shortest path from s to t in the surviving graph $G - F$.[3] The path π traverses the vertices of several trees in the forest $T - F$. We say that an edge is *new* if its endpoints belong to two different trees in $T - F$. Let N be the set of new edges in π.

Now consider an MSF M of the graph $H - F$ (w.r.t. w'). This is also an MSF of the graph $G' - F$ (w.r.t. w') as shown by the following lemma.

▶ **Lemma 1.** *For every $F \subseteq E(G)$ with $|F| \leq f$, any MSF M of $H - F$ (w.r.t. w') is also an MSF of $G' - F$ (w.r.t. w').*

Proof. In what follows, whenever ties arise we break them by prioritizing the edges in H. First we show that, given any cut-set[4] C of G', H contains the $\min\{|C|, f+1\}$ lightest edges of C. Indeed, for any set M_i, consider the set $C_i = C \setminus \cup_{j=0}^{i-1} M_j$. Either C_i is non empty, and therefore M_i contains the lightest edge in C_i, or $C_i = \emptyset$ which means that each edge in C belongs to some set M_j and hence to H.

Let M' be an MSF of $G' - F$. We prove the claim by showing that each edge $e \in E(M')$ must also belong to M. Let C' be the cut-set of G' that contains e and every edge $e' \in E(G')$ that forms a cycle with e in $M' \cup \{e'\}$. Since e is the lightest edge of $C' \setminus F$, it is within the $f+1$ lightest edges of C'. As a consequence $e \in E(H - F)$, and it also belongs to M as it is the lightest edge in $C' \cap E(H - F)$. ◀

Let $\pi' = \pi_M(s, t)$ and notice that π' traverses each tree of the forest $T - F$ at most once since edges in $E(T)$ have weight 0 in H. Once again, let N' be the set of new edges of π'. We now provide an upper bound to the distance $d_{H-F}(t)$ using the path π':

[3] We assume that such a path exists, as otherwise $d_{G-F}(t) = +\infty$ which implies $d_{H-F}(t) = +\infty$, and we are done.

[4] A cut-set of a graph X is a subset of $E(X)$ whose removal increases the number of connected components of X.

▶ **Lemma 2.** $d_{H-F}(t) \leq w(\pi') \leq \sum_{e\in N'} w'(e) + d_G(t)$.

Proof. Let M be an MSF of the graph $H - F$ (w.r.t. w'). The first inequality is trivial as $\pi' = \pi_M(s,t)$ is a path (not necessarily shortest) between s and t in (a subgraph of) $H - F$, hence we focus on proving the second inequality.

Let $T_0, \ldots, T_h$ be the trees of $T - F$ traversed by π', in order, and let $e'_i = (v_{i-1}, u_i)$ be the new edge in π' connecting a vertex v_{i-1} of T_{i-1} to a vertex u_i of T_i. In such a way we have $N' = \{e'_1, \ldots, e'_h\}$. We call r_i the vertex in $V(T_i) \cap V(\pi')$ that has the lowest depth in T_i.[5] According to this definition, r_0 coincides with s, r_h is the lowest common ancestor between u_h and t, and r_i is the lowest common ancestor between u_i and v_i, for every $0 < i < h$.

We prove by induction on i that $w(\pi'[s, r_i]) \leq \sum_{j=1}^{i} w'(e'_j)$. The base case $i = 0$ is trivially true. Now suppose that the inductive hypothesis holds for i, we prove it also for $i + 1$:

$$\begin{aligned} w(\pi'[s, r_{i+1}]) &= w(\pi'[s, r_i]) + d_{T_i}(r_i, v_i) + w(e'_{i+1}) + d_{T_{i+1}}(u_{i+1}, r_{i+1}) \\ &\leq \sum_{j=1}^{i} w'(e'_j) + d_T(v_i) + w(e'_{i+1}) + d_T(u_{i+1}) = \sum_{j=1}^{i} w'(e'_j) + w'(e'_{i+1}) = \sum_{j=1}^{i+1} w'(e'_j). \end{aligned}$$

We now use the fact that $d_{T_h}(r_h, t) = d_T(r_h, t) = d_G(r_h, t)$ to prove the claim:

$$w(\pi') = w(\pi'[s, r_h]) + w(\pi'[r_h, t]) \leq \sum_{j=1}^{h} w'(e'_j) + d_{T_h}(r_h, t) \leq \sum_{j=1}^{h} w'(e'_j) + d_G(t). \qquad \blacktriangleleft$$

Next lemma shows that the weights of the new edges of π' are, in turn, upper bounded by the weight of some new edge of the path π.

▶ **Lemma 3.** *For each $e' \in N'$, we have $w'(e') \leq \max_{e\in N} w'(e)$.*

Proof. Let $e' = (x, y)$ be an arbitrary edge in N'. W.l.o.g., we assume that the path π' traverses the vertices s, x, y, t in this order. We recall that the path π' traverses each tree in $T - F$ at most once, i.e., all the vertices of π' that belong to the same tree in $T - F$ must be contiguous in π'. Moreover, as e' is new, x and y belong to two different trees in $T - F$.

Let Z be the set of trees of the forest $T - F$ that are traversed by the path π. Let u' be the last vertex of $\pi'[s, x]$ that belongs to a tree, say T_u, of Z (see Figure 1). Observe that u' is always defined since s belongs to some tree of Z. In a similar way, let v' be the first vertex of $\pi'[y, t]$ that belongs to a tree, say T_v, of Z. Again, observe that v' is always defined as t belongs to some tree of Z other than that containing s, and so $T_u \neq T_v$, and finally notice that $e' \in E(\pi'[u', v'])$. We know that π traverses both T_u and T_v (in some order), so we let π^* be the minimal (w.r.t. inclusion) subpath of π with one endpoint, say u, in $V(T_u)$, and the other endpoint, say v, in $V(T_v)$.

Let $N^* = E(\pi^*) \cap N$ be the set of new edges in π^*. Notice that $N^* \neq \emptyset$ as $T_u \neq \mathcal{T}_v$, and that adding the edges in N^* (weighted according to w') to M forms (at least) a cycle C containing both e' and an edge in N^*, say e^*. Since M is an MSF of $G' - F$, as shown by Lemma 1, we have that $w'(e') \leq w'(e^*) \leq \max_{e\in N} w'(e)$. ◀

Finally, next lemma relates the weights w' of the new edges of π to distances in the surviving graph $G - F$.

[5] We think of T_i as rooted in the vertex of $V(T_i)$ which is closest to s in T.

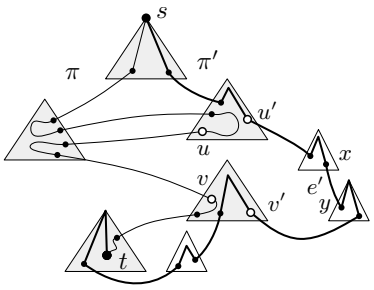


Figure 1 The forest $T - F$ obtained by deleting the failed edges in F from T. The path π is the shortest path between s and t in $G - F$, while π' (in bold) is the unique path in M between the same vertices. Gray trees contain a vertex of π and are therefore in Z. Edges having endpoints in different trees are *new*.

▶ **Lemma 4.** *For $e \in N$, $w'(e) \leq 2d_{G-F}(t)$.*

Proof. Let $e = (u, v)$ with $d_{G-F}(u) \leq d_{G-F}(v)$. Since e lies on the shortest path $\pi = \pi_{G-F}(s, t)$, we can write:

$$w'(e) = d_T(u) + w(e) + d_T(v) \leq d_{G-F}(v) + d_T(v) \leq 2d_{G-F}(v) \leq 2d_{G-F}(t). \quad \blacktriangleleft$$

We are now ready to prove the main result of this section:

▶ **Theorem 5.** *The graph H returned by Algorithm 1 is an f-EFT $(2|F|+1)$-ASPT of G. Moreover, Algorithm 1 requires $O(fm\,\alpha(m,n))$ time and $O(m)$ space.*

Proof. First, observe that $\pi' = \pi_M(s, t)$ contains at most $|F|$ new edges. Indeed all the edges in $T - F$ have weight 0, while the remaining edges have a positive weight. This means that $E(T - F) \subseteq E(M)$. As $T - F$ has no more than $|F| + 1$ connected components, we have that at most $|F|$ other edges – which are not in $E(T - F)$ – can belong to M.

By using the above fact in conjunction with Lemmas 2–4, we can write:

$$\begin{aligned} d_{H-F}(t) \leq w(\pi') &\leq \sum_{e \in N'} w'(e) + d_G(t) \leq |F| \max_{e \in N'} w'(e) + d_G(t) \\ &\leq |F| \max_{e \in N} w'(e) + d_G(t) \leq 2|F| d_{G-F}(t) + d_{G-F}(t) = (2|F|+1) d_{G-F}(t). \end{aligned} \quad (1)$$

We recall that this holds for every vertex $t \in V(G)$. Concerning the computational complexity of Algorithm 1, we make use of Chazelle's algorithm [10] – that computes an MSF in $O(m\,\alpha(m, n))$ time and linear space – to compute the $f + 1$ MSFs $M_0, \ldots, M_f$. $\blacktriangleleft$

2.1 A Corresponding Oracle

In this section we show how to build an oracle that, given a positively real-weighted graph G and a distinguished source vertex s, is able to answer queries of the form: *Given a set F of at most f edge failures, and a destination node t in G, report a $(2|F|+1)$-approximate path/distance from s to t in $G - F$.*

We first compute an SPT T of G and a f-EFT $(2|F|+1)$-ASPT H of G, as shown in the previous section. Then, the oracle is composed of three ingredients:

- the tree T and all the distances $d_T(v) = d_G(v)$ from s to any vertex $v \in V(G)$;
- an MSF sensitivity oracle Q of H w.r.t. the weights w', built as shown in Section 3;
- an oracle to answer *lowest common ancestor* (LCA) queries between two vertices in T. Such an oracle can be built in linear time and has a constant query time [24].

The resulting size is $O(fn\log^2 n)$ and the time required to build our oracle is $O(fm\,\alpha(m,n)+fn\log^3 n)$. Interestingly, if we do not know the value of f in advance, we can build, in $O(m\log^3 n)$ time, an oracle of size $O(m\log^2 n)$ that is able to report $(2|F|+1)$ approximate paths/distances, for *any* number $|F|$ of faults.

We will make use of the following additional property of our MSF oracle Q, that will be shown in Section 3: *Q can report, in $O(|F|^2\log^2 n)$ time, all the new edges (and their weights), on the unique path from s to t in the updated MSF, in order.*

Answering a Path Query

To return a $(2|F|+1)$-approximate path between s and t, it suffices to report the path $\pi' = \pi_M(s,t)$, as shown by Equation (1).

We query the MSF oracle Q for the new edges on the unique path from s to t in the updated MSF. Let $\langle e'_1,\ldots,e'_h\rangle$ be these new edges, in order, with $e'_i = (v_{i-1},u_i)$. For $0<i<h$, let r_i be the LCA between u_i and v_i, and let r_h be the LCA between u_h and t. We now have all the pieces to reconstruct and return the path π'. Indeed, if we let $\pi'_i = \pi_T(u_i,r_i)\circ\pi_T(r_i,v_i)$, the following holds:

$$\pi' = \pi_T(s,v_0)\circ e'_1\circ\pi'_1\circ e'_2\circ\pi'_2\circ\cdots\circ\pi'_{h-1}\circ e'_h\circ\pi_T(u_h,r_h)\circ\pi_T(r_h,t) \tag{2}$$

where each subpath is entirely in T and all the endpoints are known. The whole procedure requires $O(|F|^2\log^2 n)$ time to perform the query on Q, $O(|F|)$ time for the LCA queries, and $O(|\pi'|)$ time to reconstruct the path. The overall query time is therefore $O(|F|^2\log^2 n+|\pi'|)$.

Answering a Distance Query

To report the length of a $(2|F|+1)$-approximate path from s to t, we can replace each subpath in Equation (2) with the corresponding distance, in order to obtain:

$$w(\pi') = d_T(v_0)+\sum_{i=1}^{h-1}\Big(w(e'_i)+d_T(u_i,r_i)+d_T(r_i,v_i)+w(e'_{i+1})\Big)+d_T(u_h,r_h)+d_T(r_h,t).$$

The above quantity can be computed in $O(h)=O(|F|)$ time, once we know the edges $e_1,\ldots,e_h$ and we notice that $w(e'_i) = w'(e'_i)-d_T(v_{i-1})-d_T(u_i)$, and that if x is a descendant of r_i in T, then $d_T(r_i,x) = d_T(x)-d_T(r_i)$. The overall query time is thus $O(|F|^2\log^2 n)$.

3 A Minimum Spanning Tree Sensitivity Oracle

In this section we present an oracle that, given a real-weighted graph G with n vertices and m edges, along with any *minimum spanning tree* (MST) T of G, is able to answer queries of the form:

"Given a set of k edge updates on G (i.e., edge insertions, deletions and weight modifications), let T' be the new MST of G. What are the edges in the symmetric difference of $E(T)$ and $E(T')$?"[6]

In other words, the oracle can report all the edges of T that leave the MST as a consequence of the updates, along with all the new edges in T' that enter the MST in their place. The

[6] For the sake of avoiding technicalities, we assume that each edge is subject to at most a single update and we also assume that the graph G always remains connected, so that we simply talk about an MST instead of an MSF of G. For instance, this can be easily guaranteed by adding a dummy vertex $x\notin V(G)$ that is connected to all the vertices of $V(G)$ with edges of large weights.

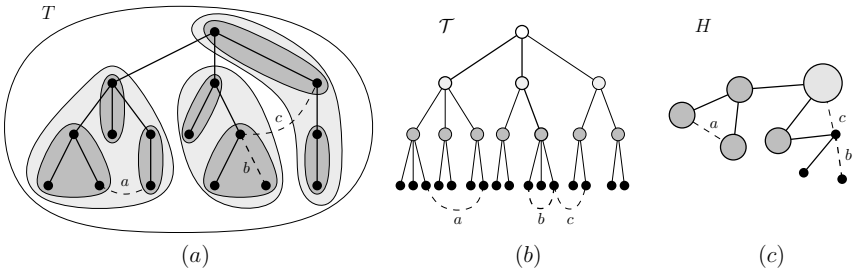


Figure 2 Hierarchical clustering of the vertices of T. (a) shows all the sets of $\mathcal{C}$, where black vertices are singletons. (b) shows the tree $\mathcal{T}$ associated with the clustering $\mathcal{C}$. (c) shows the graph H whose vertices are computed by Algorithm 2. The edges in F are depicted using dashed lines.

oracle requires $O(m \log^2 n)$ space and can be built $O(m \log^2 n)$ space and $O(m \log^3 n)$ time, while a query involving k updates can be answered in $O(k^2 \log^2 n)$ time and space.

Our oracle exploits the fact that, when few updates are to be handled, the changes in the resulting MST will be small. This implies that large portions of T and T' will coincide, and knowing these portions would allow us to save a considerable amount of work compared to the time needed to recompute T' from scratch. To this aim, we build a structure that maintains a set of connected subtrees of T at different levels of granularity.

In details, we will use a hierarchical clustering of the vertices of T. Our clustering is inspired by the construction of *topology trees* given in [22]. In [22], the author solves the *dynamic* MST problem by using a collection of topology trees that are built on top of an auxiliary graph representing shrunk components of G. We use our clustering in a different way and, as we do not need to support permanent updates of G, we are also able to simplify the construction. Due to space limitations, the full description of our construction will be given in the extended version of the paper, while here we provide a sketch of it.

We start by describing the properties of our hierarchical clustering. Let Δ be the maximum degree[7] of a vertex in T. Each cluster C will have a level $\ell(C) \in \{0, \ldots, L\}$, and we will call $\mathcal{C}_i$ the set of clusters of level i. Our clustering will guarantee that:

- **P_1.** Clusters of each level i are a partition of the vertices of T, i.e., they are pairwise disjoint and $\bigcup_{C \in \mathcal{C}_i} C = V(G)$;
- **P_2.** The vertices in each cluster induce a connected component of T;
- **P_3.** Clusters of level 0 are singletons, i.e., they contain a single vertex of T;
- **P_4.** There is only one cluster of level L (and it coincides with $V(T)$);
- **P_5.** Each cluster of level $i \geq 1$ is the union of at least 2 and at most Δ clusters of level $i-1$.

It follows from the above properties that a cluster of level i contains at least 2^i vertices, and hence $L \leq \log n$. Figure 2 (a) shows an example of such a clustering. This hierarchy can be represented by a tree $\mathcal{T}$ of height L rooted in the unique cluster in $\mathcal{C}_L$. The children of a cluster of level $i \geq 1$ in $\mathcal{T}$ are the clusters of level $i-1$ it contains (see Figure 2 (b)).

For each pair of clusters C, C' with $C \neq C'$ we maintain an ordered set $E(C, C')$ containing all the edges of $E(G)$ with one endpoint in C and the other in C'. This set is ordered according to edge weights in a non-decreasing fashion. Let $C(u)$ be the set of the $L+1$ clusters of the hierarchy (one for each level) that contain vertex u. It is easy to see that an edge

[7] In order to compute the clustering, the tree T will be rooted. We still define the degree of a vertex v in T to be the number of edges that are incident v, including the edge from v to its parent in T, if any.

Algorithm 2: Algorithm for computing the set of vertices (i.e., clusters) of H.

```
1 R ← C_L
2 for (u, v) ∈ F do
3 |   while root_T(u) = root_T(v) do
4 |   |   C ← root_T(u)
5 |   |   T ← T \ {C}
6 |   |   R ← (R \ {C}) ∪ children_T(C)                    // Split C
7 return R
```

$(u, v) \in E(G)$ appears in at most $|C(u)| \cdot |C(v)| = O(\log^2 n)$ sets, and hence the overall number of elements in the sets is at most $O(m \log^2 n)$.

We now describe how a query can be answered. In order to do so, it is useful to split each weight update operation involving an edge e into two separate operations, namely the deletion of e followed by its reinsertion with the new (updated) weight. By doing so, all the operations in F are now either insertions or deletions. For the sake of clarity, we first consider the case in which all the updates F are edge deletions, and we will show later how this can be extended to deal also with edge insertions.

Handling Edge Deletions

In order to handle deletions, we use Algorithm 2 to construct an auxiliary graph H whose vertices are clusters. The algorithm will compute a set R of clusters of $\mathcal{T}$ that will coincide with $V(H)$. Initially R contains the unique cluster in C_L that is the root of $\mathcal{T}$ and represents the whole tree T. At each time, the set of clusters in R, although of different levels, will always form a partition of the vertices of T. The algorithm proceeds iteratively, by considering one after the other the edges of F. When an edge (u, v) is considered, if u and v belong to the same cluster C of R, we *split* C, i.e., we remove C from $\mathcal{T}$ and R, and we add to R all the clusters of level $\ell(C) - 1$ contained in C. In this way $\mathcal{T}$ is always a forest and R contains the roots of its trees.

In the end, H is such that all the edges in F have their endpoints into different clusters of $V(H)$. Moreover, as each edge in F can produce at most L splits, and each split operation can increase the number of vertices by at most $\Delta - 1$, we have that H contains at most $f\, L\, (\Delta - 1) = O(\Delta\, f \log n)$ vertices (see Figure 2 (c)).

To construct the set $E(H)$ we consider all the pairs C, C' of vertices in $V(H)$. For each of these pairs we examine the edges in $E(C, C')$, in order, and we select the first edge e so that $e \notin F$, if any. Then, if e exists, we add the edge (C, C') to H with weight $w(e)$.

We can now compute an MST $\widetilde{T}$ of H in time $O(\Delta^2\, f^2 \log^2 n)$ by using any standard MST algorithm. Finally, we look at the edges of $\widetilde{T}$ and we answer the query by returning the edges in $E(\widetilde{T})$ that are not in $E(T)$. Notice also that, once $\widetilde{T}$ has been computed, it is easy to report all the edges in $E(\widetilde{T}) \setminus E(T)$ that belong to the unique path between any two vertices in the updated MST. This kind of query can still be answered in $O(\Delta^2\, f^2 \log^2 n)$ time, and it is needed by our fault-tolerant ASPT oracle of Section 2.1.

Handling General Edge Updates

It turns out that the complexity of the problem lies in handling the edge-deletion operations. Indeed, once this has been done, the remaining edge-insertion operations can be easily performed. To this aim, we reorganize the batch by first performing all the delete operations, and we make use of a *top-tree* [1], i.e., a data structure that dynamically maintains a

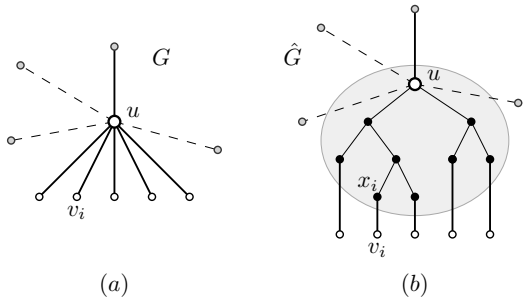


Figure 3 Reducing the degree of vertices in T: on the left side, the tree T (solid edges) embedded in G, on the right side the superimposition of the binary tree to T in order to get a maximum degree of 3. Thin solid edges have weight 0, while the weight of (x_i, v_i) is $w(u.v_i)$.

(weighted) forest under edge-insertion (*link*) and edge-deletion (*cut*) operations. Moreover, given two vertices u and v, top-trees are able to report the heaviest edge that lies on the path between u and v in the current forest. Each of these operations can be performed in $O(\log \eta)$ time where η is the number of vertices of the forest.

The idea is to maintain the current MST T' by using a top-tree that is initialized when the oracle is built to represent the tree T. This takes $O(n \log n)$ time. Then, we perform all the edge-deletion operations (as already described), while updating the top-tree accordingly (this requires $O(|F| \log n)$ time since the number of needed link and cut operations is $O(|F|)$).

Now we handle the insertions one by one. In order to insert a new edge $e = (u, v)$, we search for the heaviest edge e' of the path connecting u and v in T'. If e' is heavier than e, we cut e' from T' and we link the two resulting components by adding the edge e. It is easy to see that this procedure requires an overall time of $O(|F| \log n)$.

By keeping track of all the $O(|F|)$ updates in the MST T', we can easily answer a query consisting of both edge-insertion and edge-deletion operations in $O(\Delta^2 f^2 \log^2 n)$ time.

Reducing the Degree of T

So far, the complexity of our oracle depends on the maximum degree Δ of a vertex in T. However, using standard techniques (see, e.g., [22]), we now show that the updates on the original graph G and its MST T can be mapped onto an auxiliary graph $\hat{G}$ with weight function $\hat{w}$ and a corresponding MST $\hat{T}$, such that $\hat{G}$ has asymptotically the same size of G, and each vertex of $\hat{G}$ has a degree at most 3 in $\hat{T}$.

Initially $\hat{G}$, $\hat{w}$, and $\hat{T}$ coincide with G, w, and T, respectively. We iteratively search for a vertex u in $\hat{T}$ that has more than 2 children, and we lower its degree. Let $\text{children}_{\hat{T}}(v) = \{v_1, \ldots, v_h\}$, we proceed as follows: we remove all the edges in $\{(u, v_i) : 1 \leq i \leq h\}$ from both $\hat{G}$ and $\hat{T}$, then we add to both $\hat{G}$ and $\hat{T}$ a binary tree whose root coincides with u, and that has exactly h leaves $x_1, \ldots, x_h$. We assign weight $\hat{w}(e) = 0$ to all the edges e of this tree. Finally, we add to $\hat{G}$ and $\hat{T}$ an edge (x_i, v_i) for each $1 \leq i \leq h$, and we set $\hat{w}(x_i, v_i) = w(u, v_i)$. An example of such a transformation is shown in Figure 3.

Each time we have to perform a weight update or delete operation on an edge (u, v_i) of G, we instead perform it on the corresponding edge (x_i, v_i). Insertions and operations involving edges in $E(G) \setminus E(T)$ do not require any special care. In a similar way, whenever the answer of a query contains an edge (x_i, v_i), we replace it with the corresponding edge (u, v_i). Clearly, $O(n)$ vertices and edges are added by this process, and hence $|V(G)| = \Theta(|V(\hat{G})|)$ and $|E(G)| = \Theta(|E(\hat{G})|)$.

Once the maximum degree of the tree has been reduced to a constant, the query time of our oracle becomes $O(f^2 \log^2 n)$. To achieve such a query time, however, we must be careful in our implementation as it will be discussed in the full version of the paper.

References

1. Stephen Alstrup, Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Transactions on Algorithms*, 1(2):243–264, 2005. URL: http://doi.acm.org/10.1145/1103963.1103966, doi:10.1145/1103963.1103966.
2. Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993. URL: http://dx.doi.org/10.1007/BF02189308, doi:10.1007/BF02189308.
3. Giorgio Ausiello, Paolo Giulio Franciosa, Giuseppe Francesco Italiano, and Andrea Ribichini. On resilient graph spanners. In *ESA*, pages 85–96, 2013.
4. Surender Baswana and Neelesh Khanna. Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs. *Algorithmica*, 66(1):18–50, 2013. URL: http://dx.doi.org/10.1007/s00453-012-9621-y, doi:10.1007/s00453-012-9621-y.
5. Surender Baswana, Kavitha Telikepalli, Kurt Mehlhorn, and Seth Pettie. Additive spanners and (alpha, beta)-spanners. *ACM Transactions on Algorithms*, 7(1):5, 2010. URL: http://dx.doi.org/10.1145/1868237.1868242, doi:10.1145/1868237.1868242.
6. Aaron Bernstein and David R. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *STOC*, pages 101–110, 2009.
7. Davide Bilò, Fabrizio Grandoni, Luciano Gualà, Stefano Leucci, and Guido Proietti. Improved purely additive fault-tolerant spanners. In *ESA*, pages 167–178, 2015.
8. Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Fault-tolerant approximate shortest-path trees. In *ESA*, pages 137–148, 2014. URL: http://dx.doi.org/10.1007/978-3-662-44777-2_12, doi:10.1007/978-3-662-44777-2_12.
9. Gilad Braunschvig, Shiri Chechik, and David Peleg. Fault tolerant additive spanners. In *WG*, pages 206–214, 2012. URL: http://dx.doi.org/10.1007/978-3-642-34611-8_22, doi:10.1007/978-3-642-34611-8_22.
10. Bernard Chazelle. A minimum spanning tree algorithm with inverse-ackermann type complexity. *J. ACM*, 47(6):1028–1047, 2000. URL: http://doi.acm.org/10.1145/355541.355562, doi:10.1145/355541.355562.
11. Shiri Chechik. New additive spanners. In *SODA*, pages 498–512, 2013. URL: http://dx.doi.org/10.1137/1.9781611973105.36, doi:10.1137/1.9781611973105.36.
12. Shiri Chechik. Approximate distance oracles with constant query time. In *STOC*, pages 654–663, 2014. URL: http://doi.acm.org/10.1145/2591796.2591801, doi:10.1145/2591796.2591801.
13. Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. Fault-tolerant spanners for general graphs. In *STOC*, pages 435–444, 2009.
14. Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. f-sensitivity distance oracles and routing schemes. In *ESA*, pages 84–96, 2010.
15. Annalisa D'Andrea, Mattia D'Emidio, Daniele Frigioni, Stefano Leucci, and Guido Proietti. Path-fault-tolerant approximate shortest-path trees. In *SIROCCO*, pages 224–238, 2015. URL: http://dx.doi.org/10.1007/978-3-319-25258-2_16, doi:10.1007/978-3-319-25258-2_16.
16. Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: better and simpler. In *PODC*, pages 169–178, 2011.

17 Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *SODA*, pages 506–515, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496826.

18 Michael Elkin and Seth Pettie. A linear-size logarithmic stretch path-reporting distance oracle for general graphs. In *SODA*, pages 805–821, 2015. URL: http://dx.doi.org/10.1137/1.9781611973730.55, doi:10.1137/1.9781611973730.55.

19 David Eppstein. Offline algorithms for dynamic minimum spanning tree problems. *J. Algorithms*, 17(2):237–250, 1994. URL: http://dx.doi.org/10.1006/jagm.1994.1033, doi:10.1006/jagm.1994.1033.

20 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification-a technique for speeding up dynamic graph algorithms (extended abstract). In *FOCS*, pages 60–69, 1992. URL: http://dx.doi.org/10.1109/SFCS.1992.267818, doi:10.1109/SFCS.1992.267818.

21 Paul Erdős. Extremal problems in graph theory. In *Theory of Graphs and its Applications*, pages 29–36, 1964.

22 Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985. URL: http://dx.doi.org/10.1137/0214055, doi:10.1137/0214055.

23 Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *FOCS*, pages 748–757, 2012.

24 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984. URL: http://dx.doi.org/10.1137/0213024, doi:10.1137/0213024.

25 Monika Rauch Henzinger and Valerie King. Maintaining minimum spanning forests in dynamic graphs. *SIAM J. Comput.*, 31(2):364–374, 2001. URL: http://dx.doi.org/10.1137/S0097539797327209, doi:10.1137/S0097539797327209.

26 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001. URL: http://doi.acm.org/10.1145/502090.502095, doi:10.1145/502090.502095.

27 Enrico Nardelli, Guido Proietti, and Peter Widmayer. Swapping a failing edge of a single source shortest paths tree is good and fast. *Algorithmica*, 35(1):56–74, 2003.

28 Merav Parter. Vertex fault tolerant additive spanners. In *DISC*, pages 167–181, 2014. URL: http://dx.doi.org/10.1007/978-3-662-45174-8_12, doi:10.1007/978-3-662-45174-8_12.

29 Merav Parter. Dual failure resilient BFS structure. In *PODC*, pages 481–490, 2015.

30 Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In *ESA*, pages 779–790, 2013. URL: http://dx.doi.org/10.1007/978-3-642-40450-4_66, doi:10.1007/978-3-642-40450-4_66.

31 Merav Parter and David Peleg. Fault tolerant approximate BFS structures. In *SODA*, pages 1073–1092, 2014.

32 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. URL: http://doi.acm.org/10.1145/1044731.1044732, doi:10.1145/1044731.1044732.

On a Fragment of AMSO and Tiling Systems

Achim Blumensath[*1], **Thomas Colcombet**[†2], **and Paweł Parys**[‡3]

1 **Faculty of Informatics, Masaryk University, Czech Republic**
`blumensath@fi.muni.cz`
2 **CNRS, LIAFA, Université Paris Diderot, Paris 7, France**
`thomas.colcombet@liafa.univ-paris-diderot.fr`
3 **Institute of Informatics, University of Warsaw, Poland**
`parys@mimuw.edu.pl`

Abstract

We prove that satisfiability over infinite words is decidable for a fragment of asymptotic monadic second-order logic. In this fragment we only allow formulae of the form $\exists t \forall s \exists r\, \varphi(r, s, t)$, where φ does not use quantifiers over number variables, and variables r and s can be only used simultaneously, in subformulae of the form $s < f(x) \leq r$.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases monadic second-order logic, boundedness, tiling problems

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.19

1 Introduction

This paper continues a line of research trying to find logics where satisfiability is decidable over infinite words (and infinite trees). The most well-known logic of this kind is monadic second-order logic (MSO) considered in the seminal work of Büchi [8]. Extending MSO by the ability of comparing some quantities quickly leads to undecidability. The idea behind the logic MSO+U and a more recently introduced logic called *asymptotic monadic second-order logic* (AMSO) is to extend MSO by the ability to express boundedness properties of some sequences of numbers. In MSO+U this is realized by an additional quantifier U stating that there are arbitrarily large finite sets satisfying the given formula. AMSO does not have a built in ability to refer to the size of sets. Instead, it describes weighted structures (in particular weighted infinite words), which are structures in which the elements are labelled by natural numbers, called their *weights*. More precisely, AMSO extends MSO by quantifiers over variables of a new kind, ranging over natural numbers. These variables can be compared with weights in the word, but only under a certain positivity requirement: existentially quantified numbers can only serve as upper bounds, while universally quantified numbers can only serve as lower bounds. The two logics MSO+U and AMSO happen to be inter-reducible as far as the decidability of satisfiability is concerned [1], and, unfortunately, this means that both are undecidable over infinite words [5]. Nevertheless, some natural fragments of these logics remain decidable.

* Work partially supported by the German Science Foundation, grant No. BL 1127/2-2, and the Czech Science Foundation, grant No. P202/12/G061.
† The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 259454.
‡ Work supported by the fellowship of the Foundation for Polish Science, during the author's post-doc stay at Université Paris Diderot.

© Achim Blumensath, Thomas Colcombet, and Paweł Parys;
licensed under Creative Commons License CC-BY

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 19; pp. 19:1–19:14

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

In [2] the satisfiability problem for MSO+U is solved over infinite trees for formulae where the quantifier U is at the outermost position. A significantly more powerful fragment of the logic, although over infinite words, was shown to be decidable in [4] using automata with counters. These automata were further developed into the theory of regular cost functions [10]. Another possibility is to consider the weak fragment of the logic (WMSO+U), where set quantification is restricted to finite sets. Satisfiability for this logic was shown to be decidable over infinite words [3] and infinite trees [6]. Note that the mentioned decidability results can be used to solve, via reductions, several seemingly unrelated problems, among others: the star height problem [14], the finite power property problem [17], deciding properties of CTL* [9], the realizability problem for prompt LTL [15], deciding the winner in cost parity games [12], or deciding certain properties of energy games [7].

Concerning AMSO, which was more recently introduced [1], so far no fragments are known to be decidable (except trivial ones). Such fragments should, at least, circumvent the arguments of undecidability of AMSO, that involve complicated number quantifiers nested inside complicated quantification over infinite sets. There are two ways to avoid this: either to consider the weak fragment (WAMSO), where set quantification is restricted to finite sets, or to consider the number-prenex fragment ($\mathrm{AMSO}^{\mathrm{np}}$), where number quantifiers are required to be placed only at the head of the formula. It turns out that these two fragments are inter-reducible (Theorem 5 in [1]). It is conjectured that these two fragments have a decidable satisfiability problem over infinite words. Under a topological point of view, it is known that MSO+U and AMSO inhabit all finite levels of the projective hierarchy [13, 1], while WAMSO is much simpler since it only inhabits the finite levels of the Borel hierarchy.

et us emphasize the fact that WAMSO is not related at all to WMSO+U, even though AMSO and MSO+U are highly related. This is due to the fact that, since AMSO and MSO+U have significantly different syntax, the restriction to finite set quantifiers has dramatically different consequences. In particular languages definable in WAMSO inhabit all finite levels of the Borel hierarchy, while WMSO+U is confined in the third level.

In [1], the satisfiability problem for $\mathrm{AMSO}^{\mathrm{np}}$/WAMSO was reduced to a certain form of tiling problem. The main contribution of this paper is to solve a special case of this tiling problem. In consequence we can solve the satisfiability problem for a fragment of $\mathrm{AMSO}^{\mathrm{np}}$, which we denote $\mathrm{AMSO}^{\mathrm{np}}_{2s}$. In this fragment we only allow formulae of the form $\exists t \forall s \exists r\, \varphi(r,s,t)$, where φ does not use quantifiers over number variables, and the variables r and s can be only used simultaneously, in subformulae of the form $s < f(x) \leq r$. For the proof, we develop a new generalization of the Simon's theorem about factorization forests [16].

2 Preliminaries

Asymptotic monadic second-order logic (AMSO for short) extends MSO by the ability to describe asymptotic properties of quantities. It refers to *weighted structures* $\langle \mathfrak{A}, \overline{f} \rangle$ consisting of a relational structure $\mathfrak{A}$ and a tuple of functions $f_i \colon dom(\mathfrak{A}) \to \mathbb{N}$ (the *weight functions*). We only consider the case when $\mathfrak{A}$ is an infinite word (ω-word). Syntactically AMSO extends MSO by the following constructions:

- quantifiers over *number variables* that range over natural numbers, and
- atomic formulae $f(x) \leq r$, where f is a weight function, x a first-order variable, and r a number variable; such formulae are restricted to appear positively inside the existential quantifier binding r (and dually: negatively inside a universal quantifier).

We will usually reserve the letters $x, y, z, \ldots$ for first-order variables and the letters $r, s, t, \ldots$ for number variables.

The main theorem of this paper is about a fragment of AMSO, denoted $\mathrm{AMSO}^{\mathrm{np}}_{2s}$, where the formulae are of the form $\exists t \forall s \exists r\, \varphi(r,s,t)$ where φ does not use number quantifiers, and the variables r and s can be only used simultaneously, in subformulae of the form $s < f(x) \leq r$ (formally: $(f(x) \leq r) \wedge \neg(f(x) \leq s)$).

▶ **Example 2.1.** The following are formulae of $\mathrm{AMSO}^{\mathrm{np}}_{2s}$:

- $\exists t \forall x\, (f(x) \leq t)$ says that the weights are bounded;
- $\forall s \exists r \forall x \exists y\, (y > x \wedge s < f(y) \leq r)$ says that infinitely many weights occur infinitely often in the weighted infinite word;
- the disjunction of the above two (we can move the quantifiers to the front).

▶ Remark. It is easy to see that a formula of the form

$$\exists t_1 \dots \exists t_k \forall s_1 \dots \forall s_l \exists r_1 \dots r_m\, \varphi(r_1, \dots, r_m, s_1, \dots, s_l, t_1, \dots, t_k)$$

is equivalent to $\exists t \forall s \exists r\, \varphi(r, \dots, r, s, \dots, s, t, \dots, t)$.[1] For this reason we allow in $\mathrm{AMSO}^{\mathrm{np}}_{2s}$ only formulae with single quantifiers $\exists t \forall s \exists r$, having in mind that decidability immediately extends to formulae with blocks of such quantifiers.

The following is the main result of this paper.

▶ **Theorem 2.2.** *Given a formula $\psi \in \mathrm{AMSO}^{\mathrm{np}}_{2s}$, it is decidable whether there exists a weighted infinite word in which ψ is satisfied.*

The Commutative Lossy Tiling Problem

Theorem 9 of [1] reduces satisfiability of $\mathrm{AMSO}^{\mathrm{np}}$ to a certain (multidimensional) *lossy tiling problem*. In this paper we solve a commutative variant of this problem, in dimension one.

A *picture* $p\colon \{1, \dots, h\} \times \{1, \dots, w\} \to \Sigma$ is a rectangle labelled by letters from a finite alphabet Σ, where h and w are *height* and *width* of the picture. For $i \in \{1, \dots, w\}$, the i-th *column* of the picture is the word $p(1,i)p(2,i)\dots p(h,i)$; similarly we define the j-th row for $j \in \{1, \dots, h\}$. A language $K \subseteq \Sigma^*$ is *commutative* (*lossy*) if it is closed under reordering (respectively: removing) of letters. In the *commutative lossy tiling problem* we are given regular languages $K, L \subseteq \Sigma^*$ (the *column language* and the *row language*), where the column language K is commutative and lossy. A *solution* of the tiling system (K, L) is a picture p such that all columns in p belong to K and all rows in p belong to L. We ask whether, for all $h \in \mathbb{N}$, there exists a solution of height h. Notice that since K is commutative and lossy, we can reorder rows in a solution and again obtain a solution; we can also remove some rows and obtain a solution of smaller height. Consequently demanding solutions of each height $h \in \mathbb{N}$ amounts to demanding solutions of arbitrarily large height $h \in \mathbb{N}$.

3 From the Logic to Tiling Systems

The reduction from satisfiability of $\mathrm{AMSO}^{\mathrm{np}}$ to the multidimensional lossy tiling problem is given in [1], but we need to observe that the restriction to $\mathrm{AMSO}^{\mathrm{np}}_{2s}$ yields the commutative lossy tiling problem. Let us concentrate on the case where the formulae contain only one weight function; satisfiability of the general case easily reduces to this situation.

Before starting, we eliminate the outermost existential quantifier. Suppose that we have a formula $\psi = \exists t \forall s \exists r\, \varphi(r,s,t) \in \mathrm{AMSO}^{\mathrm{np}}_{2s}$. We create a formula $\psi' = \forall s \exists r\, \varphi'(r,s) \in \mathrm{AMSO}^{\mathrm{np}}_{2s}$

[1] See Proposition 14 in the appendix to [1], available at the authors' webpages.

using an additional unary predicate $\mathsf{small}(x)$: φ' is obtained from φ by replacing each atom $f(x) \leq t$ by $\mathsf{small}(x)$, and by replacing each subformula $s < f(x) \leq r$ by $s < f(x) \leq r \wedge \neg\mathsf{small}(x)$. It is easy to see that ψ is satisfiable if and only if ψ' is satisfiable. The idea is that small marks those positions on which the weight function f "is small".

Next, we apply the reduction of [1] to the formula ψ'. Let us explain briefly that the resulting tiling system is indeed a commutative lossy tiling system. The reduction is realized in three steps.

In the first step, the satisfiability of $\mathrm{AMSO}^{\mathrm{np}}$ is reduced to the *limit satisfiability problem*. The idea is to chop an infinite word into infinitely many finite pieces that have the same theory (making repeated use of the Theorem of Ramsey). Originally, this is a theory with respect to all $\mathrm{AMSO}^{\mathrm{np}}$ formulae up to some quantifier rank. We should replace it by the theory with respect to formulae where r and s are only used simultaneously, in subformulae of the form $s < f(x) \leq r$. Such theories have all compositionality properties needed for the proof which, thus, still goes through after this modification. The resulting formulae in the limit satisfiability problem test only for the theory of the finite words. So again r and s are only used simultaneously, in subformulae of the form $s < f(x) \leq r$.

In the second step, it is argued that a formula of the form $\forall s \exists r \varphi(r, s)$ is equivalent to $\forall s \varphi(s + 1, s)$. This step is not affected by our modification.

In the third step, the limit satisfiability problem is reduced to the lossy tiling problem. First, we observe that, because there is just one universally quantified variable s, the resulting tiling system has dimension one. Then we have to slightly change the resulting tiling system to make it commutative. The alphabet of the system was $\Sigma \times \{<, =, >\}$, and the column language was $K = \bigcup_{a\in\Sigma}(a, <)^*((a, =) \cup \varepsilon)(a, >)^*$. Intuitively, the meaning of a letter $(a, <)$ (or $(a, =)$, $(a, >)$) is that the row number is smaller (respectively: equal, greater) than the value of the weight function on this position (thus in each column initial rows contain $(a, <)$, then there is at most one $(a, =)$ marking the value of the weight function, and then we have $(a, >)$). Now in our formulae we cannot distinguish small values from big values, we can only test whether $s < f(x) \leq s + 1$ holds. For this reason $(a, <)$ and $(a, >)$ become indistinguishable and can be replaced by one letter, call it $(a, \neq)$. The row language now becomes $K = \bigcup_{a\in\Sigma}(a, \neq)^*((a, =) \cup \varepsilon)(a, \neq)^*$, which is commutative.

4 Monoids

In this section we slightly rephrase the problem of deciding the commutative lossy tiling problem using algebraic methods. Recall that every regular language (in particular the row language L) can be recognized by a morphism into a finite monoid. This means that there exists a morphism $\varphi\colon \Sigma^* \to M$ into a finite monoid M, and a set $F \subseteq M$ such that $L = \varphi^{-1}(F)$. It will be more convenient to label the picture directly with elements of M instead of Σ (using $\varphi(a)$ instead of a). The row language then becomes $\pi^{-1}(F)$, where the *evaluation* map $\pi\colon M^* \to M$ is the morphism defined by $\pi(s_1 \dots s_k) = s_1 \cdot \dots \cdot s_k$. The column language changes into $K' = \{\varphi(a_1) \dots \varphi(a_h) \mid a_1 \dots a_h \in K\}$, which again is commutative and lossy.

Next, we observe that we can restrict our considerations to sets F that are singletons. Namely, the tiling system $(K', \pi^{-1}(F))$ has arbitrarily high solutions if and only if for some $s \in F$ the system $(K', \pi^{-1}(s))$ has arbitrarily high solutions. Indeed, every solution of the latter system is a solution of the former. On the other hand, from a solution of $(K', \pi^{-1}(F))$ of height h we can choose rows evaluating to the most popular element $s_h \in F$ and obtain a solution of $(K', \pi^{-1}(s_h))$ of height at least $\frac{h}{|F|}$. Although elements s_h depend on h, some of them has to be used for infinitely many h (that is, for arbitrarily large h).

As a final simplification, let us analyze the column language. For a language L, let $L^{\downarrow}$ be the closure of L under removing letters (we add to L all words obtained by removing letters in words from L), and $L^{\circlearrowleft}$ the closure of L under reordering letters (we add to L all words obtained by reordering letters in words from L). A language (over M) is called a *base language* if it is of the form $(wA^*)^{\downarrow\circlearrowleft}$, where $A \subseteq M$ and $w \in (M \setminus A)^*$ (words in $(wA^*)^{\downarrow\circlearrowleft}$ can use letters from A arbitrarily many times, and letters from w at most as many times as they appear in w). Base languages play an important role in our proof. We use the letter ρ to denote base languages. Notice that the content of a base language $(wA^*)^{\downarrow\circlearrowleft}$ determines A uniquely, and w up to the order of its letters (with the assumption that w does not contain letters from A). The set A is called the *global part* of $\rho = (wA^*)^{\downarrow\circlearrowleft}$. We denoted it by $gl(\rho)$. The *norm* $\|\rho\|$ of a base language ρ is the length $|w|$.

It is a consequence of Higman's Lemma that every lossy language (over M) can be written as a finite union of languages of the form $(A_0^* b_1 A_1^* \dots b_k A_k^*)^{\downarrow}$, where $A_0, \dots, A_k \subseteq M$ and $b_1, \dots, b_k \in M$. Our column language K is lossy and commutative, so it is a finite union of base languages. Summing up, we can restate our problem as follows:

Input: a finite monoid M, a finite set B of base languages over M, an element $s \in M$;
Question: does there exist for every $h \in \mathbb{N}$ a picture of height h each column of which belongs to $\bigcup B$ and every row of which to $\pi^{-1}(s)$?

For a picture p we define the *evaluation* of p, denoted $\pi(p)$, as the word of the same length as the height of p, whose i-th letter equals the evaluation of the i-th row of p. Then, instead of requesting that every row of p belongs to $\pi^{-1}(s)$, we can say that $\pi(p) \in s^*$.

5 The Decision Procedure

Our decision procedure maintains a set of base languages such that for every word from some of these languages there is a picture evaluating to this word where each column belongs to $\bigcup B$. New base languages are added following two kinds of schemas, the *product schema* and *diagonal schema*. These schemas are just ways of describing pictures of arbitrarily large size, evaluating to all words in some base language. The main difficulty is to prove completeness, i.e., showing that using some other fancy pictures one cannot obtain more base languages than we obtain using pictures generated from our schemas.

Let us now define the two kinds of schemas we use to generate new base languages. Let ρ_1 and ρ_2 be base languages. A *product schema* for ρ_1, ρ_2 is given by a picture q whose rows are divided into *special rows* and *global rows* such that (for $j \in \{1, 2\}$)

1. q has width 2 and the j-th column belongs to ρ_j, and
2. the height of q is at most $\|\rho_1\| + \|\rho_2\| + |M|^2$, and
3. the j-th letter of each global row belongs to $gl(\rho_j)$.

The base language *generated* by q is $(wA^*)^{\downarrow\circlearrowleft}$, where w consists of the letters of $\pi(q)$ corresponding to the special rows and A contains the letters of $\pi(q)$ corresponding to the global rows. We only allow schemas q for which w does not contain letters from A.

While defining a diagonal schema we need to use the power-set monoid. The set $\mathcal{P}(M)$ of subsets of M has a natural monoid structure: $C \cdot D = \{c \cdot d \mid c \in C, d \in D\}$. We say that a set of base languages B is *uniform* when it is nonempty, for all $\rho_1, \rho_2 \in B$ we have $gl(\rho_1) = gl(\rho_2)$, and this set is idempotent. For a uniform B we write $gl(B)$ for the set $gl(\rho)$ with $\rho \in B$. The set of all finite uniform sets of base languages over M is denoted by $UBL(M)$.

x	a	c	z	x
a	b	a	c	c
b	c	y	b	a

y	z	x	x	z	y	z	x	x	a	c	z	x
z	x	z	x	y	a	c	z	z	x	z	y	x
x	a	c	x	x	z	z	y	x	z	x	z	x
a	b	a	c	a	b	a	c	a	b	a	c	c
b	c	z	y	z	y	z	x	y	x	x	b	a

Figure 1 On the left we have an example diagonal schema. Elements of $gl(B)$ are shaded in gray. The first row is a global row, and the other two are special rows (we suppose that $a \cdot b \cdot a \cdot c$ is idempotent). The double line divides the schema horizontally into two pictures. On the right there is a picture created out of the schema for $n = 3$. Here double lines are introduced only for readability. Gray cells are stretched into longer areas evaluating to the same value (e.g. $x = z \cdot x \cdot z \cdot x \cdot y$).

Let B be a uniform set of base languages. A *diagonal schema* for B is given by a picture q whose rows are divided into *special rows* and *global rows* and which is divided horizontally into pictures $q_1, \ldots, q_k$ (which means that $q_1, \ldots, q_k$ have as many rows as q, and the i-th row of q is the concatenation of the i-th rows of $q_1, \ldots, q_k$) such that:

1. each column of q belongs to $\bigcup B$, and
2. each special row of each q_j either has length 1, or evaluates to an idempotent, or it contains a letter belonging to $gl(B)$, and
3. the first and the last letter of each global row of each q_j belongs to $gl(B)$.

The base language *generated* by q is $(wA^*)^{\downarrow\circlearrowleft}$ where w consists of the letters of $\pi(q)$ corresponding to the special rows and A contains the letters of $\pi(q)$ corresponding to the global rows. Again, we only allow schemas q for which w does not contain letters from A. An example diagonal schema is depicted in Figure 1 on the left.

The following theorem states soundness and completeness of our schemas.

▶ **Theorem 5.1.** *Let B_0 be a finite set of base languages over a monoid M. For a function $\eta\colon UBL(M) \to \mathbb{N}$ let $B_0^{\leq\eta} = B_0$ and for each $i > 0$, inductively, let $B_i^{\leq\eta}$ be the set of all base languages ρ such that*

- *$\rho \in B_{i-1}^{\leq\eta}$, or*
- *ρ is generated by some product schema for some base languages $\rho_1, \rho_2 \in B_{i-1}^{\leq\eta}$, or*
- *ρ is generated by some diagonal schema for a uniform set of base languages $B \subseteq B_{i-1}^{\leq\eta}$, of width and height at most $\eta(B)$.*

Then there is a computable function $\eta\colon UBL(M) \to \mathbb{N}$ such that for every $s \in M$ the following two statements are equivalent.

- *For each $h \in \mathbb{N}$, there exists a picture p of height h such that $\pi(p) \in s^*$ and each column belongs to $\bigcup B_0$.*
- *For $x = 3 \cdot (2^{|M|} + 1)^2$, there exists a base language $\rho \in B_x^{\leq\eta}$ with $s \in gl(\rho)$.*

Notice that this theorem implies the decidability of the commutative lossy tiling problem. Indeed, given $B_{i-1}^{\leq\eta}$ we can calculate $B_i^{\leq\eta}$ because the number of product and diagonal schemas to consider is finite (the size of product schemas is bounded by definition, and the size of diagonal schemas is bounded by the function η).

6 Soundness

In this section we prove the easier direction of Theorem 5.1: the implication from the second to the first statement. The proof is based on the following two lemmas.

▶ **Lemma 6.1.** *Let ρ be a base language generated by some product schema for ρ_1, ρ_2 and let $u \in \rho$. Then there exists a picture p each column of which belongs to $\rho_1 \cup \rho_2$ and such that $\pi(p) = u$.*

▶ **Lemma 6.2.** *Let ρ be a base language generated by some diagonal schema for a uniform set of base languages B and let $u \in \rho$. Then there exists a picture p each column of which belongs to $\bigcup B$ and such that $\pi(p) = u$.*

Using these the lemmas we can prove the soundness implication of Theorem 5.1 as follows. Let $B_i^{\leq\eta}$ be the sets from Theorem 5.1. The function η bounding the sizes of diagonal schemas does not matter for this implication. We will prove by induction on i that if $u \in \bigcup B_i^{\leq\eta}$, then there exists a picture p each column of which belongs to $\bigcup B_0$ and such that $\pi(p) = u$. Then the statement of the lemma follows by taking $u = s^h$ since we have $s \in gl(\rho)$ for some $\rho \in B_x^{\leq\eta}$, which implies that $u \in \bigcup B_x^{\leq\eta}$.

For $i = 0$ the claim is trivial: we can take a picture p containing u as the only column. For $i > 0$, let $u \in \bigcup B_i^{\leq\eta}$. Then $u \in \rho$ for some $\rho \in B_i^{\leq\eta}$. If $\rho \in B_{i-1}^{\leq\eta}$ the claim follows by inductive hypothesis. Otherwise, we are in the second or the third case of definition of $B_i^{\leq\eta}$. Hence, we can apply Lemma 6.1 or 6.2 to obtain a picture p' each column of which belongs to $\bigcup B_{i-1}^{\leq\eta}$ and such that $\pi(p') = u$. Moreover, by inductive hypothesis there exists, for each column u_j of p', a picture p_j each column of which belongs to $\bigcup B_0$ and such that $\pi(p_j) = u_j$. To obtain p we replace in p' the j-th column u_j by p_j, for every j. Then $\pi(p) = \pi(p')$ and p has the desired properties.

In the remaining part of this section we prove Lemmas 6.1 and 6.2.

Proof of Lemma 6.1. The proof follows immediately from the definitions. Let q be a product schema for ρ_1, ρ_2 which generates ρ. Since the global rows of q contain only letters from the global parts of ρ_1, ρ_2, we can duplicate in q any global row without destroying the property that the j-th column belongs to ρ_j. We can also remove any row and reorder the rows. By performing such operations we can obtain a picture p such that $\pi(p) = u$. ◀

Proof of Lemma 6.2. Let $\rho = (wA^*)^{\downarrow\circlearrowleft}$, let q be a diagonal schema for B generating ρ, and let $q_1, \ldots, q_k$ be the pictures into which q is divided. W.l.o.g. we assume that each global row of q evaluates to a different element of A (otherwise we remove redundant rows). Note that, if the lemma holds for some word u, then it holds also for any u' obtained from u by removing and reordering letters (because we can remove and reorder the rows of the resulting picture). Thus it is enough to consider, for each $n \in \mathbb{N}$, a column u which begins by w and then has each letter of A repeated n times.

The idea of constructing a picture p out of the diagonal schema q is depicted in Figure 1. For each $j \in \{1, \ldots, k\}$ we create a picture p_j by modifying q_j as follows. p_j will have $|A| \cdot (n-1)$ more rows than q_j; more precisely, each global row of q_j will produce n rows of p_j, while each special row of q_j will produce only one row of p_j. Fix some j and let m be the width of q_j. If $m = 1$, we just replace each global row by n copies. Assume now that $m > 1$. Then the width of p_j will be nm.

We start by considering a special row v. If $\pi(v)$ is idempotent, we can just repeat the content of the row n times without changing the value of the product. Otherwise, by definition there exists an index i such that the i-th letter of v belongs to $gl(B)$. As the first $i-1$ letters of the new row we take the first $i-1$ letters of v. As the last $m-i$ letters of the new row we take the last $m-i$ letters of v. On the remaining $mn - m + 1$ positions we place letters from $gl(B)$ in such a way that their product is equal to the i-th letter of v (this

is possible since $gl(B)$ is idempotent by uniformity of B). Again, the value of the product remains unchanged.

Finally, we consider a global row v of q_j. We will produce n rows in p_j; the i-th of them, for $i \in \{1, \ldots, n\}$, is created in the following way. On the first $(i-1)m+1$ positions of the new row we place letters from $gl(B)$ in such a way that their product is equal to the first letter of v (recall that by definition the first and the last letter of v are in $gl(B)$). On the last $(n-i)m+1$ positions of the new row we place letters from $gl(B)$ in such a way that their product is equal to the last letter of v. On the remaining $m-2$ positions we put the middle $m-2$ letters of v, without the first and the last letter.

For the picture p we take the concatenation of $p_1, \ldots, p_k$ (which means that the i-th row of p is obtained by concatenating the i-th rows of $p_1, \ldots, p_k$). We observe that the evaluation of p is u (the rows created out of special rows evaluate to w, and the rows created out of global rows evaluate to elements of A, each n times). It remains to observe that each column of p (so of each p_j) belongs to $\bigcup B$. When p_j has only one column, this is clear, because it is obtained by duplicating some letters from $gl(B)$ in a column from $\bigcup B$. Otherwise (with m as above), the column with number $i + i'm$ of p_j (for $i \in \{1, \ldots, m\}$) is obtained from the column number i of q_j (which is in $\bigcup B$): the letters which are not in $gl(B)$ are taken at most once, on the other positions we take some letters from $gl(B)$. Thus the new column is also in $\bigcup B$. ◀

7 Completeness

In this section we prove the remaining direction of Theorem 5.1: the implication from the first to the second statement. The strategy is as follows. First we consider special cases that can be described by a single schema. In Section 7.1 we analyze pictures of width 2. For these one can extract a product schema. In Section 7.2 we analyze pictures whose columns come from a union of a uniform set of base languages. These can be turned into a diagonal schema. As a technical tool we introduce in Section 7.3 a new version of the Factorization Trees Theorem [16]. This theorem is used in Section 7.4 to decompose arbitrary picture into simple fragments corresponding to single schemas, which allows us to conclude the proof. During the whole section we consider the monoid M as fixed.

7.1 Products

We start by analyzing width 2 pictures in order to turn them into product schemas.

▶ **Lemma 7.1.** *Let ρ_1, ρ_2 be two base languages and let p be a picture of width 2 such that the first column belongs to ρ_1 and the second one to ρ_2. Then there exists a product schema for ρ_1, ρ_2 which generates a base language ρ such that $\pi(p) \in \rho$ and $gl(\rho) = gl(\rho_1) \cdot gl(\rho_2)$.*

Proof. We take $\rho = (wA^*)^{\downarrow\circlearrowright}$ where $A = gl(\rho_1) \cdot gl(\rho_2)$ and w consists of those letters of $\pi(p)$ which are not in A (taken as many times as they appear in $\pi(p)$). Obviously $\pi(p) \in \rho$. In q we include all rows of p that do not evaluate to an element of A. These will be the special rows. Note that in each of these rows either the first letter does not belong to $gl(\rho_1)$, or the second letter does not belong to $gl(\rho_2)$. Thus we have at most $\|\rho_1\| + \|\rho_2\|$ of such rows. Moreover, for each $r \in gl(\rho_1)$ and each $s \in gl(\rho_2)$, we add to q a row with r in the first column and s in the second one. These will be the global rows. We have $|gl(\rho_1)| \cdot |gl(\rho_2)| \leq |M|^2$ of them. We see that q is a product schema for ρ_1, ρ_2 that generates ρ. ◀

7.2 Uniform Case

Next, we consider a special case when the set of base languages allowed in columns is uniform, and we show that such a picture can be transformed into a single diagonal schema.

▶ **Lemma 7.2.** *There is a computable function $\eta\colon UBL(M) \to \mathbb{N}$ such that, for every finite uniform set of base languages B and every picture p each column of which belongs to $\bigcup B$, there exists a diagonal schema for B of width and height at most $\eta(B)$ that generates a base language ρ such that*

- *$\pi(p) \in \rho$ and*
- *$E = gl(B)$ and $A = gl(\rho)$ satisfy $E \subseteq A = E \cdot A \cdot E$.*

Let us comment on the second condition ($E \subseteq A = E \cdot A \cdot E$). It enforces that the base language ρ (and hence also the diagonal schema) is more robust. This will be useful later. Namely, the global part of ρ contains not only the letters that appear many times in $\pi(p)$, but also all letters from $gl(B)$ (since $E \subseteq A$) and all results of surrounding the former letters by letters from $gl(B)$ (since $E \cdot A \cdot E \subseteq A$). Note that we always have $A \subseteq E \cdot A \cdot E$, as each global row begins and ends by a letter from $gl(B)$.

The proof of the lemma is based the following fact saying that each word can be chopped into a small number of idempotents and single letters. To simplify notation, we write $\exp(x)$ for 2^x.

▶ **Fact 7.3.** *Let M' be a finite monoid and w a word over M'. Then we can divide w into fragments $w = w_1 \dots w_k$ for $k \leq \exp(3|M'|)$ such that, for every i, either $|w_i| = 1$, or $\pi(w_i)$ is idempotent.*

This fact is applied to a picture, in order to split it horizontally as in a diagonal schema. While reading the next lemma have in mind that E will be used for $gl(B)$.

▶ **Lemma 7.4.** *Let p be a picture and $E \subseteq M$. Let x be the number of rows of p which contain only letters from $M \setminus E$ and let y be the smallest number such that in each column of p there are at most y positions containing a letter from $M \setminus E$. Then, for some $k \leq \exp(3(y-x+1)|M|^y)$, we can divide p horizontally into pictures $p_1, \dots, p_k$ in such a way that each row of each p_j either has length 1, or evaluates to an idempotent, or contains a letter from E.*

Proof. We prove the claim by induction on $y - x$ (note that $x \leq y$). Consider the monoid $M' = M^x$ with coordinatewise multiplication. Let I be the set of (numbers of) those rows which contain only letters from E (by definition $|I| = x$). Let $w \in (M')^*$ be the word consisting of the rows of p which are in I (each letter contains the elements of M appearing in the x rows of a column). Applying Fact 7.3 to w, we obtain a factorisation $w = w_1 \dots w_m$ for $m \leq \exp(3|M|^x) \leq \exp(3|M|^y)$ where each w_j either has length 1, or evaluates to an idempotent. We divide p into $p'_1, \dots, p'_m$ in the same way: the width of p'_j is the same as the length of w_j. Then every row of each p'_j which is in I either has length 1, or evaluates to an idempotent. For each p'_j we proceed in one of two ways.

- If each row of p'_j which is not in I contains a letter from E, this p'_j satisfies the statement of the lemma.
- Otherwise, there exists a row of p'_j not in I which contains only letters from $M \setminus E$. Then $x' \geq x + 1$ and $y' \leq y$, where x' is the number of rows of p'_j which contain only letters from $M \setminus E$ and y' is the smallest number such that in each column of p'_j there are at most y' positions containing a letter from $M \setminus E$. We use the inductive hypothesis for p'_j to obtain a subdivision of p'_j as required by the statement of the lemma.

Since each of the subdivisions returns at most $\exp(3(y'-x'+1)|M|^{y'}) \leq \exp(3(y-x)|M|^y)$ pictures, in total we have at most $m \cdot \exp(3(y-x)|M|^y) \leq \exp(3(y-x+1)|M|^y)$ pictures. ◀

Proof of Lemma 7.2. Set $E = gl(B)$. First, we divide p into pictures $p_1, \ldots, p_k$ by applying Lemma 7.4 to the picture p and to the set E. Note that the number y in the statement of the lemma is equal to the maximal norm of a base language in B, and that $x \geq 0$. We have $k \leq \exp(3(y-x+1)|M|^y) \leq \exp(3(y+1)|M|^y)$. Let I_1 be the set of all those numbers i of rows of p such that the first or the last letter of the i-th row of some p_j is in $M \setminus E$. Note that $|I_1| \leq 2ky$ (where y is again the maximal norm of a base language in B): we look for letters from $M \setminus E$ only in $2k$ columns (the first and the last column of each p_j), and in each of these columns we have at most y letters from $M \setminus E$. The picture p with this division is almost a diagonal schema as needed (when the rows from I_1 are treated as the special rows). However we still need to reduce its size and ensure that $E \subseteq A = E \cdot A \cdot E$.

For each i, we denote by s_i the evaluation of the i-th row without the first and the last letter (so the value of the i-th row can be obtained by multiplying its first letter by s_i and by its last letter). Let I_2 be the set of numbers $i \notin I_1$ of rows of p such that there are less than $|E|^2$ numbers $j \notin I_1$ for which $s_i = s_j$. Notice that $|I_2| \leq |M|^3$ (we have at most $|E|^2 - 1 \leq |M|^2$ rows for each of $|M|$ possible values of s_i). Set $I = I_1 \cup I_2$.

Next, let A' be the set of s_i for all $i \notin I$. Let $A = (E \cdot A' \cdot E) \cup E$ and let w contain those letters of $\pi(p)$ which are not in A (as many times as they appear in $\pi(p)$); we take $\rho = (wA^*)^{\downarrow\circlearrowleft}$. As E is idempotent, it follows that $\pi(p) \in \rho$ and $E \subseteq A = E \cdot A \cdot E$. It remains to construct a diagonal schema q for B that generates ρ.

The width of q will be the same as of p. We also divide q into $q_1, \ldots, q_k$ of the same widths as $p_1, \ldots, p_k$. We include in q all those rows of p which do not evaluate to an element of A. These will be the special rows. Note that by the statement of Lemma 7.4, any row of p can be taken as a special row: inside each p_j it either has length 1, or evaluates to an idempotent, or it contains a letter belonging to E. Moreover, all these rows are in I; indeed, any other row $i \notin I$ evaluates to $r \cdot s_i \cdot r'$, where $s_i \in A'$ and r, r' are the first and the last letter of the row, which are in E by definition of I_1. Consequently, there are at most $|I|$ such rows.

Then, for each $s \in A'$ we consider $|E|^2$ rows $i \notin I$ for which $s_i = s$ (we have at least $|E|^2$ such rows by definition of I_2) and we modify them as follows. For each pair $r, r' \in E$ we add to q one such row in which we replace the first letter by r and the last letter by r'. These will be global rows. This works as the first and the last letter of each such row inside each p_j belong to E and the replaced letters are also in E. Additionally, for each $s \in E$, we add to q a row containing only letters from E, which evaluates to s (as E is idempotent, we can find such rows of every desired length). These will also be global rows. This works since all letters of these rows are in E.

We see that every column of q belongs to $\bigcup B$: it is a column of p with some letters removed and some letters from E added. The special rows evaluate exactly to the letters of w. The global rows of the first kind evaluate to all elements of $E \cdot A' \cdot E$, and the global rows of the second kind to all elements of E. Thus q generates the base language ρ.

It remains to bound the size. The number of rows in q is at most

$$|I| + |E| \cdot |A'| \cdot |E| + |E| \leq 2ky + 2|M|^3 + |M| \leq 2y \cdot \exp(3(y+1)|M|^y) + 3|M|^3,$$

where y is the maximal norm of a base language in B. We denote the last number by $\theta(B)$ (it depends only on B and $|M|$).

We also have to restrict the width of q. Since we have started from an arbitrary picture p, the width can be arbitrary; so we have to remove some columns. Fix some q_j that has more

than one column. In each special row whose value is not idempotent there is some letter from E. In each such row we choose one of these letters and we mark the column containing it (we don't want to remove this column). We also mark the first and the last column of q_j; they contain letters from E in global rows, so we also don't want to remove them. We have marked at most $\theta(B)+2$ columns. We want to remove some not-marked columns, so that the resulting picture evaluates to the same word. For each number of columns i, consider the picture consisting of the first i columns of q_j; let w_i be the evaluation of this picture (w_i is a word in M^h, where $h \leq \theta(B)$ is the height of q_j). Whenever $w_i = w_l$ for some $i < l$, we can remove the columns number $i+1, \dots, l$, and the whole new picture will still evaluate to $\pi(q_j)$; we do this only when none of these columns is marked. We repeat this removal procedure as long as such pair of indices i, l exists. By the Pigeon Hole Principle, among any $|M|^h+1$ numbers we can find two i, l for which $w_i = w_l$. Thus, after the removal, we have at most $(\theta(B)+1)\cdot(|M|^h+1)+1$ columns in q_j. Because we do not remove marked columns, the properties of a diagonal schema are preserved. In total we have at most $k\cdot((\theta(B)+1)\cdot(|M|^h+1)+1) \leq \exp(3(y+1)|M|^y)\cdot((\theta(B)+1)\cdot(|M|^h+1)+1)$ columns. We denote the last number by $\eta(B)$. Note that $\theta(B) \leq \eta(B)$. Thus, not only the width but also the height of q is bounded by $\eta(B)$. ◀

7.3 Factorization Trees

In this subsection we present a new generalization of the Factorization Trees Theorem [16]. In this generalization the result in an "idempotent" node depends on some additional data in the arguments. This theorem will be used in Section 7.4 to decompose an arbitrary picture into pictures of the special form considered in Sections 7.1 and 7.2.

The nodes of our factorization trees will be labelled by elements of some set D, possibly infinite. We also have a finite monoid M' and a projection $\sigma\colon D \to M'$. The construction is parameterized by two functions. The function $pr\colon D^2 \to D$ describes a product. The other function

$$st\colon \{d_1 \dots d_c \in D^+ \mid \sigma(d_1) = \dots = \sigma(d_c) \text{ is idempotent}\} \to D$$

describes an operation which will be used in idempotent nodes. We require that these functions satisfy the following axioms:

$$(*)\quad \sigma(pr(a,b)) = \sigma(a)\cdot\sigma(b)\,, \qquad \text{for all } a, b \in D,$$

$$(**)\quad \sigma(st(d_1\dots d_c)) = \sigma(d_1) \quad \text{or} \quad \sigma(st(d_1\dots d_c)) <_{\mathcal{J}} \sigma(d_1)\,, \qquad \text{for all } d_1\dots d_c \in dom(st).$$

The preorder $\leq_{\mathcal{J}}$ in the second axiom is defined by $r \leq_{\mathcal{J}} s$ if there are u_1, u_2 such that $r = u_1 \cdot s \cdot u_2$ (recall that each monoid contains an identity element, that is allowed as u_1 and u_2). Two elements are $\mathcal{J}$-equivalent, denoted $r \sim_{\mathcal{J}} s$, when $r \leq_{\mathcal{J}} s$ and $s \leq_{\mathcal{J}} r$. Equivalence classes of this relation are called $\mathcal{J}$-classes. We write $r <_{\mathcal{J}} s$ when $r \leq_{\mathcal{J}} s$, but $r \not\sim_{\mathcal{J}} s$. A *factorization tree* is a tree labelled by elements of D whose nodes are of one of three forms:

- a *leaf*,
- a *binary node* with exactly two children; it is labelled by $pr(d_1, d_2)$, where d_1, d_2 are the labels of its children,
- an *idempotent node* with at least three children labelled by $d_1, \dots, d_c$ such that $\sigma(d_1) = \dots = \sigma(d_c)$ is idempotent; the node itself is labelled by $st(d_1 \dots d_c)$.

The word (in D^+) read from the leaves of a factorization tree t (from left to right) is called the *input* of t, and the label of the root of t is called its *output*.

Note that standard factorization trees as in [16] can be obtained by taking $D = M'$ and $st(e \dots e) = e$. In computation trees for a stabilization monoid [11], we again have $D = M'$, but $st(e \dots e)$ now depends on the number of arguments: it is e for short sequences $e \dots e$, and $e^\sharp$ for longer $e \dots e$. The key result is the existence of factorization trees of constant height as described in the following theorem.

▶ **Theorem 7.5.** *For every $v \in D^+$, there exists a factorization tree with input v and height at most*[2] $3(|M'| + 1)^2$.

This theorem can be proved basically in the same way as its version for stabilization monoids ([11], Theorem 3.3): the tree is constructed in a bottom-up way, so it is not a problem that the result in an idempotent node depends in some way on the subtree constructed below.

7.4 The Final Argument

In this subsection we conclude our proof of the missing implication of Theorem 5.1. The function η is taken from Lemma 7.2. Let $B_i^{\leq\eta}$ be sets of base languages as in Theorem 5.1, for some finite set of base languages B_0. Each $B_i^{\leq\eta}$ is finite. Let h be the smallest number greater than the norm of each base language in $B_x^{\leq\eta}$, where $x = 3 \cdot (2^{|M|} + 1)^2$. Take some picture p of height h each column of which belongs to $\bigcup B_0$ and for which $\pi(p) \in s^*$. Our goal is to find $\rho \in B_x^{\leq\eta}$ such that $s \in gl(\rho)$.

We use the theorem about factorization trees from the previous subsection. For D we take the set of pairs (w, ρ), where $w \in M^h$ and ρ is a base language containing w. We set $M' = \mathcal{P}(M)$ and $\sigma((w, \rho)) = gl(\rho)$. It remains to define the functions pr and st.

To define pr, consider two letters (w_1, ρ_1) and (w_2, ρ_2) from D. Let p be the picture with two columns w_1 and w_2. By Lemma 7.1, there exists a base language ρ such that $\pi(p) \in \rho$, $gl(\rho) = gl(\rho_1) \cdot gl(\rho_2)$, and there exists a product schema for ρ_1, ρ_2 generating ρ. We define $pr((w_1, \rho_1), (w_2, \rho_2)) = (\pi(p), \rho)$. Then Axiom $(*)$ is satisfied because $gl(\rho) = gl(\rho_1) \cdot gl(\rho_2)$. Observe also that when $\rho_1, \rho_2 \in B_j^{\leq\eta}$, for some j, then $\rho \in B_{j+1}^{\leq\eta}$.

To define st, consider elements $(w_1, \rho_1) \dots (w_k, \rho_k) \in D^+$ such that $gl(\rho_1) = \dots = gl(\rho_k)$ is idempotent. Let p be the picture with k columns $w_1, \dots, w_k$, set $B = \{\rho_1, \dots, \rho_k\}$, and let $E = gl(B)$. Then B is a uniform set of base languages and each column of p belongs to $\bigcup B$. By Lemma 7.2, there exists a base language ρ such that $\pi(p) \in \rho$, $E \subseteq gl(\rho) = E \cdot gl(\rho) \cdot E$, and there exists a diagonal schema for B of width and height at most $\eta(B)$ generating ρ. We define $st((w_1, \rho_1) \dots (w_k, \rho_k)) = (\pi(p), \rho)$. Observe that when $\rho_i \in B_j^{\leq\eta}$, for some j and all i, then $\rho \in B_{j+1}^{\leq\eta}$. Axiom $(**)$ is satisfied due to the following fact.

▶ **Fact 7.6.** *Let $E, A \subseteq M$ where E is idempotent and $E \subseteq A = E \cdot A \cdot E$. Then either $A = E$ or $A <_{\mathcal{J}} E$.*

To conclude the proof, recall that p is a picture of height h each column of which belongs to $\bigcup B_0$ and such that $\pi(p) \in s^*$. We want to find a base language $\rho \in B_x^{\leq\eta}$ with $s \in gl(\rho)$. Consider a word $w = (d_1, \rho_1) \dots (d_m, \rho_m) \in D^+$, where d_i is the i-th column of p and $\rho_i \in B_0$ is some base language with $d_i \in \rho_i$. By Theorem 7.5 there exists a factorization tree t with height at most x and input w. Let (d, ρ) be its output. Note that $d = \pi(p) = s^h$ (by definition of pr and st), and $d \in \rho$ (by definition of D). Moreover, $\rho \in B_x^{\leq\eta}$ (more generally, when a root of a subtree of height at most i is labelled by some (d', ρ'), then $\rho' \in B_i^{\leq\eta}$). As h is greater than the size of ρ, we have $s \in gl(\rho)$, which is what we wanted to prove.

[2] One can obtain a bound of $3|M'|$, but this requires a more complicated proof.

References

1 Achim Blumensath, Olivier Carton, and Thomas Colcombet. Asymptotic monadic second-order logic. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 – 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 87–98. Springer, 2014. doi:10.1007/978-3-662-44522-8_8.

2 Mikołaj Bojańczyk. The finite graph problem for two-way alternating automata. *Theor. Comput. Sci.*, 3(298):511–528, 2003. doi:10.1016/S0304-3975(02)00866-6.

3 Mikołaj Bojańczyk. Weak MSO with the unbounding quantifier. *Theory Comput. Syst.*, 48(3):554–576, 2011. doi:10.1007/s00224-010-9279-2.

4 Mikołaj Bojańczyk and Thomas Colcombet. Bounds in w-regularity. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 285–296. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.17.

5 Mikołaj Bojańczyk, Paweł Parys, and Szymon Toruńczyk. The MSO+U Theory of (N,<) Is Undecidable. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd International Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 21:1–21:8. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.21.

6 Mikołaj Bojańczyk and Szymon Toruńczyk. Weak MSO+U over infinite trees. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th – March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 648–660. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.648.

7 Tomás Brázdil, Krishnendu Chatterjee, Antonín Kucera, and Petr Novotný. Efficient controller synthesis for consumption games with multiple resource types. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification – 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, pages 23–38. Springer, 2012. doi:10.1007/978-3-642-31424-7_8.

8 J. Richard Büchi. On a decision method in a restricted second order arithmetic. In *Logic, Methodology and Philosophy of Science (Proc. 1960 Internat. Congr.)*, pages 1–11, Stanford, Calif., 1962. Stanford Univ. Press.

9 Claudia Carapelle, Alexander Kartzow, and Markus Lohrey. Satisfiability of CTL* with constraints. In Pedro R. D'Argenio and Hernán C. Melgratti, editors, *CONCUR 2013 – Concurrency Theory – 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings*, volume 8052 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 2013. doi:10.1007/978-3-642-40184-8_32.

10 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th Internatilonal Collogquium, ICALP 2009, Rhodes, greece, July 5-12, 2009, Proceedings, Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 139–150. Springer, 2009. doi:10.1007/978-3-642-02930-1_12.

11 Thomas Colcombet. Regular cost functions, part I: logic and algebra over words. *Logical Methods in Computer Science*, 9(3), 2013. doi:10.2168/LMCS-9(3:3)2013.

12 Nathanaël Fijalkow and Martin Zimmermann. Cost-parity and cost-streett games. *Logical Methods in Computer Science*, 10(2), 2014. doi:10.2168/LMCS-10(2:14)2014.

13 Szczepan Hummel and Michal Skrzypczak. The topological complexity of MSO+U and related automata models. *Fundam. Inform.*, 119(1):87–111, 2012.

14 Daniel Kirsten. Distance desert automata and the star height problem. *ITA*, 39(3):455–509, 2005. doi:10.1051/ita:2005027.

15 Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. From liveness to promptness. *Formal Methods in System Design*, 34(2):83–103, 2009. doi:10.1007/s10703-009-0067-z.

16 Imre Simon. Factorization forests of finite height. *Theor. Comput. Sci.*, 72(1):65–94, 1990. doi:10.1016/0304-3975(90)90047-L.

17 Szymon Toruńczyk. *Languages of profinite words and the limitedness problem.* PhD thesis, Warsaw University, 2011.

The Complexity of Phylogeny Constraint Satisfaction

Manuel Bodirsky[1], Peter Jonsson[2], and Trung Van Pham[3]

1 **Institut für Algebra, TU Dresden, Germany**
`manuel.bodirsky@tu-dresden.de`
2 **Department of Computer and System Science, Linköpings Universitet, Sweden**
`peter.jonsson@liu.se`
3 **Institut für Algebra, TU Dresden, Germany**
`trung.van_pham@tu-dresden.de`

Abstract

We systematically study the computational complexity of a broad class of computational problems in phylogenetic reconstruction. The class contains for example the rooted triple consistency problem, forbidden subtree problems, the quartet consistency problem, and many other problems studied in the bioinformatics literature. The studied problems can be described as *constraint satisfaction problems* where the constraints have a first-order definition over the rooted triple relation. We show that every such phylogeny problem can be solved in polynomial time or is NP-complete. On the algorithmic side, we generalize a well-known polynomial-time algorithm of Aho, Sagiv, Szymanski, and Ullman for the rooted triple consistency problem. Our algorithm repeatedly solves linear equation systems to construct a solution in polynomial time. We then show that every phylogeny problem that cannot be solved by our algorithm is NP-complete. Our classification establishes a dichotomy for a large class of infinite structures that we believe is of independent interest in universal algebra, model theory, and topology. The proof of our main result combines results and techniques from various research areas: a recent classification of the model-complete cores of the reducts of the homogeneous binary branching C-relation, Leeb's Ramsey theorem for rooted trees, and universal algebra.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases constraint satisfaction problems, computational complexity, phylogenetic reconstruction, Ramsey theory, model theory

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.20

1 Introduction

Phylogenetic consistency problems are computational problems that have been studied for phylogenetic reconstruction in computational biology, but also in other areas dealing with large amounts of possibly inconsistent data about trees, such as computational genealogy or computational linguistics. Given a collection of *partial information* about a tree, we would like to know whether the information is *consistent* in the sense that there exists a single tree that it is compatible with all the given partial information. A concrete example of a computational problem in this context is the *rooted triple consistency problem*. In an instance of this problem, we are given a set V of variables, and a set of triples from V^3, written in the form $ab|c$ where $a, b, c \in V$, and we would like to know whether there exists a rooted tree T whose leaves are from V such that for each of the given triples $ab|c$ the youngest common ancestor of a and b in this tree is below the youngest common ancestor of a and c. Aho, Sagiv, Szymanski, and Ullmann presented a polynomial-time algorithm for this problem [1].


© Manuel Bodirsky, Peter Jonsson, and Trung V. Pham;
licensed under Creative Commons License CC-BY
33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 20; pp. 20:1–20:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Many computational problems that are defined similarly as the rooted triple consistency problem have been studied in the literature. Examples include the *subtree avoidance problem* (Ng, Steel, and Wormald [24]) and the *forbidden triple problem* (Bryant [16]) which are NP-hard problems. Bodirsky & Mueller [8] have determined the complexity of rooted phylogeny problems for the special case where the where the relations are disjunctive combinations of the rooted triple relation. This result covers, for instance, the subtree avoidance problem and the forbidden triple problem.

We present a considerable strengthening of this result, and classify the complexity of phylogeny problems for all sets of phylogeny constraints that can be first-order defined with the mentioned rooted triple relation and equality (of leaves). The reader should be aware that many problems of this type may appear exotic from a biological point of view — the name "phylogeny" should not be taken too literally. Our results show that each of the problem problems obtained in this way is either polynomial-time solvable or NP-complete. As we will demonstrate later (see Section 2), this class of problems is expressive enough to contain also *unrooted phylogeny problems*. A famous example of such an unrooted phylogeny problem is the NP-complete *quartet consistency problem* (Steel [25]): here we are given a set V of variables, and a set of quartets ab:cd with $a, b, c, d \in V$, and we would like to know whether there exists a tree T with leaves from V such that for each of the given quartets ab:cd the shortest path from a to b does not intersect the shortest path from c to d in T. Another phylogeny problem that has been studied in the literature and that falls into the framework of this paper (but not into the one in [8]) is the *tree discovery problem* [1]: here, the input consists of a set of 4-tuples of variables, and the task is to find a rooted tree T such that for each quadruple (x, y, u, v) in the input the youngest common ancestor of x and y is a proper descendant of the youngest common ancestor of u and v.

The proof of the complexity classification is based on a variety of methods and results. Our first step is that we give an alternative description of phylogeny problems as constraint satisfaction problems (CSPs) over a countably infinite domain where the constraint relations are first-order definable over the (up to isomorphism unique) *homogeneous binary branching C-relation*, a well-known structure in model theory. We let C denote this particular relation. A central result that simplifies our work considerably is a recent analysis of the endomorphism monoids of such relations [5]. Informally, this result implies that there are precisely four types of phylogeny problems: (1) trivial (i.e., if there is a solution, there is a constant solution), (2) rooted, (3) unrooted, and (4) degenerate cases that have been called *equality CSPs* [6]. Rooted and unrooted phylogeny problems will be introduced formally in Theorem 14. We will show that all unrooted phylogeny problems are NP-hard, and the complexity of all equality CSPs is already known.

The basic method to proceed from there is the *algebraic approach* to constraint satisfaction problems. Here, one studies certain sets of operations (known as *polymorphisms*) instead of analyzing the constraints themselves. An important tool to work with polymorphisms over infinite domains is Ramsey theory. In this paper, we need a Ramsey result for rooted trees due to Leeb [23], for proving that polymorphisms behave canonically on large parts of the domain (in the sense of Bodirsky & Pinsker [10]), and this allows us to perform a simplified combinatorial analysis.

Interestingly, all phylogeny problems that can be solved in polynomial time fall into one class and can be solved by the same algorithm. This algorithm is a considerable extension of the algorithm by Bodirsky & Mueller [8] for the rooted triple consistency problem, and the algorithm by Bodirsky & Mueller is in turn a considerable extension of the algorithm by Aho, Sagiv, Szymanski, and Ullmann [1]. The algorithm by Aho et al. is based on analysing

connected components in particular graphs while our algorithm is based on repeatedly solving systems of linear Boolean equations. An illustrative example of a phylogeny problem that can be solved in polynomial time by our algorithm, but not the algorithms from [1, 8], is the following computational problem: the input is a 4-uniform hypergraph with vertex set V; the question is whether there exists a rooted tree T with leaf set V such that every hyperedge in the input T has two disjoint subtrees that each contain precisely two of the vertices of the hyperedge.

All phylogeny problems that cannot be solved by our algorithm are NP-complete. Our results are stronger than this complexity dichotomy, though, and we prove that every phylogeny problem satisfies a universal-algebraic dichotomy statement that holds for a large class of infinite structures (Theorem 24), which is of independent interest in the study of homogeneous structures and their polymorphism clones. In this respect, the situation is similar as in previous classifications for CSPs where the constraints are first-order definable over the order of the rationals $(\mathbb{Q}; <)$ from [7] or the random graph [11]. In comparison to these previous works, the dichotomy we present here is easier to state (there is just one tractable class), but harder to prove with the existing methods: in particular, unlike the situation for constraints that are first-order definable over the random graph [11], the polymorphisms that characterise the tractable cases cannot be chosen to be canonical (in the sense of Bodirsky & Pinsker [10]) on the entire domain. As such, our dichotomy provides an important test-case for potentially much wider classifications of CSPs over homogeneous structures.

The paper has the following structure. We give basic definitions concerning phylogeny problems in Section 2 and explain how these problems can be viewed as constraint satisfaction problems for reducts of the homogeneous binary branching C-relation. Section 3 provides a brief but self-contained introduction to the universal-algebraic approach. In Section 4 we translate structural properties of phylogenetic relations into definability properties in terms of syntactically restricted formulas, which we call *affine Horn formulas*. Here we also state our tractability result. In Section 5, we characterize the tractable class of phylogeny problems with polymorphisms. Finally, in Section 6 we put everything together and state and prove our main results including the previously mentioned complexity dichotomy. Section 5 can safely be skipped by readers that are only interested in the complexity dichotomy and not in the stronger algebraic dichotomy. A report version of this paper with full proofs can be found in the appendix.

2 Phylogeny problems

All structures in this paper are assumed to be countable. In this section, we first define (in Sections 2.1 and 2.2) a class of phylogeny problems and illustrate it by showing instances from this class that have been studied in the literature. We continue in Section 2.3 by showing how to formulate such phylogeny problems as *constraint satisfaction problems* over an infinite domain.

2.1 Rooted trees

We fix some standard terminology concerning rooted trees. Let T be a tree (i.e., an undirected, acyclic, and connected graph) with a distinguished vertex r, the *root* of T. The vertices of T are denoted by $V(T)$. All trees in this paper will be *binary*, i.e., all vertices except for the root have either degree 3 or 1, and the root has either degree 2 or 0. The *leaves* $L(T)$ of T are the vertices of T of degree one.

For $u, v \in V(T)$, we say that *u lies below v* if the path from u to r passes through v. We say that *u lies strictly below v* if u lies below v and $u \neq v$. The *youngest common ancestor (yca)* of a set of vertices $S \subseteq V(T)$ is the node u that lies above all vertices in S and has maximal distance from r; this node is uniquely determined by S.

▶ **Definition 1.** The *leaf structure* of a binary rooted tree T is the relational structure $(L(T); C)$ where $C(a, b, c)$ holds in C if and only if $\mathrm{yca}(\{b, c\})$ lies strictly below $\mathrm{yca}(\{a, b, c\})$ in T. We also call T the *underlying tree* of the leaf structure.

It is well-known that a rooted tree is uniquely determined by its leaf structure.

▶ **Definition 2.** For finite $S_1, S_2 \subseteq L(T)$, we write $S_1|S_2$ if neither of $\mathrm{yca}(S_1)$ and $\mathrm{yca}(S_2)$ lies below the other. For sequences of (not necessarily distinct) vertices $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$ with $n, m \geq 1$ we write $x_1, \ldots, x_n|y_1, \ldots, y_m$ if $\big(\bigcup_{1\leq i\leq n}\{x_i\}\big)\big|\big(\bigcup_{1\leq i\leq m}\{y_i\}\big)$.

In particular, $x|yz$ (which is the notation that is typically used in the literature on phylogeny problems) is equivalent to $C(x, y, z)$.

Note that if $x|yz$ then this includes the possibility that $y = z$; however, $x|yz$ implies that $x \neq y$ and $x \neq z$. Hence, for every triple x, y, z of leaves in a rooted binary tree, exactly one of $x|yz$, $y|xz$, $z|xy$, $x = y = z$ holds. Also note that $x_1, \ldots, x_n|y_1, \ldots, y_m$ if and only if $x_i x_j|y_k$ and $x_i|y_k y_l$ for all $i, j \leq n$ and $k, l \leq m$.

2.2 Phylogeny problems

An *atomic phylogeny formula* is a formula of the form $x|yz$ or of the form $x = y$.

A *phylogeny formula* is a quantifier-free formula ϕ that is built from atomic phylogeny formulas with the usual Boolean connectives (disjunction, conjunction, negation).

We say that a phylogeny formula ϕ with variables V is *satisfiable* if there exists a rooted binary tree T and a mapping $s\colon V \to L(T)$ such that ϕ is satisfied by T under s (with the usual semantics of first-order logic). In this case we also say that (T, s) is a *solution* to ϕ.

Let $\Phi = \{\phi_1, \phi_2, \dots\}$ be a finite set of phylogeny formulas. Then the *phylogeny problem* for Φ is the following computational problem.

Phylo(Φ)
INSTANCE: A finite set V of variables, and a finite set Ψ of phylogeny formulas obtained from phylogeny formulas $\phi \in \Phi$ by substituting the variables from ϕ by variables from V.
QUESTION: Is there a tree T and a mapping $s\colon V \to L(T)$ such that (T, s) satisfies all formulas from Ψ?

We use $x_1, \ldots, x_n|y_1, \ldots, y_m$ as a shortcut for $\bigwedge_{i,j\in\{1,\ldots,n\},k,l\in\{1,\ldots,m\}}(y_k|x_i x_j \wedge x_i|y_k y_l)$ and we use $\text{all-diff}(x_1, \ldots, x_k)$ as a shortcut for $\bigwedge_{1\leq i<j\leq k} x_i \neq x_j$.

▶ **Example 3.** The following NP-complete problem was introduced and studied in a closely related form by Ng, Steel, and Wormald [24]. We are given a set of rooted trees on a common leave set V, and we would like to know whether there exists a tree T with leave set V such that, intuitively, for each of the given trees T' the tree T does *not* match with the tree T'. The hardness proof for this problem given Ng, Steel, and Wormald [24] shows that already the phylogeny problem $\mathrm{Phylo}\big(\{\neg x|yz \wedge \text{all-diff}(x, y, z), \neg(u|xy \wedge v|yu) \wedge \text{all-diff}(x, y, u, v)\}\big)$, which can be seen as a special case of the problem above, is NP-hard. ◀

▶ **Example 4.** The quartet consistency problem described in the introduction can be cast as $\mathrm{Phylo}(\{\phi\})$ where ϕ is the phylogeny formula $(xy|u \wedge xy|v) \vee (x|uv \wedge y|uv)$. Indeed, this

formula describes all rooted trees with leaves x, y, u, v where the shortest path from x to y does not intersect the shortest path from u to v (whether or not this is true is in fact independent from the position of the root). ◀

Our main result is a full classification of the computational complexity of Phylo(Φ).

▶ **Theorem 5.** *Let Φ be a finite set of phylogeny formulas. Then Phylo(Φ) is in P or NP-complete.*

2.3 Phylogeny problems as CSPs

As mentioned in the introduction, every phylogeny problem can be formulated as a constraint satisfaction problem over an infinite domain. This reformulation will be essential for using universal-algebraic and Ramsey-theoretic tools.

Let Γ be a structure with relational signature $\tau = \{R_1, R_2, \dots\}$. This is, Γ is a tuple $(D; R_1^\Gamma, R_2^\Gamma, \dots)$ where D is the (finite or infinite) *domain* of Γ and where $R_i^\Gamma \subseteq D^{k_i}$ is a relation of arity k_i over D. When Δ and Γ are two τ-structures, then a *homomorphism* from Δ to Γ is a mapping h from the domain of Δ to the domain of Γ such that for all $R \in \tau$ and for all $(x_1, \dots, x_k) \in R^\Delta$ we have $(h(x_1), \dots, h(x_k)) \in R^\Gamma$.

Suppose that the signature τ of Γ is finite. Then the *constraint satisfaction problem for* Γ, denoted by CSP(Γ), is the following computational problem.

CSP(Γ)
INSTANCE: A finite τ-structure Δ.
QUESTION: Is there a homomorphism from Δ to Γ?

We say that Γ is the *template* or the *constraint language* of the problem CSP(Γ). To formulate phylogeny problems as CSPs, let $\Phi = \{\phi_1, \dots, \phi_n\}$ be a finite set of phylogeny formulas. If $x_1, \dots, x_{k_i}$ are the variables of ϕ_i, then we introduce a new relation symbol R_i of arity k_i, and we write τ for the set of all these relation symbols.

For an instance Ψ of Phyl(Φ) with variables V, we associate to Ψ a τ-structure Δ_Ψ with domain V as follows. For $R \in \tau$ of arity k, the relation R^Δ contains the tuple $(y_1, \dots, y_k) \in V^k$ if and only if the instance Ψ contains a formula ψ that has been obtained from a formula $\phi \in \Phi$ by replacing the variables $x_1, \dots, x_k$ of ϕ by the variables $y_1, \dots, y_k \in V$.

▶ **Proposition 6.** *Let Φ be a finite set of phylogeny formulas. Then there exists a τ-structure Γ_Φ with countable domain $\mathbb{L}$ and the following property: an instance Ψ of* Phyl(Φ) *is satisfiable if and only if Δ_Ψ homomorphically maps to Γ_Φ.*

The structure Γ_Φ in Proposition 6 is by no means unique, and such structures are easy to construct. The specific choice for Γ_Φ presented below is important later in the proof of our complexity classification; as we will see, it has many pleasant model-theoretic properties. To define Γ_Φ, we first define a 'base structure' $(Ł; C)$, and then define Γ_Φ in terms of $(Ł; C)$. The structure $(Ł; C)$ is a well-studied object in model theory and the theory of infinite permutation groups, and will be defined via *Fraïssé-amalgamation*.

Homomorphisms from Γ to Γ are called *endomorphisms* of Γ. An *automorphism* of Γ is a bijective endomorphism whose inverse is also an endomorphism. The set containing all endomorphisms of Γ is denoted End(Γ) while the set of all automorphisms is denoted Aut(Γ). A relational structure Γ is called *homogeneous* if every isomorphism between finite induced substructures of Γ can be extended to an automorphism of Γ. Homogeneous structures Γ with finite relational signature are *ω-categorical*, i.e., all countable structures that satisfy the same first-order sentences as Γ are isomorphic (see e.g. Cameron [18] or Hodges [21]).

When working with relational structures, it is often convenient to not distinguish between a relation and its relation symbol. For instance, when we write $(L(T), C)$ for a leaf structure (Definition 1), the letter C stands both for the relation symbol, and for the relation itself. This should never cause confusion.

▶ **Proposition 7** (Proposition 7 in Bodirsky, Jonsson, & Van Pham [5]). *There exists an (up to isomorphism unique) homogeneous structure $(\mathbb{L}; C)$ with the property that all its finite substructures are isomorphic to leaf structures of finite rooted binary trees.*

The structure $(\mathbb{L}; C)$ is well-studied in the literature, and the relation C is commonly referred to as the *binary branching homogeneous C-relation*.

▶ **Definition 8.** Let Δ be a structure. Then a relational structure Γ with the same domain as Δ is called a *reduct* of Δ if all relations of Γ have a first-order definition in Δ (using conjunction, disjunction, negation, universal and existential quantification, as usual).

It is well-known that all structures with a first-order definition in an ω-categorical structures are again ω-categorical (we refer once again to Hodges [21], Theorem 7.3.8; the analogous statement for homogeneity is false.) Furthermore, an ω-categorical structure is homogeneous if and only if it has *quantifier-elimination*, that is, every first-order formula is over Γ equivalent to a quantifier-free formula; see Hodges [21].

Proof of Proposition 6. Let Φ be a finite set of phylogeny formulas. Let Γ_Φ be the reduct of $(\mathbb{L}; C)$ defined as follows. For every $\phi \in \Phi$ with free variables $x_1, \dots, x_k$, we have the k-ary relation R_ϕ in Γ_ϕ which is defined by the formula ϕ over $(\mathbb{L}; C)$. It follows straightforwardly from the definitions that this structure has the properties required in the statement of Proposition 6.

Conversely, every CSP for a reduct $\Gamma = (\mathbb{L}; R_1, \dots, R_n)$ of $(\mathbb{L}; C)$ corresponds to a phylogeny problem. Let ϕ_i be a quantifier-free first-order definition of R_i in $(\mathbb{L}; C)$. When Δ is an instance of CSP(Γ), consider the instance Ψ of Phyl($\{\phi_1, \dots, \phi_n\}$) where the variables V are the vertices of Δ, and where Ψ contains for every tuple $(v_1, \dots, v_n) \in R_i^\Delta$ the formula $\phi_i(v_1, \dots, v_n)$. It is again straightforward to verify that Δ homomorphically maps to Γ if and only if Ψ is a satisfiable instance of Phyl($\{\phi_1, \dots, \phi_n\}$). Therefore, the class of phylogeny problems corresponds precisely to the class of CSPs whose template is a reduct of $(\mathbb{L}; C)$. ◀

3 The universal-algebraic approach

We apply the so-called *universal-algebraic approach* to obtain our results. For a more detailed introduction to this approach, see Bodirsky [3]. We discuss some important concepts and present certain results in the following three subsections.

3.1 Primitive positive definability and interpretability

A first-order formula ϕ with free variables $z_1, \dots, z_k$ over the signature τ is *primitive positive* if it is of the form $\exists x_1, \dots, x_n\, (\psi_1 \wedge \dots \wedge \psi_m)$, where $\psi_1, \dots, \psi_m$ are *atomic*, that is, of the form $R(y_1, \dots, y_k)$ or of the form $y_1 = y_2$, for $R \in \tau$ and $y_1, \dots, y_k \in \{x_1, \dots, x_n, z_1, \dots, z_k\}$. When Γ is a τ-structure, then ϕ defines over Γ a k-ary relation, namely the set of all k-tuples that satisfy ϕ in Γ. We let $\langle\Gamma\rangle$ denote the set of all finitary relations that are primitive positive definable in Γ. The following result motivates why we are interested in positive primitive definability in connection with the complexity of CSPs.

▶ **Lemma 9** (Jeavons [22]). *Let* Γ *be a constraint language, and let* Γ' *be the structure obtained from* Γ *by adding the relation* R*. If* R *is primitive positive definable in* Γ*, then* $\mathrm{CSP}(\Gamma)$ *and* $\mathrm{CSP}(\Gamma')$ *are polynomial-time equivalent.*

This result was originally proved for finite-domains CSPs but the proof extends immediately to infinite-domain CSPs.

Primitive positive interpretations are a generalisation of primitive positive definitions, and are often used for proving NP-hardness results; we refer the reader to Bodirsky [3] for more information about this. We will consider the relation $\mathrm{NAE} = \{0,1\}^3 \setminus \{(0,0,0),(1,1,1)\}$ in connection with primitive positive interpretations. Clearly $\mathrm{CSP}(\{0,1\}; \mathrm{NAE})$ is NP-complete.

▶ **Definition 10.** A relational σ-structure Δ has a *(first-order) interpretation* I in a τ-structure Γ if there exists a natural number d, called the *dimension* of I, and

- a τ-formula $\delta_I(x_1, \ldots, x_d)$ – called the *domain formula*,
- for each atomic σ-formula $\phi(y_1, \ldots, y_k)$ a τ-formula $\phi_I(\overline{x}_1, \ldots, \overline{x}_k)$ where the $\overline{x}_i$ denote disjoint d-tuples of distinct variables – called the *defining formulas*,
- a surjective map h from all d-tuples of elements of Γ that satisfy δ_I to Δ – called the *coordinate map*,

such that for all atomic σ-formulas ϕ and all tuples in the domain of h, $\Delta \models \phi(h(\overline{a}_1), \ldots, h(\overline{a}_k))$ if and only if $\Gamma \models \phi_I(\overline{a}_1, \ldots, \overline{a}_k)$.

If the formulas δ_I and ϕ_I are all primitive positive, we say that the interpretation I is *primitive positive*. We say that Δ is *pp interpretable with parameters in* Γ is Δ has an interpretation I where the formulas δ_I and ϕ_I might involve elements from Γ (the *parameters*). That is, interpretations with parameters in Γ interpretations in the expansion of Γ by finitely many constants. The importance of primitive positive interpretations follows from the following lemma.

▶ **Lemma 11.** *Let* Γ *and* Δ *be structures with finite relational signature. Suppose that* Γ *is* ω*-categorical and that* Δ *has a primitive positive interpretation in* Γ*. Then there is a polynomial-time reduction from* $\mathrm{CSP}(\Delta)$ *to* $\mathrm{CSP}(\Gamma)$*. If* Γ *is a model-complete core, then the interpretation might even be with parameters and the conclusion of the lemma still holds.*

3.2 Polymorphisms

Primitive positive definability can be characterised by preservation under so-called *polymorphisms* – this is the starting point of the universal-algebraic approach to constraint satisfaction (see, for instance, Bulatov, Jeavons, and Krokhin [17] for this approach over finite domains). The *(direct–, categorical–, or cross–) product* $\Gamma_1 \times \Gamma_2$ of two relational τ-structures Γ_1 and Γ_2 is a τ-structure on the domain $D_{\Gamma_1} \times D_{\Gamma_2}$. For all relations $R \in \tau$ the relation $R\big((x_1, y_1), \ldots, (x_k, y_k)\big)$ holds in $\Gamma_1 \times \Gamma_2$ iff $R(x_1, \ldots, x_k)$ holds in Γ_1 and $R(y_1, \ldots, y_k)$ holds in Γ_2. Homomorphisms from $\Gamma^k = \Gamma \times \cdots \times \Gamma$ to Γ are called *polymorphisms* of Γ. When R is a relation over the domain D, then we say that f *preserves* R (or that R is *closed under* f) if f is a polymorphism of $(D; R)$. Note that unary polymorphisms of Γ are endomorphisms of Γ. When ϕ is a first-order formula that defines R, and f preserves R, then we also say that f *preserves* ϕ.

The set of all polymorphisms $\mathrm{Pol}(\Gamma)$ of a relational structure forms an algebraic object called *clone* [26], which is a set of operations defined on a set D that is closed under composition and that contains all projections. The set $\mathrm{Pol}(\Gamma)$ is also locally closed, in the following sense. A set of functions $\mathcal{F}$ with domain D is *locally closed* if every function f with the following property belongs to $\mathcal{F}$: for every finite subset A of D there is some operation

$g \in \mathcal{F}$ such that $f(a) = g(a)$ for all $a \in A^k$. We write $\overline{F}$ for the smallest set that is locally closed and contains F.

Polymorphism clones can be used to characterise primitive positive definability over a finite structure; this follows from results by Bodnarčuk, Kalužnin, Kotov, and Romov [15] and Geiger [20]. The characterisation remains true if the structure is ω-categorical.

▶ **Theorem 12** (Bodirsky & Nešetřil [9]). *Let Γ be an ω-categorical structure. Then the primitive positive definable relations in Γ are precisely the relations preserved by the polymorphisms of Γ.*

3.3 Model-complete cores

Let Γ and Δ be structures with relational signature τ. A homomorphism of Γ to Δ is said to be an *embedding* if it is injective and preserves $\neg R$ for all $R \in \tau$. The structure Γ is a *core* if all its endomorphisms are embeddings. Note that endomorphisms preserve existential positive formulas, and embeddings preserve existential formulas. The structure $(\mathbb{L}; C)$ for example is a core.

A first-order theory T is said to be *model-complete* if every embedding between models of T preserves all first-order formulas. A structure is called *model-complete* if its first-order theory is model-complete. The structure $(\mathbb{L}; C)$ is model-complete since it is even homogeneous. We say that two structures Γ and Δ are *homomorphically equivalent* if there exists a homomorphism from Γ to Δ, and one from Δ to Γ. Clearly, homomorphically equivalent structures have identical CSPs.

▶ **Theorem 13** (Bodirsky [2]). *Let Γ be an ω-categorical structure. Then Γ is homomorphically equivalent to an ω-categorical model-complete core Δ. The structure Δ is unique up to isomorphism, and again ω-categorical.*

Hence, we speak in the following of *the* model-complete core of an ω-categorical structure. The model-complete cores of reducts of $(\mathbb{L}; C)$ have been classified recently [5].

▶ **Theorem 14** (Bodirsky, Jonsson, & Pham [5]). *Let Γ be a reduct of $(\mathbb{L}; C)$, and Δ its model-complete core. Then one of the following applies.*

1. *Δ has just one element.*
2. *Δ is isomorphic to a reduct of $(\mathbb{L}; =)$.*
3. *$\Delta = \Gamma$ has the same endomorphisms as $(\mathbb{L}; Q)$ where Q is the relation defined by the formula given in Example 4 (the 'unrooted' situation).*
4. *$\Delta = \Gamma$ has the same endomorphisms as $(\mathbb{L}; C)$ (the 'rooted' situation).*

Define the relations

$$C_d := \{(x, y, z) \in \mathbb{L}^3 \ : \ x | yz \wedge y \neq z\}, \text{ and}$$
$$Q_d := \{(x, y, u, v) \in \mathbb{L}^4 \ : \ Q(x, y, u, v) \wedge x \neq y \wedge u \neq v\}.$$

The following result is a consequence of Theorem 14.

▶ **Lemma 15.** *Let Γ be a reduct of $(\mathbb{L}; C)$ which does not have a constant endomorphism and which is not homomorphically equivalent to an equality constraint language. Then Γ is a model-complete core, and C_d or Q_d are primitive positive definable in Γ.*

4 Affine Horn formulas

Recall that a Boolean relation R is called *affine* if it can be defined by a system of linear equation systems over the 2-element field. It is well-known (see e.g. [19]) that a Boolean relation is affine if and only if it is preserved by the function $(x, y, z) \mapsto x + y + z \pmod 2$.

▶ **Definition 16.** Let $B \subseteq \{0,1\}^n$ be a Boolean relation. Then $\phi_B(z_1, \ldots, z_n)$ stands for the formula

$$z_1 = \cdots = z_n \vee \bigvee_{t \in B \setminus \{(0,0,\ldots,0),(1,1,\ldots,1)\}} \{z_i : t_i = 0\} | \{z_i : t_i = 1\} \, .$$

The formula ϕ_B is called *affine* if $B \cup \{(0,0,\ldots,0),(1,1,\ldots,1)\}$ is affine.

▶ **Definition 17.** An *affine Horn clause* is a formula of the form $x_1 \neq y_1 \vee \cdots \vee x_n \neq y_n$ or of the form $x_1 \neq y_1 \vee \cdots \vee x_n \neq y_n \vee \phi(z_1, \ldots, z_k)$ where ϕ is an affine formula. An *affine Horn formula* is a conjunction of affine Horn clauses. A relation $R \subseteq \mathbb{L}^k$ is called *affine Horn* if it can be defined by an affine Horn formula over $(\mathbb{L}; C)$. A phylogeny constraint language is called *affine Horn* if all its relations are affine Horn.

▶ **Example 18.** The relation $\{(z_1, z_2, z_3, z_4) \in \mathbb{L}^4 : z_1 z_2 | z_3 z_4 \text{ and } z_1 = z_2 \Leftrightarrow z_3 = z_4\}$ is affine Horn. To see this, first note that it can equivalently be defined by the formula

$$(z_1 z_2 | z_3 z_4 \vee z_1 = z_2 = z_3 = z_4) \wedge (z_1 \neq z_2 \vee z_3 = z_4) \wedge (z_3 \neq z_4 \vee z_1 = z_2) \wedge z_1 \neq z_3.$$

It is sufficient to verify that each conjunct is an affine Horn clause. We do this here for the first conjunct. Consider the relation $R = \{(0,0,0,0),(1,1,0,0),(0,0,1,1),(1,1,1,1)\}$, which is affine since $(z_1, z_2, z_3, z_4) \in R$ if and only if $z_1 + z_2 = 0 \pmod 2$ and $z_3 + z_4 = 0 \pmod 2$. We see that $\phi_R(z_1, z_2, z_3, z_4)$ is equivalent to $z_1 = z_2 = z_3 = z_4 \vee z_1 z_2 | z_3 z_4$.

The relation $N := \{(x, y, z) \in \mathbb{L}^3 : (xy|z \vee x|yz)\}$ has been called the *forbidden triple relation* by Bryant [16] and it plays an important role in the classification. Bryant showed that $\mathrm{CSP}(\mathbb{L}; N)$ is NP-complete. We are therefore particularly interested in those reducts Γ of $(\mathbb{L}; C)$ where $N \notin \langle\Gamma\rangle$. We will prove later that when Γ is a reduct of $(\mathbb{L}; C)$ with finite relational signature such that $C \in \langle\Gamma\rangle$ and $N \notin \langle\Gamma\rangle$, then $\mathrm{CSP}(\Gamma)$ is in P. The following result is the combinatorial heart of this paper.

▶ **Theorem 19.** *Let Γ be a reduct of $(\mathbb{L}; C)$ such that $C \in \langle\Gamma\rangle$ and $N \notin \langle\Gamma\rangle$. Then all relations in $\langle\Gamma\rangle$ are affine Horn.*

In the proof of Theorem 19 we use the algebraic approach in combination with a Ramsey theorem for trees, due to Leeb[23]; also see Bodirsky [4]. The outline is as follows: if the relation N is not primitive positive definable in Γ, there must be a polymorphism of Γ that does not preserve it, by Theorem 12. We apply Ramsey theory to prove that polymorphisms of Γ must behave *canonically* on large parts of the domain; the technique we use here is developed in a larger context by Bodirsky, Pinsker, and Tsankov [14]. The obtained polymorphisms in turn imply strong structural properties on the relations they preserve which can then be used to prove that all relations that are primitive positive definable in Γ are affine Horn.

▶ **Theorem 20.** *There is a polynomial-time algorithm that decides whether a given affine Horn formula is satisfiable over $(\mathbb{L}; C)$.*

We give a sketch of how the algorithm works. The key is a procedure which does the following: either it returns a solution where all variables take different values in $\mathbb{L}$ or it returns a set of variables that must take equal value in all solutions. If variables that are syntactically forced to be different are contracted, the algorithm rejects, and otherwise we find a solution after a linear number of variable contractions. The idea for the key procedure is as follows: we solve a particular affine Boolean equation system in order to determine which variables will be mapped below the same child of the root in a solution to the instance. This can be done in polynomial time by Gaussian elimination. If there is no solution, the procedure returns all variables, and if there is a solution, it recursively proceeds with two sub-instances induced by the solution to the equation system.

▶ **Corollary 21.** *Let Γ be a reduct of $(\mathbb{L}; C)$ which is affine Horn and has a finite signature. Then* CSP(Γ) *can be solved in polynomial time.*

We can now prove Theorem 5.

Proof of Theorem 5. Let Γ be a reduct of $(\mathbb{L}; C)$ with finite relational signature and let Δ be the model-complete core of Γ. The structure Δ is homomorphically equivalent to Γ by Theorem 13 so CSP(Γ) and CSP(Δ) have the same complexity. We need to consider four cases by Lemma 15.

(1) Δ has just one element and CSP(Δ) is trivially in P.
(2) Δ is isomorphic to a reduct of $(\mathbb{L}; =)$ and CSP(Δ) is either in P or NP-hard by Bodirsky & Kára [6].
(3) $C_d \in \langle\Delta\rangle$. It is easy to show that if $C_d \in \langle\Delta\rangle$, then $C \in \langle\Delta\rangle$, too. In this case, the complexity of CSP(Δ) depends on whether $N \in \langle\Delta\rangle$ or not. If $N \in \langle\Delta\rangle$ then CSP(Δ) is NP-hard (Bryant [16]) as discussed in Section 4. Otherwise, Theorem 19 implies that all relations in Γ are affine Horn and CSP(Γ) is in P by Corollary 21.
(4) $Q_d \in \langle\Delta\rangle$ and CSP(Δ) is NP-hard due to Steel [25]. ◀

5 Affine tree operations

The border between NP-hardness and tractability for phylogeny problems can be stated in terms of polymorphisms. To characterize such polymorphisms, we introduce a certain kind of binary operations over $\mathbb{L}$ which we call *affine tree operations*. The syntactic characterization of affine Horn constraint languages (from Section 4) is convenient to work with when, for instance, constructing algorithms. However, it is not very convenient when studying polymorphisms. In this section, we construct an operation, called tx, such that every relation that is first order definable in $(\mathbb{L}; C)$ is preserved by tx if and only if it can be defined by an affine Horn formula. The operation tx is constructed as follows.

Let U, V be two finite subsets of $\mathbb{L}$. A function $f\colon \mathbb{L}^2 \to \mathbb{L}$ is called *perfectly dominated (by the first argument) on $U \times V$* if the following conditions holds.

- For all $u_1, u_2, u_3 \in U$ and $v_1, v_2, v_3 \in V$ if $u_1|u_2u_3$ then $f(u_1, v_1)|f(u_2, v_2)f(u_3, v_3)$ and
- for all $u \in U$ and $v_1, v_2, v_3 \in V$ if $v_1|v_2v_3$ then $f(u, v_1)|f(u, v_2)f(u, v_3)$.

Let $f\colon \mathbb{L}^2 \to \mathbb{L}$ be an injective function, and U be a finite subset of $\mathbb{L}$. We inductively define whether f is *semidominated on U^2* as follows. If $U = \emptyset$ or $|U| = 1$ then f is semidominated on $U \times U$. Otherwise, f is semidominated on $U \times U$ if there are $U_1, U_2 \subseteq U$ such that $U = U_1 \cup U_2$, $U_1|U_2$, and the following conditions hold.

- f is semidominated on $U_1 \times U_1$ and $U_2 \times U_2$;

- $f(U_1 \times U_1)|f(U_2 \times U_2)$ and $f(U_1 \times U_2)|f(U_2 \times U_1)$;
- $f((U_1 \times U_1) \cup (U_2 \times U_2))|f((U_1 \times U_2) \cup (U_2 \times U_1))$;
- $f(x, y)$ is perfectly dominated on $U_1 \times U_2$ and $f(y, x)$ is perfectly dominated on $U_2 \times U_1$.

We say that an operation $f : \mathbb{L}^2 \to \mathbb{L}$ is an *affine tree operation* if f is semidominated on $U \times U$ for every finite subset U of $\mathbb{L}$. We are now ready for the main result of this section.

▶ **Theorem 22.** *There exists an affine tree operation, which we call* tx*, and endomorphisms* e_1, e_2 *of* $(\mathbb{L}; C)$ *such that* $e_1(\mathrm{tx}(x, y)) = e_2(\mathrm{tx}(y, x))$ *and for every reduct* Γ *of* $(\mathbb{L}; C)$*, the following are equivalent:*

1. Γ *is preserved by* tx*.*
2. *all relations in* $\langle\Gamma\rangle$ *are affine Horn.*
3. *all relations in* Γ *are affine Horn.*

The above theorem can be proved by the idea of the following lemma that comes from the proof of Proposition 6.6 in Bodirsky, Pinsker and Pongracz [13].

▶ **Lemma 23.** *Let* Δ *be* ω*-categorical, and* $f \in \mathrm{Pol}^{(2)}(\Delta)$*. Suppose that for every finite subset* A *of the domain* D *of* Δ *there exists an* $\alpha \in \mathrm{Aut}(\Delta)$ *such that* $f(x, y) = \alpha(f(y, x))$ *for all* $x, y \in A$*. Then there are* $e_1, e_2 \in \mathrm{Aut}(\Delta)$ *such that* $e_1(f(x, y)) = e_2(f(y, x))$ *for all* $x, y \in D$*.*

6 Main result

Our results are much stronger than the complexity classification from Theorem 5, though. We have a dichotomy for reducts of $(\mathbb{L}; C)$ which remains interesting even if P=NP, and which we view as a fundamental result not just in the context of constraint satisfaction. Our dichotomy can be phrased in various different but equivalent ways, using terminology from universal algebra and topology; we mention that there is also an equivalent formulation using primitive positive interpretability. We first introduce the necessary concepts, and then state how they are linked together in the strongest formulation of our results.

Let $\mathcal{C}$ and $\mathcal{D}$ denote two clones as defined in Section 3. A function $\xi : \mathcal{C} \to \mathcal{D}$ is called a *clone homomorphism* if it sends every projection in $\mathcal{C}$ to the corresponding projection in $\mathcal{D}$, and it satisfies the identity $\xi(f(g_1, \ldots, g_n)) = \xi(f)(\xi(g_1), \ldots, \xi(g_n))$ for all n-ary $f \in \mathcal{C}$ and all m-ary $g_1, \ldots, g_n \in \mathcal{C}$. Such a homomorphism ξ is *continuous* if the map ξ is continuous with respect to the topology of pointwise convergence, where the closed sets are precisely the sets that are locally closed as defined in Section 3. We write $\mathbf{1}$ for the clone on the set $\{0, 1\}$ that only contains the projections and carries the discrete topology.

A binary polymorphism of Γ is called *symmetric modulo endomorphisms* if there are endomorphisms e_1 and e_2 of Γ such that $\forall x, y.\, e_1(f(x, y)) = e_2(f(y, x))$.

▶ **Theorem 24.** *Let* Γ *be a reduct of* $(\mathbb{L}; C)$*, and let* Δ *be the model-complete core of* Γ*. Then the following are equivalent.*

1. Δ *has a symmetric polymorphism modulo endomorphisms.*
2. *For all elements* $a_1, \ldots, a_n$ *of* Δ *there is no clone homomorphism from* $\mathrm{Pol}(\Delta, a_1, \ldots, a_n)$ *to* $\mathbf{1}$*.*
3. *For all elements* $a_1, \ldots, a_n$ *of* Δ *there is no continuous clone homomorphism from* $\mathrm{Pol}(\Delta, a_1, \ldots, a_n)$ *to* $\mathbf{1}$*.*
4. *For all elements* $a_1, a_2, \ldots, a_n$ *of* Δ *there is no primitive positive interpretation of* NAE *in* $(\Delta, a_1, a_2, \ldots, a_n)$*.*

If these conditions apply, $\mathrm{CSP}(\Gamma)$ *is in P, otherwise* $\mathrm{CSP}(\Gamma)$ *is NP-complete.*

Proof sketch. The implication (1) ⇒ (2) can be shown to hold in general for ω-categorical model-complete cores Δ and the implication (2) ⇒ (3) is trivial. The implication (3) ⇒ (4) follows from Theorem 28 in [12]. For the implication (4) ⇒ (1), we use the classification of Δ into four types from Theorem 14. For the first type, Δ has just one element and hence satisfies item 1. For the second type, the statement follows from results by Bodirsky and Kára [6]; in fact, tx is a suitable polymorphism. For the third type, $Q_d \in \langle\Gamma\rangle$ by Lemma 15 and one can show that $Q \in \langle\Gamma\rangle$, too. Furthermore, NAE has a primitive positive definition in $(\mathbb{L}; Q, a_1, a_2, a_3)$ for arbitrary pairwise distinct constants $a_1, a_2, a_3 \in \mathbb{L}$ so NAE has a primitive positive definition in (Γ, a_1, a_2, a_3). By Theorem 28 in [12], there is a continuous clone homomorphism from $\mathrm{Pol}(\Gamma, a_1, a_2, a_3)$ to **1**. We can disregard this case since it contradicts our basic assumption.

We now focus on the fourth type. It can be shown that in this case $C \in \langle\Gamma\rangle$ since $C_d \in \langle\Gamma\rangle$ by Lemma 15. If $N \in \langle\Gamma\rangle$, then NAE has a primitive positive definition in (N, a_1, a_2) where $a_1, a_2 \in \mathbb{L}$ are distinct constants. This contradicts (4). If $N \notin \langle\Gamma\rangle$, then Theorem 19 implies that every relation in $\langle\Gamma\rangle$ is affine Horn. By Theorem 22, tx is a binary commutative polymorphism modulo endomorphisms.

If items 1.—4. hold, then Δ has only one element or tx is a binary polymorphism of Γ. If the former holds, then CSP(Γ) is trivially in P. If the latter holds, then it follows from Theorems 20 and 22 that CSP(Γ) is in P. If items 1.—4. do not hold, then there is a clone homomorphisms from $(\Delta, a_1, \ldots, a_n)$ to **1** for some elements $a_1, \ldots, a_n \in \mathbb{L}$ and NAE has a primitive positive interpretation in an expansion of Γ with a finite number of constants. Thus, CSP(Γ) is NP-complete. ◀

The fact that the continuity condition in item 3 of Theorem 24 can simply be dropped in item 2 is remarkable. Indeed, we do not know whether there is an ω-categorical structure whose polymorphism clone homomorphically maps to **1**, but not via a continuous clone homomorphism (see the discussion in [13]).

Suppose that Γ is a reduct of $(\mathbb{L}; C)$ with finite relational signature such that $C \in \langle\Gamma\rangle$. Then one might ask whether the *meta-problem* of deciding the complexity of CSP(Γ) is effective. Here we assume that Γ is given via quantifier-free first-order definitions of its relations in $(\mathbb{L}; C)$. We can then use the techniques developed by Bodirsky, Pinsker, and Tsankov [14] to effectively test whether the relation N is in $\langle\Gamma\rangle$. Thus, the meta-problem for phylogeny problems is decidable.

Acknowledgements. The first and third author have received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 257039.) The second author is partially supported by the Swedish Research Council (VR) under grant 621-2012-3239.

References

1 Alfred V. Aho, Yehoshua Sagiv, Thomas G. Szymanski, and Jeffrey 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.

2 Manuel Bodirsky. Cores of countably categorical structures. *Logical Methods in Computer Science*, 3(1):1–16, 2007.

3 Manuel Bodirsky. Complexity Classification in Infinite-Domain constraint satisfaction. Mémoire d'habilitation à diriger des recherches, Université Diderot – Paris 7. Available at arXiv:1201.0856, 2012.

4 Manuel Bodirsky. Ramsey Classes: Examples and Constructions. To appear in the Proceedings of the 25th British Combinatorial Conference; arXiv:1502.05146, 2015.

5 Manuel Bodirsky, Peter Jonsson, and Trung Van Pham. The reducts of the homogeneous binary branching C-relation. Preprint arXiv:1408.2554, 2014.

6 Manuel Bodirsky and Jan Kára. The Complexity of Equality Constraint Languages. *Theory of Computing Systems*, 3(2):136–158, 2008. A conference version appeared in the proceedings of Computer Science Russia (CSR'06).

7 Manuel Bodirsky and Jan Kára. The Complexity of Temporal Constraint Satisfaction Problems. *Journal of the ACM*, 57(2):1–41, 2009. An extended abstract appeared in the Proc. of the Symp. on Theory of Computing (STOC'08).

8 Manuel Bodirsky and Jens K. Mueller. Rooted Phylogeny Problems. *Logical Methods in Computer Science*, 7(4), 2011. An extended abstract appeared in the proc. of ICDT'10.

9 Manuel Bodirsky and Jaroslav Nešetřil. Constraint Satisfaction with Countable Homogeneous Templates. *Journal of Logic and Computation*, 16(3):359–373, 2006.

10 Manuel Bodirsky and Michael Pinsker. Reducts of Ramsey structures. *AMS Contemporary Mathematics, vol. 558*, pages 489–519, 2011.

11 Manuel Bodirsky and Michael Pinsker. Schaefer's theorem for graphs. In *STOC*, pages 655–664, 2011. Preprint of the long version available at arxiv.org/abs/1011.2894.

12 Manuel Bodirsky and Michael Pinsker. Topological Birkhoff. *Transactions of the American Mathematical Society*, 367:2527–2549, 2015.

13 Manuel Bodirsky, Michael Pinsker, and András Pongrácz. Projective clone homomorphisms. Preprint, available at ArXiv:1409.4601, 2014.

14 Manuel Bodirsky, Michael Pinsker, and Todor Tsankov. Decidability of definability. *Journal of Symbolic Logic*, 78(4):1036–1054, 2013. A conference version appeared in the Proceedings of LICS 2011.

15 V. G. Bodnarčuk, Lev A. Kalužnin, Victor Kotov, and Boris A. Romov. Galois theory for Post algebras, part I and II. *Cybernetics*, 5:243–539, 1969.

16 David Bryant. Building Trees, Hunting for Trees, and Comparing Trees. PhD-thesis at the University of Canterbury, 1997.

17 Andrei A. Bulatov, Andrei A. Krokhin, and Peter G. Jeavons. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34:720–742, 2005.

18 Peter J. Cameron. *Oligomorphic permutation groups*. Cambridge University Press, Cambridge, 1990.

19 Hubie Chen. A rendezvous of Logic, Complexity, and Algebra. *SIGACT News*, 37(4):85–114, 2006.

20 David Geiger. Closed Systems of functions and predicates. *Pacific Journal of Mathematics*, 27:95–100, 1968.

21 Wilfrid Hodges. *Model theory*. Cambridge University Press, Cambridge, 1993.

22 Peter Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200:185–204, 1998.

23 Klaus Leeb. *Vorlesungen über Pascaltheorie*, volume 6 of *Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung*. Friedrich-Alexander-Universität Erlangen-Nürnberg, 1973.

24 Meei Pyng Ng, Mike Steel, and Nicholas C. Wormald. The difficulty of constructing a leaf-labelled tree including or avoiding given subtrees. *Discrete Applied Mathematics*, 98:227–235, 2000.

25 Michael Steel. The complexity of Reconstructing Trees from Qualitative Charaters and Subtrees. *Journal of Classification*, 9:91–116, 1992.

26 Ágnes Szendrei. *Clones in universal algebra*. Séminaire de Mathématiques Supérieures. Les Presses de l'Université de Montréal, 1986.

The MSO+U Theory of $(\mathbb{N}, <)$ Is Undecidable

Mikołaj Bojańczyk[1], Paweł Parys*[2], and Szymon Toruńczyk*[3]

1 **University of Warsaw, ul. Banacha 2, 02-097 Warszawa, Poland**
`bojan@mimuw.edu.pl`
2 **University of Warsaw, ul. Banacha 2, 02-097 Warszawa, Poland**
`parys,szymtor@mimuw.edu.pl`
3 **University of Warsaw, ul. Banacha 2, 02-097 Warszawa, Poland**
`szymtor@mimuw.edu.pl`

Abstract

We consider the logic MSO+U, which is monadic second-order logic extended with the unbounding quantifier. The unbounding quantifier is used to say that a property of finite sets holds for sets of arbitrarily large size. We prove that the logic is undecidable on infinite words, i.e. the MSO+U theory of $(\mathbb{N}, \leq)$ is undecidable. This settles an open problem about the logic, and improves a previous undecidability result, which used infinite trees and additional axioms from set theory.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases automata, logic, unbounding quantifier, bounds, undecidability

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.21

1 Introduction

A celebrated result of Büchi is that the monadic second-order (MSO) theory is decidable for the structure of natural numbers with order

$$(\mathbb{N}, \leq).$$

Stated differently, satisfiability of MSO is decidable over infinite words. This paper shows that the decidability fails after MSO is extended with the unbounding quantifier. The unbounding quantifier, denoted by

$$\mathsf{U}X.\ \varphi(X),$$

binds a set variable X and says that $\varphi(X)$ holds for arbitrarily large finite sets X. As usual with quantifiers, the formula $\varphi(X)$ might have other free variables beside of X. Denote by MSO+U the extension of MSO by this quantifier. The main contribution of the paper is the following theorem.

▶ **Theorem 1.1.** *The* MSO+U *theory of* $(\mathbb{N}, \leq)$ *is undecidable.*

A corollary of the main theorem is undecidability of the logic MSO+inf, which is a logic on profinite words defined in [14], because decidability of MSO+U reduces to decidability of MSO+inf. Another corollary is that the satisfiability on weighted infinite words is undecidable for the logic AMSO introduced in [1], again because of a reduction from decidability of MSO+U.

* Author supported by the National Science Center (decision DEC-2012/07/D/ST6/02443).

© Mikołaj Bojańczyk, Paweł Parys, and Szymon Toruńczyk;
licensed under Creative Commons License CC-BY

Background

The logic MSO+U was introduced in [2], where it was shown that satisfiability is decidable for formulas on infinite trees where the U quantifier is only used once and not under the scope of set quantification. A significantly more powerful fragment of the logic, albeit for infinite words, was shown decidable in [5] using automata with counters. These automata were further developed into the theory of cost functions initiated by Colcombet in [10]. The decidability result from [5] entails decidability of the star height problem.

The difficulty of MSO+U comes from the interaction between the unbounding quantifier and quantification over possibly infinite sets. This motivated the study of WMSO+U, which is the variant of MSO+U where set quantification is restricted to finite sets. On infinite words, satisfiability of WMSO+U is decidable, and the logic has an automaton model [3]. Similar results hold for infinite trees [7]. The results from [7] have been used to decide properties of CTL* [9]. Currently, the strongest decidability result in this line is about WMSO+U on infinite trees extended with quantification over infinite paths [4]. The latter result entails decidability of problems such as the realisability problem for prompt LTL [13], deciding the winner in cost parity games [11], or deciding certain properties of energy games [8].

While the above results showed that fragments MSO+U can be decidable, and can be used to prove results not directly related to the logic itself, it was not known whether the full logic was decidable. The first evidence that MSO+U can be too expressive was given in [12], where it was shown that MSO+U can define languages of infinite words that are arbitrarily high in the projective hierarchy from descriptive set theory. This result was used in [6], where it was shown that, modulo a certain assumption from set theory (namely V=L), the MSO+U theory of the complete binary tree is undecidable. The result from [6] implies that there can be no algorithm which decides MSO+U on the complete binary tree, and which has a correctness proof in the ZFC axioms of set theory. This paper strengthens the result from [6] in two ways: first, we use no additional assumptions from set theory, and second, we prove undecidability for words and not trees.

2 Vector Sequences

It is clear that extending MSO by the ability to express precise equality of some quantities, like set sizes, immediately leads to undecidability. The idea behind our undecidability proof is to show that, under a certain encoding, MSO+U can express that two vector sequences have the same dimension. We begin by presenting some observations about vector sequences.

Define a *number sequence* to be an element of $\mathbb{N}^\omega$, and define a *vector sequence* to be an element of $(\mathbb{N}^*)^\omega$, i.e. an infinite sequence of vectors of natural numbers of possibly different dimensions. We write $\mathbf{f}, \mathbf{g}$ for vector sequences and f, g for number sequences. If f is a number sequence and $\mathbf{f}$ is a vector sequence, then we write $f \in \mathbf{f}$ if for every position i, the i-th number in the sequence f appears in one of the coordinates of the i-th vector in the vector sequence $\mathbf{f}$. For example, the relationship $f \in \mathbf{f}$ is satisfied by

$$f = 0, 0, 0, \dots \qquad \mathbf{f} = (0), (1, 0), (2, 1, 0), (3, 2, 1, 0), \dots$$

Number sequences are called *asymptotically equivalent* if they are bounded on the same sets of positions. For example, the sequence of squares is asymptotically equivalent to every number sequence with infinite $\liminf$. A vector sequence $\mathbf{f}$ is called an *asymptotic mix* of a vector sequence $\mathbf{g}$ if every $f \in \mathbf{f}$ is asymptotically equivalent to some $g \in \mathbf{g}$. A vector sequence of dimension d is one where all vectors have dimension d.

▶ **Lemma 2.1.** *Let $d \in \mathbb{N}$. There exists a vector sequence of dimension d which is not an asymptotic mix of any vector sequence of dimension $d-1$.*

Proof. In the definitions above, a *sequence* is a family indexed by natural numbers – formally, a function from the indexing set $\mathbb{N}$ to some universe. Since the definition of asymptotic mix does not use the order structure of the indexing set, in the proof of this lemma we allow families to be indexed by other countable sets, namely by vectors of natural numbers. All notions introduced above lift to the setting of families indexed by a fixed countable set. By induction on d, we will prove the following claim about vector families indexed by $\mathbb{N}^d$. We claim that the d-dimensional identity

$$\mathbf{id} : \mathbb{N}^d \to \mathbb{N}^d,$$

is not an asymptotic mix of any vector family

$$\mathbf{g} : \mathbb{N}^d \to \mathbb{N}^{d-1}.$$

The induction base of $d = 1$ is vacuous. Let us prove the claim for dimension d assuming that it has been proved for smaller dimensions.

Toward a contradiction, suppose that the d-dimensional identity is an asymptotic mix of some $\mathbf{g} : \mathbb{N}^d \to \mathbb{N}^{d-1}$. Consider the subset of arguments $\{0\} \times \mathbb{N}^{d-1}$. The first coordinate of the d-dimensional identity is bounded on this subset, namely it is zero, and therefore there must be some $g \in \mathbf{g}$ which is bounded on this set. By permuting the vectors in $\mathbf{g}$, without loss of generality, we assume that the first coordinate of $\mathbf{g}$ is bounded on arguments from $\{0\} \times \mathbb{N}^{d-1}$. Let

$$\mathbf{g}' : \mathbb{N}^d \to \mathbb{N}^{d-2}$$

be the vector family obtained from $\mathbf{g}$ by removing the first coordinate. Let

$$\pi_i : \mathbb{N}^d \to \mathbb{N} \qquad \text{with } i \in \{2, \ldots, d\}$$

be the projection onto the i-th coordinate, which satisfies $\pi_i \in \mathbf{id}$. Therefore, each π_i must be asymptotically equivalent to some $g_i \in \mathbf{g}$. Let $X_i \subseteq \mathbb{N}^d$ be the set of arguments x where g_i agrees with the first coordinate of $\mathbf{g}$. In other words, when restricted to arguments outside X_i the projection π_i is asymptotically equivalent to some $g_i \in \mathbf{g}'$. Since the first coordinate of $\mathbf{g}$ is bounded on the set $\{0\} \times \mathbb{N}^{d-1}$, it follows that there is some $c_i \in \mathbb{N}$ such that X_i does not contain any arguments which have zero on the first coordinate and at least c_i on the i-th coordinate. Taking c to be the maximum of all $c_2, \ldots, c_d$, we see that none of the sets $X_2, \ldots, X_d$ intersects the set

$$X = \{(0, n_2, \ldots, n_d) : n_2, \ldots, n_d \geq c\}.$$

It is easy to observe that the vector family

$$(0, n_2, \ldots, n_d) \in X \qquad \mapsto \qquad (n_2, \ldots, n_d) \tag{1}$$

is an asymptotic mix of $\mathbf{g}'$ (restricted to X), which is a vector family of dimension $d-2$. This contradicts the induction assumption, because the vector family in (1) is the $(d-1)$-dimensional identity, up to reindexing. ◀

A vector sequence is said to have *bounded dimension* if there is some d such that all vectors in the sequence have dimension at most d. A vector sequence is said to *tend to*

infinity if for every n, all but finitely many vectors in the sequence have all entries at least n. We order vector sequences coordinatewise in the following way: we write $\mathbf{f} \leq \mathbf{g}$ if for every i, the i-th vectors in both sequences have the same dimension, and the i-th vector of $\mathbf{f}$ is coordinatewise smaller or equal to the i-th vector of $\mathbf{g}$. A corollary of the above lemma is the following lemma, which characterises dimensions in terms only of boundedness properties.

▶ **Lemma 2.2.** *Let $\mathbf{f}_1, \mathbf{f}_2$ be vector sequences of bounded dimensions which tend to infinity. Then the following conditions are equivalent:*

1. *on infinitely many positions $\mathbf{f}_1$ has a vector of higher dimension than $\mathbf{f}_2$;*
2. *there exists some $\mathbf{g}_1 \leq \mathbf{f}_1$ which is not an asymptotic mix of any $\mathbf{g}_2 \leq \mathbf{f}_2$.*

Proof. Say that two vector sequences are asymptotically equivalent if they have the same dimension d, and for each coordinate $i \in \{1, \ldots, d\}$ the corresponding number sequences are asymptotically equivalent. Vector sequences that tend to infinity are maximal with respect to asymptotical equivalence in the following sense: if a vector sequence $\mathbf{f}$ of fixed dimension d tends to infinity, then for every vector sequence $\mathbf{h}$ of the same dimension there exists an asymptotically equivalent vector sequence $\mathbf{g} \leq \mathbf{f}$ (to obtain such $\mathbf{g}$, on each coordinate of each position we can take the minimum of the two numbers appearing in this place in $\mathbf{f}$ and $\mathbf{h}$). A corollary of this observation is that if $\mathbf{f}_2$ is a vector sequence of bounded dimension which tends to infinity, then every vector sequence at each (or at each except finitely many) position having dimension smaller or equal to the dimension of $\mathbf{f}_2$ is an asymptotic mix of some $\mathbf{g}_2 \leq \mathbf{f}_2$. This corollary gives the implication 2→1 in the lemma.

For the implication 1→2, we use Lemma 2.1. Let d_1 be such that on an infinite set $X \subseteq \mathbb{N}$ of positions $\mathbf{f}_1$ has dimension d_1 and $\mathbf{f}_2$ has a smaller dimension. By Lemma 2.1, there is a vector sequence

$$\mathbf{h} : X \to \mathbb{N}^{d_1}$$

of dimension d_1 which is not an asymptotic mix of any vector sequence of smaller dimension. As we have observed, $\mathbf{h}$ is asymptotically equivalent to some $\mathbf{g}_1 \leq \mathbf{f}_1$ (when restricted to positions from X), because $\mathbf{f}_1$ tends to infinity on all coordinates. Therefore, $\mathbf{g}_1$ is not an asymptotic mix of any $\mathbf{g}_2 \leq \mathbf{f}_2$ on X, since such a vector sequence $\mathbf{g}_2$ has strictly smaller dimension. We can arbitrarily extend $\mathbf{g}_1$ to all positions outside of X, and still it will not be an asymptotic mix of any $\mathbf{g}_2 \leq \mathbf{f}_2$. ◀

3 Encoding a Minsky Machine

We now use the results on vector sequences from the previous section to prove undecidability of MSO+U. To do this, it will be convenient to view an infinite word as a sequence of finite trees of bounded depth, in the following sense. Consider a word

$$w \in \{1, 2, 3, \ldots, n\}^\omega$$

which has infinitely many 1's. We view such a word as an infinite sequence of trees of depth (at most) n, denoted by $\mathsf{tree}(w)$, as described in Figure 1.

The key to the undecidability proof is the following lemma, which says that, in a certain asymptotic sense, degrees can be compared for equality. Here the degree of a tree node is defined to be the number of its children.

▶ **Lemma 3.1.** *There is an* MSO+U *formula, which defines the set of words*

$$w \in \{1, 2, 3\}^\omega$$

which have infinitely many 1's and such that $\mathsf{tree}(w)$ *has the following properties:*

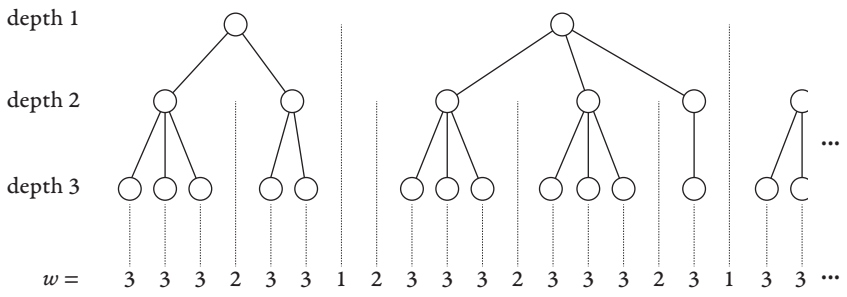


Figure 1 An example of $\mathsf{tree}(w)$ for $n = 3$. Formally speaking, the leaves of $\mathsf{tree}(w)$ are positions with label n, while the tree structure is defined by the following rule. For $1 \leq i < n$, two leaves which correspond to positions x and y with label n have a common ancestor at depth i if and only if there is no position between x and y which has label in $\{1, \ldots, i\}$. In particular, if between x and y there is a position with label 1, then x and y are in different trees of the sequence. Note that the mapping $w \mapsto \mathsf{tree}(w)$ is not one-to-one, e.g. in the picture, the first 2 just after the first 1 could be removed from w without affecting $\mathsf{tree}(w)$.

(a) *the degree of depth-2 nodes tends to infinity;*
(b) *all but finitely many nodes of depth 1 have the same degree.*

Proof. Condition (a) is easily seen to be expressible in MSO+U. One says that for every infinte set of depth-2 nodes, their degrees are unbounded.

Let us focus on condition (b). Fix a word w with infinitely many 1's as in the statement of the lemma. For an infinite set X of depth-1 nodes, define

$$\mathbf{f}_X : \mathbb{N} \to \mathbb{N}^*$$

to be the vector sequence, where the i-th vector is the sequence of degrees of the children of the i-th node from X. Condition (a) says that if X is the set of all depth-1 nodes, then $\mathbf{f}_X$ tends to infinity, which implies that $\mathbf{f}_X$ also tends to infinity for any other infinite set X of depth-1 nodes.

Call two sets X, Y of depth-1 nodes *alternating* if every two nodes in X are separated by a node in Y, and vice versa. Condition (b) is equivalent to saying that

- depth-1 nodes have bounded degree;
- one cannot find infinite alternating sets X, Y of depth-1 nodes such that infinitely often $\mathbf{f}_X$ has strictly bigger dimension than $\mathbf{f}_Y$.

The first condition is clearly expressible in MSO+U. The second condition, thanks to Lemma 2.2, can be restated as: one cannot find infinite alternating sets X, Y of depth-1 nodes such that there is some $\mathbf{g}_X \leq \mathbf{f}_X$ which is not an asymptotic mix of any $\mathbf{g}_Y \leq \mathbf{f}_Y$. This is expressible in MSO+U (the quantification over vector sequences $\mathbf{g}_X \leq \mathbf{f}_X$ amounts to selecting a subset of depth-3 nodes that are descendants of nodes in X). ◀

Minsky Machines

To prove undecidability, we reduce emptiness for Minsky machines to deciding MSO+U. By a Minsky machine we mean a (possibly nondeterministic) device which has a finite state space, and two counters that can be incremented, decremented, and tested for zero. It is undecidable whether a given Minsky machine has an accepting run, i.e. one which begins in a designated initial state with zero on both counters, and ends in a designated final state.

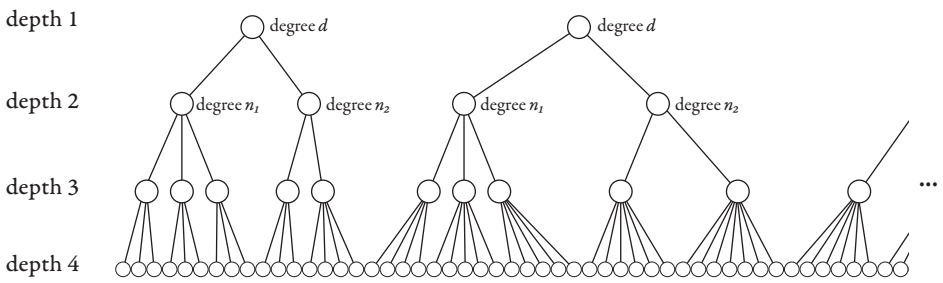


Figure 2 A sequence of trees as in Lemma 3.2. Here $d = 2$, $n_1 = 3$, and $n_2 = 2$.

Let ρ be a finite run of a Minsky machine of length d. We say that a vector of natural numbers $(n_1, \ldots, n_{2d})$ *describes* the run ρ if, for $i = 1, \ldots, d$, the numbers n_{2i-1}, n_{2i} store the value of the two counters in the i-th configuration of ρ. Note that this description does not specify fully the run ρ, as the state information is missing. The following lemma contains the reduction of Minsky machine emptiness to satisfiability of MSO+U.

▶ **Lemma 3.2.** *For every Minsky machine, one can compute a formula of* MSO+U *which defines the set of words*

$$w \in \{1, 2, 3, 4\}^\omega$$

which have infinitely many 1's and such that $\mathsf{tree}(w)$ *has the following properties, which are illustrated in Figure 2:*

(a) *the degree of depth-3 nodes tends to infinity;*
(b) *all but finitely many depth-1 nodes have the same degree d;*
(c) *for every $i \in \{1, \ldots, d\}$, all but finitely many depth-2 nodes that are an i-th child have the same degree, call it n_i;*
(d) *$n_1 - 1, \ldots, n_d - 1$ describe some accepting run of the Minsky machine.*

Proof. Condition (a) is clearly expressible in MSO+U.

We say that a sequence of trees of depth 3 is *well-formed* if the degree of depth-2 nodes tends to infinity, and that it has *almost constant degree* if all but finitely many depth-1 nodes have the same degree. Lemma 3.1 says that MSO+U can express the conjunction of being well-formed and having almost constant degree. We will use this property to define conditions (b), (c) and (d).

Define the *flattening* of $\mathsf{tree}(w)$ to be the sequence of depth-3 trees obtained from $\mathsf{tree}(w)$ by removing all depth-3 nodes and connecting all depth-4 nodes directly to their depth-2 grandparents. By condition (a), the flattening is well-formed. Since the flattening does not change the degree of depth-1 nodes, condition (b) is the same as saying that the flattening has almost constant degree, and therefore can be expressed in MSO+U thanks to Lemma 3.1.

Define a *depth-2 selector with offset i* to be a set of nodes X in the tree $\mathsf{tree}(w)$ which selects exactly one child for every depth-1 node (and therefore X contains only depth-2 nodes), and all but finitely many nodes in X are an i-th child. A *depth-2 selector*, without i being mentioned, is a depth-2 selector for some i. Being a depth-2 selector is equivalent to saying that one gets a well-formed sequence of almost constant degree if one keeps only nodes from $X_\leftarrow$ and their descendants, where $X_\leftarrow$ is the set of nodes of depth 2 that have a sibling from X to the right. Therefore, being a depth-2 selector is definable in MSO+U. Condition (c) is the same as saying that for every depth-2 selector X, if one only keeps the

nodes from X and their descendants, then the resulting sequence has almost constant degree, which can be expressed in MSO+U thanks to Lemma 3.1.

We are left with condition (d) about Minsky machines. We say that a depth-2 selector X *represents zero*, if all but finitely many nodes in X have degree one (recall that condition (d) uses $n_i - 1$ to represent a counter value, because a depth-2 node cannot have degree zero). Representing zero is definable in first-order logic. If X, Y are selectors, we say that Y *increments* X if there is some n such that all but finitely many nodes in X have degree n, and all but finitely many nodes in Y have degree $n+1$. This is equivalent to saying that if one keeps only nodes from $X \cup Y$ and their descendants, and then removes one subtree of every node from Y, then the resulting sequence of depth-3 trees has almost constant degree. Therefore incrementation is definable in MSO+U (technically, one needs to translate the above tree properties to word properties, via the encoding from Figure 1). Using formulas for representing zero and incrementation, it is easy to formalise condition (d) in MSO+U (the formula first guesses the missing state information to fully specify the run ρ, and then verifies its consistency with the Minsky machine). ◀

In particular, the formula computed in Lemma 3.2 is satisfiable if and only if the Minsky machine has an accepting run. This yields undecidability of MSO+U on infinite words, which is the same as our main Theorem 1.1.

Quantifier Complexity

Here we examine the quantification structure of the formulas in the undecidability proof. We count the number of blocks of quantifiers of same type. We do not claim that the formulas are optimal.

The more interesting part of the formula in Lemma 3.1 says: for all sets X, Y and functions $\mathbf{g}_X$ there exists a function $\mathbf{g}_Y$ such that $\mathbf{g}_X$ is an asymptotic mix of $\mathbf{g}_Y$. The condition "$\mathbf{g}_X$ is an asymptotic mix of $\mathbf{g}_Y$" is expressed as: for every $g_X \in \mathbf{g}_X$ there exists $g_Y \in \mathbf{g}_Y$ such that for every set of positions Z either both g_X and g_Y are bounded on Z or none of them is. Thus the entire formula has six blocks of quantifiers, starting from universal quantifiers (where the most internal quantifiers are U and negations of U).

The formula from Lemma 3.2 says: there exists an infinite word (a labelling of $(\mathbb{N}, \leq)$) and a labelling by states of the Minsky machine, such that for every set X either X is not a depth-2 selector or the children selected by X satisfy appropriate conditions. Saying that X is not a depth-2 selector amounts to using the formula from Lemma 3.1 negatively, starting from an existential quantifiers. The rest of the condition about X says that all but finitely many nodes in X have the same degree, and that the degree of these nodes is smaller/greater by one than the degree of the left siblings of nodes in X, which is expressible by using the formula from Lemma 3.1 positively. Concluding, the whole formula uses eight nested blocks of quantifiers: seven blocks of alternating existential and universal quantifiers, ended by quantifiers U and negations of U.

References

1 Achim Blumensath, Olivier Carton, and Thomas Colcombet. Asymptotic monadic second-order logic. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 – 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 87–98. Springer, 2014. doi:10.1007/978-3-662-44522-8_8.

2 Mikołaj Bojańczyk. A bounding quantifier. In *Computer Science Logic, 18th International Workshop, CSL 2004, 13th Annual Conference of the EACSL, Karpacz, Poland, September 20-24, 2004, Proceedings*, pages 41–55, 2004. doi:10.1007/978-3-540-30124-0_7.

3 Mikołaj Bojańczyk. Weak MSO with the unbounding quantifier. *Theory Comput. Syst.*, 48(3):554–576, 2011. doi:10.1007/s00224-010-9279-2.

4 Mikołaj Bojańczyk. Weak MSO+U with path quantifiers over infinite trees. In *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 38–49, 2014. doi:10.1007/978-3-662-43951-7_4.

5 Mikołaj Bojańczyk and Thomas Colcombet. Bounds in w-regularity. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 285–296, 2006. doi:10.1109/LICS.2006.17.

6 Mikołaj Bojańczyk, Tomasz Gogacz, Henryk Michalewski, and Michał Skrzypczak. On the decidability of MSO+U on infinite trees. In *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 50–61, 2014. doi:10.1007/978-3-662-43951-7_5.

7 Mikołaj Bojańczyk and Szymon Toruńczyk. Weak MSO+U over infinite trees. In *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th – March 3rd, 2012, Paris, France*, pages 648–660, 2012. doi:10.4230/LIPIcs.STACS.2012.648.

8 Tomás Brázdil, Krishnendu Chatterjee, Antonín Kucera, and Petr Novotný. Efficient controller synthesis for consumption games with multiple resource types. In *CAV'12*, pages 23–38, 2012. doi:10.1007/978-3-642-31424-7_8.

9 Claudia Carapelle, Alexander Kartzow, and Markus Lohrey. Satisfiability of CTL* with constraints. In *CONCUR 2013 – Concurrency Theory – 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings*, pages 455–469, 2013. doi:10.1007/978-3-642-40184-8_32.

10 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part II*, pages 139–150, 2009. doi:10.1007/978-3-642-02930-1_12.

11 Nathanaël Fijalkow and Martin Zimmermann. Cost-parity and cost-Streett games. In *FSTTCS'12*, pages 124–135, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.124.

12 Szczepan Hummel and Michał Skrzypczak. The topological complexity of MSO+U and related automata models. *Fundam. Inform.*, 119(1):87–111, 2012. doi:10.3233/FI-2012-728.

13 Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. From liveness to promptness. *Formal Methods in System Design*, 34(2):83–103, 2009. doi:10.1007/s10703-009-0067-z.

14 Szymon Toruńczyk. Languages of profinite words and the limitedness problem. In *Automata, Languages, and Programming – 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, pages 377–389, 2012. doi:10.1007/978-3-642-31585-5_35.

Time-Approximation Trade-offs for Inapproximable Problems

Édouard Bonnet[1], Michael Lampis[2], and Vangelis Th. Paschos[3]

1 **Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary**
bonnet.edouard@sztaki.mta.hu

2 **PSL Research University, Université Paris Dauphine, LAMSADE, CNRS UMR7243, Paris, France**
michail.lampis@dauphine.fr

3 **PSL Research University, Université Paris Dauphine, LAMSADE, CNRS UMR7243, Paris, France**
paschos@lamsade.dauphine.fr

Abstract

In this paper we focus on problems which do not admit a constant-factor approximation in polynomial time and explore how quickly their approximability improves as the allowed running time is gradually increased from polynomial to (sub-)exponential.

We tackle a number of problems: For Min Independent Dominating Set, Max Induced Path, Forest and Tree, for any $r(n)$, a simple, known scheme gives an approximation ratio of r in time roughly $r^{n/r}$. We show that, for most values of r, if this running time could be significantly improved the ETH would fail. For Max Minimal Vertex Cover we give a non-trivial $\sqrt{r}$-approximation in time $2^{n/r}$. We match this with a similarly tight result. We also give a $\log r$-approximation for Min ATSP in time $2^{n/r}$ and an r-approximation for Max Grundy Coloring in time $r^{n/r}$.

Furthermore, we show that Min Set Cover exhibits a curious behavior in this super-polynomial setting: for any $\delta > 0$ it admits an m^{δ}-approximation, where m is the number of sets, in just quasi-polynomial time. We observe that if such ratios could be achieved in polynomial time, the ETH or the Projection Games Conjecture would fail.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases Algorithm, Complexity, Polynomial and Subexponential Approximation, Reduction, Inapproximability

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.22

1 Introduction

One of the central questions in combinatorial optimization is how to deal efficiently with NP-hard problems, with approximation algorithms being one of the most widely accepted approaches. Unfortunately, for many optimization problems, even approximation has turned out to be hard to achieve in polynomial time. This has naturally led to a more recent turn towards super-polynomial and sub-exponential time approximation algorithms. The goal of this paper is to contribute to a systematization of this line of research, while adding new positive and negative results for some well-known optimization problems.

For many of the most paradigmatic NP-hard optimization problems the best polynomial-time approximation algorithm is known (under standard assumptions) to be the trivial

© Édouard Bonnet, Michael Lampis and Vangelis Th. Paschos;
licensed under Creative Commons License CC-BY

algorithm. In the super-polynomial time domain, these problems exhibit two distinct types of behavior. On the one hand, APX-complete problems, such as MAX-3SAT, have often been shown to display a "sharp jump" in their approximability. In other words, the only way to obtain any improvement in the approximation ratios for such problems is to accept a fully exponential running time, unless the Exponential Time Hypothesis (ETH) is false [22].

A second, more interesting, type of behavior is displayed on the other hand by problems which are traditionally thought to be "very inapproximable", such as Clique. For such problems it is sometimes possible to improve upon the (bad) approximation ratios achievable in polynomial time with algorithms running only in *sub-exponential* time. In this paper, we concentrate on such "hard" problems and begin to sketch out the spectrum of trade-offs between time and approximation that can be achieved for them.

On the algorithmic side, the goal of this paper is to design *time-approximation trade-off schemes*. By this, we mean an algorithm which, when given an instance of size n and an (arbitrary) approximation ratio $r > 1$ as a target, produces an r-approximate solution in time $T(n, r)$. The question we want to answer is what is the best function $T(n, r)$, for each particular value of r. Put more abstractly, we want to sketch out, as accurately as possible, the Pareto curve that describes the best possible relation between worst-case approximation ratio and running time for each particular problem. For several of the problems we examine the best known trade-off algorithm is some simple variation of brute-force search in appropriately sized sets. For some others, we present trade-off schemes with much better performance, using ideas from exponential-time and parameterized algorithms, as well as polynomial-time approximation.

Are the trade-off schemes we present optimal? A naive way to answer this question could be to look at an extreme, already solved case: set r to a value that makes the running time polynomial and observe that the approximation ratios of our algorithms generally match (or come close to) the best-known polynomial-time approximation ratios. However, this observation does not alone imply satisfactorily the optimality of a trade-off scheme: it leaves open the possibility that much better performance can be achieved when r is restricted to a different range of values. Thus, the second, perhaps more interesting, direction of this paper is to provide lower bound results (almost) matching several of our algorithms *for any point in the trade-off curve*. For a number of problems, these results show that the known schemes are (essentially) the best possible algorithms, everywhere in the domain between polynomial and exponential running time. We stress that we obtain these much stronger *sub-exponential inapproximability* results relying only on standard, appropriately applied, PCP machinery, as well as the ETH.

Previous work. Moderately exponential and sub-exponential approximation algorithms are relatively new topics, but most of the standard graph problems have already been considered in the trade-off setting of this paper. For Max Independent Set and Min Coloring an r-approximation in time $c^{n/r}$ was given by Bourgeois et al. [5, 3]. For Min Set Cover, a $\log r$-approximation in time $c^{n/r}$ and an r-approximation in time $c^{m/r}$, where n, m are the number of elements and sets respectively, were given by Cygan, Kowalik and Wykurz [8, 4]. For Min Independent Dominating Set an r-approximation in $c^{n \log r/r}$ is given in [2]. An algorithm with similar performance is given for Bandwidth in [9] and for Capacitated Dominating Set in [10]. In all the results above, c denotes some appropriate constant.

On the hardness side, the direct inspiration of this paper is the recent work of Chalermsook, Laekhanukit and Nanongkai [6] where the following was proved.

▶ **Theorem 1** ([6]). *For all $\varepsilon > 0$, for all sufficiently large $r = O(n^{1/2-\varepsilon})$, if there exists an r-approximation for* Max Independent Set *running in $2^{n^{1-\varepsilon}/r^{1+\varepsilon}}$ then there exists a randomized sub-exponential algorithm for* 3-SAT.

Theorem 1 essentially showed that the very simple approximation scheme of [5] is probably "optimal", up to an arbitrarily small constant in the second exponent, for a large range of values of r (not just for polynomial time). The hardness results we present in this paper follow the same spirit and in fact also rely on the technique of appropriately combining PCP machinery with the ETH, as was done in [6]. To the best of our knowledge, Max Independent Set and Max Induced Matching (for which similar results are given in [6]) are the only problems for which the trade-off curve has been so accurately bounded. The only other problem for which the optimality of a trade-off scheme has been investigated is Min Set Cover. For this problem the work of Moshkovitz [21] and Dinur and Steurer [12] showed that there is a constant $c > 0$ such that $\log r$-approximating Min Set Cover requires time $2^{(n/r)^c}$. It is not yet known if this constant c can be brought arbitrarily close to 1.

Summary of results

In this paper we want to give upper and lower bound results for trade-off schemes that match as well as the algorithm of [5] and Theorem 1 do for Max Independent Set; we achieve this for several problems (all of them are defined in Appendix).

- For Min Independent Dominating Set, there is no r-approximation in $2^{n^{1-\varepsilon}/r^{1+\varepsilon}}$ for any r, unless the *deterministic* ETH fails. This result is achieved with a direct reduction from a quasi-linear PCP and is stronger than the corresponding result for Max Independent Set (Theorem 1) in that the reduction is deterministic and works for all r.
- For Max Induced Path, there is no r-approximation in $2^{o(n/r)}$ for any $r < n$, unless the deterministic ETH fails. This is shown with a direct reduction from 3-SAT, which gives a sharper running time lower bound. For Max Induced Tree and Forest we show hardness results similar to Theorem 1 by reducing from Max Independent Set.
- For Max Minimal Vertex Cover we give a scheme that returns a $\sqrt{r}$-approximation in time $c^{n/r}$, for any $r > 1$. We complement this with a reduction from Max Independent Set which establishes that a $\sqrt{r}$-approximation in time $2^{n^{1-\varepsilon}/r^{1+\varepsilon}}$ (for any r) would disprove the randomized ETH.
- For Min ATSP we adapt the classical $\log n$-approximation into a $\log r$-approximation in $c^{n/r}$. For Max Grundy Coloring we give a simple r-approximation in $c^{n/r}$. For both problems membership in APX is still an open problem.
- Finally, we consider Min Set Cover. Its approximability in terms of m is poorly understood, even in polynomial time. With a simple refinement of an argument given in [23] we show how to obtain for any $\delta > 0$ an m^δ-approximation in quasi-polynomial time $2^{\log^{(1-\delta)/\delta} n}$. We also observe that, if the ETH and the Projection Games Conjecture [21] are true, there exists $c > 0$ such that m^c-approximation cannot be achieved in polynomial time. This would imply that the approximability of Min Set Cover changes dramatically from polynomial to quasi-polynomial time. The only other problem which we know to exhibit this behavior is Graph Pricing [6].

2 Preliminaries and Baseline Results

Algorithms

In this paper we consider time-approximation trade-off schemes. Such a scheme is an algorithm that, given an input of size n and a parameter r, produces an r-approximate

solution (that is, a solution guaranteed to be at most a factor r away from optimal) in time $T(n, r)$. Sometimes we will overload notation and allow trade-off schemes to have an approximation ratio that is some other function of r, if this makes the function $T(n, r)$ simpler. We begin with an easy, generic, such scheme, that simply checks all subsets of a certain size.

▶ **Theorem 2.** *Let* Π *be an optimization problem on graphs, for which the solution is a set of vertices and feasibility of a solution can be verified in polynomial time. Suppose that* Π *satisfies one of the following sets of conditions:*

1. *The objective is* min *and some solution can be produced in polynomial time.*
2. *The objective is* max *and for any feasible solution* S *there exists* $u \in S$ *such that* $S \setminus \{u\}$ *is also feasible (weak monotonicity).*

Then, for any $r > 1$ *(that may depend on the order* n *of the input) there exists an* r*-approximation for* Π *running in time* $O^*((er)^{n/r})$.

Proof. The algorithm simply tries all sets of vertices of size up to n/r. These are at most $n/r\binom{n}{r} = O^*((er)^{n/r})$. Each set is checked for feasibility and the best feasible set is picked. In the case of minimization problems, either we will find the optimal solution, or all solutions contain at least n/r vertices, so an arbitrary solution (which can be produced in polynomial time) is an r-approximation. In the case of maximization, the weak monotonicity condition ensures that there always exists a feasible solution of size at most n/r. ◀

Because of Theorem 2, we will treat this kind of qualitative trade-off performance (r approximation in time exponential in $n \log r/r$) as a "baseline". It is, however, not trivial if this performance can be achieved for other types of graph problems (e.g. ordering problems). Let us also note that, for maximization problems that satisfy strong monotonicity (all subsets of a feasible solution are feasible) the running time of Theorem 2 can be improved to $O^*(2^{n/r})$ [5].

Hardness

The Exponential Time Hypothesis (ETH) [16] is the assumption that there is no $2^{o(n)}$-algorithm that decides 3-SAT instances of size n. All of our hardness results rely on the ETH or the (stronger) randomized ETH, which states the same for randomized algorithms.

For most of our hardness results we also make use of known quasi-linear PCP constructions. Such constructions reduce 3-SAT instances of size n into CSPs with size $n \log^{O(1)} n$, so that there is a gap between satisfiable and unsatisfiable instances. Assuming the ETH, these constructions give a problem that cannot be approximated in time $2^{o(n/\log^{O(1)} n)}$ which we often prefer to write as $2^{n^{1-\varepsilon}}$, though this makes the lower bound slightly weaker. We note that, because of the poly-logarithmic factor added by even the most efficient known PCPs, current techniques are often unable to distinguish between whether the optimal running time for r-approximating a problem is, say $2^{n/r}$ or $r^{n/r}$. The existence of linear PCPs, which at the moment is open, could help further our understanding in this direction. To make the sections of this paper more independent, we will cite the PCP theorems we use as needed.

3 Min Independent Dominating Set

The result of this section is a reduction showing that for Min Independent Dominating Set, no trade-off scheme can significantly beat the baseline performance of Theorem 2, which qualitatively matches the best known scheme for this problem [2]. Thus, in a sense Min Independent Dominating Set is an "inapproximable" problem in sub-exponential time.

Interestingly, Min Independent Dominating Set was among the first problems to be shown to be inapproximable in both polynomial time [15] and FPT time [13].

To show our hardness result, we will need an almost linear PCP construction with perfect completeness. Such a PCP was given by Dinur [11].

▶ **Lemma 3** ([11], Lemma 8.3). *There exist constants $c_1, c_2 > 0$ and a polynomial time reduction that transforms any SAT instance ϕ on n variables with $m = O(n)$ clauses, into a constraint graph $G = \langle (V, E), \Sigma, \mathcal{C} \rangle$ such that:*

- $|V| + |E| \leqslant n(\log n)^{c_1}$ *and* Σ *is of constant size.*
- *If ϕ is satisfiable, then UNSAT$(G) = 0$.*
- *If ϕ is not satisfiable, then UNSAT$(G) \geqslant 1/(\log n)^{c_2}$.*

Let us recall the relevant definitions from [11]. A constraint graph is a CSP whose variables are the vertices of G and take values over Σ. All constraints have arity 2 and correspond to the edges of E; with each constraint C_e we associate a set of satisfying assignments from Σ^2. UNSAT(G) is the fraction of unsatisfied constraints that correspond to the optimal assignment to V. Observe that we only need here a PCP theorem where UNSAT(G) is at least inverse poly-logarithmic in n (rather than constant). The important property we need for our reduction is perfect completeness (that is, UNSAT$(G) = 0$ in the YES case).

▶ **Theorem 4.** *Under ETH, for any $\varepsilon > 0$ and $r \leqslant n$, an r-approximation for* Min Independent Dominating Set *cannot take time $O^*(2^{n^{1-\varepsilon}/r^{1+\varepsilon}})$.*

Proof. Let $G = \langle (V, E), \Sigma, \mathcal{C} = \{C_e : e \in E\} \rangle$ be the constraint graph obtained from any SAT formula ϕ, applying the above lemma. Let $s = |\Sigma|$, $n = |V|$ and $m = |E|$. We define an instance $G' = (V', E')$ of Min Independent Dominating Set in the following way. For each vertex $v \in V$ and $a \in \Sigma$, we add a vertex $w_{v,a}$ in V'. For each v, the s vertices $w_{v,1}, w_{v,2}, \ldots, w_{v,s}$ are pairwise linked in G' together with a dummy vertex $w_{v,0}$ and form a clique denoted by C_v. The idea would naturally be that taking $w_{v,a}$ in the independent dominating set corresponds to coloring v by a. For each edge $e = uv \in E$, and for each satisfying assignment $(i, j) \in C_e$ we add an independent set $I_{e,(i,j)}$ of r' vertices in V' (for some r' that will be specified later), we link $w_{u,i}$ to all the vertices of the independent sets $I_{e,(i',j')}$ where $i' \in \Sigma \setminus \{i\}$ (and $j' \in \Sigma$), and we link $w_{v,j}$ to all the vertices of the independent sets $I_{e,(i',j')}$ where $(i', j) \in C_e$. We finally add, for each edge $e = uv$, an independent set I_e of r' vertices, and we link $w_{u,i}$ to all the vertices of I_e if there is a pair $(i, j) \in C_e$ for some $j \in \Sigma$.

If ϕ is satisfiable, then UNSAT$(G) = 0$, so there is a coloring $c : V \to \Sigma$ satisfying all the edges. Thus, $\bigcup_{v \in V} \{w_{v,c(v)}\}$ is an independent dominating set of size n. It is independent since there is no edge between $w_{v,a}$ and $w_{v',a'}$ whenever $v \neq v'$. It dominates $\bigcup_{v \in V} C_v$ since one vertex is taken per clique. It also dominates I_e for every edge e, by construction. We finally have to show that all the independent sets $I_{uv,(i,j)}$ are dominated. If $c(u) \neq i$, then $I_{uv,(i,j)}$ is dominated by $w_{u,c(u)}$ (since $(c(u), c(v)) \in C_e$). We now assume that $c(u) = i$. Then $I_{uv,(i,j)}$ is dominated by $w_{v,c(v)}$, since $(c(u), c(v)) \in C_e$.

If ϕ is not satisfiable, then UNSAT$(G) \geqslant 1/(\log n)^{c_2}$. Any independent dominating set S has to take one vertex per clique C_v (to dominate the dummy vertex $w_{v,0}$). Let A be $S \cap \bigcup_{v \in V} C_v$, and let $c : V \to \Sigma$ be the coloring corresponding to A. Coloring c does not satisfy at least $m/(\log n)^{c_2}$ edges. Let $E'' \subseteq E$ be the set of unsatisfied edges. For each edge $e = uv \in E''$, let us show that at least one independent set of the form $I_{uv,(i,j)}$ is not dominated by A. We may first observe that $I_{uv,(i,j)}$ can only be dominated by $w_{u,c(u)}$ or by $w_{v,c(v)}$. If there is no pair $(c(u), j') \in C_e$ for any j', then I_e is not dominated by

construction. If there is a pair $(c(u), j') \in C_e$ for some j', then $I_{e,(c(u),j')}$ is not dominated by $w_{u,c(u)}$ by construction, and is not dominated by $w_{v,c(v)}$ since $(c(u), c(v)) \notin C_e$.

The only way of dominating those independent sets is to add to the solution all the vertices composing them, so a minimum independent dominating set is of size at least $n + {r'm}/{(\log n)^{c_2}} \geqslant {r'n(\log n)^{c_1}}/{(\log n)^{c_2}} = rn$ setting $r' = {r(\log n)^{c_2}}/{(\log n)^{c_1}}$.

An r'-approximation for Min Independent Dominating Set can therefore decide the satisfiability of ϕ. The number of vertices in the instance of Min Independent Dominating Set is $n' = |V'| \leqslant (s+1)n + r'm(s^2+1) = O(nr'(\log n)^{c_1})$. So, for any $\varepsilon > 0$, if the r'-approximation algorithm for Min Independent Dominating Set runs in time $O^*(2^{{n'}^{1-\varepsilon}/{r'}^{1+\varepsilon}})$, it contradicts ETH. Renaming r' by r and n' by n, an r-approximation would not be possible in time $O^*(2^{n^{1-\varepsilon}/r^{1+\varepsilon}})$, for any $\varepsilon > 0$ and $r \leqslant n$. ◀

4 Max Minimal Vertex Cover

In this section we deal with the Max Minimal Vertex Cover problem, which is the dual of Min Independent Dominating Set (which is also known as Minimum Maximal Independent Set). Interestingly, this turns out to be (so far) the only problem for which its time-approximation trade-off curve can be well-determined, while being far from the baseline performance of Theorem 2. To show this result we first present an approximation scheme that relies on a classic idea from parameterized complexity: the exploitation of a small vertex cover.

▶ **Theorem 5.** *For any r such that $1 < r \leqslant \sqrt{n}$,* Max Minimal Vertex Cover *is r-approximable in time $O^*(2^{3n/r^2})$.*

Proof. Our r-approximation algorithm begins by calculating a maximal matching M of the input graph. If $|M| \geqslant {n}/{r}$ then the algorithm simply outputs any arbitrary minimal vertex cover of G. The solution, being a valid vertex cover, must have size at least $|M| \geqslant {n}/{r}$, and is therefore an r-approximation.

Otherwise, we partition the edges of M into r equal-sized groups arbitrarily. Let $V_i, 1 \leqslant i \leqslant r$ be the set of vertices matched by the edges in group i. By the bound on the size of M we have that $|V_i| \leqslant {2n}/{r^2}$. We use L to denote the set of vertices unmatched by M. Note that L is of course an independent set.

The basic building block of our algorithm is a procedure which, given an independent set I, builds a minimal vertex cover of G that does not contain any vertices of I. This can be done in polynomial time by first selecting $V \setminus I$ as a vertex cover of G, and then repeatedly removing from the cover redundant vertices one by one, until the solution is minimal. It is worthy of note here that this procedure guarantees the construction of a minimal vertex cover with size at least $|N(I)|$, where $N(I)$ is the set of vertices with a neighbor in I.

The algorithm now proceeds as follows: for each $i \in \{1, \ldots, r\}$ we iterate through all sets $S \subset V_i$ such that S is an independent set. For each such S we initially build the set $S' := S \cup (L \setminus N(S))$. In words, we add to S all its non-neighbors from L to obtain S', which is thus also an independent set. The algorithm then builds a minimal vertex cover of size at least $|N(S')|$ using the procedure of the previous paragraph. In the end we select the largest of the covers produced in this way.

The algorithm has the claimed running time. The number of independent sets contained in V_i is at most $2^{3n/r^2}$, since $G[V_i]$ has at most ${2n}/{r^2}$ vertices and contains a perfect matching. Everything else takes polynomial time.

Let us therefore check the approximation ratio. Fix an optimal solution and let $R_i, i \in \{1, \ldots, r\}$ be the set of vertices of V_i *not* selected by this solution. Also, let R_L be the vertices

of L not selected by the solution. Observe that $R := R_L \cup \bigcup_{1 \leqslant i \leqslant r} R_i$ is an independent set, and the solution has size opt $= |N(R)|$, because all vertices of the solution must have an unselected neighbor.

Observe now that there must exist an $i \in \{1, \ldots, r\}$ such that $|N(R_i \cup R_L)| \geqslant {}^{|N(R)|}/_{r}$. This is a consequence of the fact that for any two sets I_1, I_2 such that $I_1 \cup I_2$ is independent we have $N(I_1 \cup I_2) = N(I_1) \cup N(I_2)$. Now, since the algorithm iterated through all independent sets in V_i, it must have tried the set $S := R_i$. From this it built the independent set $S' := R_i \cup (L \setminus N(R_i))$. Observe that $S' \supseteq R_i \cup R_L$, because R_L does not contain any neighbors of R_i. It follows that $|N(S')| \geqslant |N(R_i \cup R_L)|$. Since the solution produced has size at least $|N(S')|$ we get the promised approximation ratio. ◀

The corresponding hardness result consists of a reduction from the Max Independent Set instances constructed in Theorem 1.

▶ **Theorem 6.** *Under randomized ETH, for any $\varepsilon > 0$ and $r \leqslant n^{1/2-\varepsilon}$, no r-approximation for* Max Minimal Vertex Cover *can take time $O^*(2^{{n^{1-\varepsilon}}/{r^{2+\varepsilon}}})$.*

The reduction is from Max Independent Set. However, we will need to rely on the structure of the instances produced for Theorem 1 in [6]. We restate here the relevant theorem:

▶ **Theorem 7** ([6], Theorem 5.2)**.** *For any sufficiently small $\varepsilon > 0$ and any $r \leqslant n^{1/2-\varepsilon}$, there is a randomized polynomial reduction, which, from an instance of* SAT *ϕ on n variables, builds a graph G with $n^{1+\varepsilon}r^{1+\varepsilon}$ vertices such that with high probability:*

- *If ϕ is a YES-instance, then $\alpha(G) \geqslant n^{1+\varepsilon}r$.*
- *If ϕ is a NO-instance, then $\alpha(G) \leqslant n^{1+\varepsilon}r^{2\varepsilon}$.*

Theorem 6. Let ϕ be any instance of SAT and $G = (V, E)$ be the graph built from ϕ with the reduction of Theorem 5.2 in [6]. Keeping the same notation, we add $\lceil r \rceil$ pendant vertices to each vertex of G and we call this new graph G'. The best solution for Max Minimal Vertex Cover in G' is to fix a maximum independent set I of G and to take the $\lceil r \rceil$ pendant vertices to each vertices of I, plus the vertices of $V \setminus I$. This is true since $\lceil r \rceil$ is at least 1. Let opt be the size of a largest minimal vertex cover.

If ϕ is a YES-instance, then $\alpha(G) \geqslant n^{1+\varepsilon}r$, and opt $> n^{1+\varepsilon}r^2$. If ϕ is a NO-instance, then $\alpha(G) \leqslant n^{1+\varepsilon}r^{2\varepsilon}$, and opt $< n^{1+\varepsilon}r^{1+2\varepsilon} + n^{1+\varepsilon}r^{1+\varepsilon} < 2n^{1+\varepsilon}r^{1+2\varepsilon}$. Therefore, an approximation with ratio $r' = {r^{1-2\varepsilon}}/{2}$ for Max Minimal Vertex Cover would permit to solve SAT. Assuming ETH, this cannot take time $2^{o(n)}$.

As $n' := |V(G')| = n^{1+\varepsilon}r^{2+\varepsilon}$, such an approximation would not be possible in time $2^{{n'^{1-\varepsilon}}/{r^{2+\varepsilon}}}$. Renaming r' by r and n' by n, an r-approximation would not be possible in time $O^*(2^{{n^{1-\varepsilon}}/{r^{2+6\varepsilon}}})$. ◀

5 Induced Path, Tree and Forest

In this section we study the Max Induced Path, Tree and Forest problems, where we are looking for the largest set of vertices inducing a graph of the respective type. These are all hard to approximate in polynomial time [17, 20], and we observe that an easy reduction from Max Independent Set shows that the generic scheme of Theorem 2 is almost tight in sub-exponential time for the latter two. However, the most interesting result of this section is a direct reduction we present from 3-SAT to Max Induced Path. This reduction allows us to establish inapproximability for this problem *without* the PCP theorem, thus eliminating the ε from the running time lower bound.

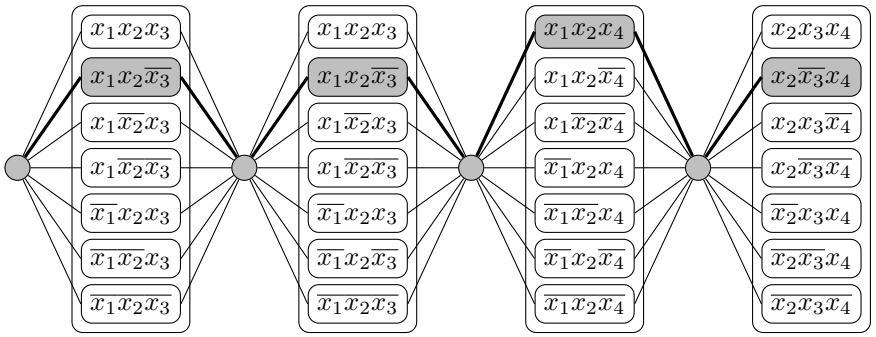


Figure 1 The graph H_1 built for the instance $\{x_1 \vee \neg x_2 \vee x_3, x_1 \vee x_2 \vee \neg x_3, \neg x_1 \vee x_2 \vee \neg x_4, x_2 \vee \neg x_3 \vee x_4\}$. G is obtained by laying end to end r copies of H_1. The rectangle boxes are the cliques C_i^j, and the contradicting edges are not shown. An induced path with $2m$ vertices is represented in gray and can be extended into one with $2rm$ vertices in G (the formula being satisfiable).

▶ **Theorem 8.** *Under ETH, for any $\varepsilon > 0$ and sufficiently large $r \leqslant n^{1/2-\varepsilon}$, an r-approximation for* Max Induced Forest *or* Max Induced Tree *cannot take time* $2^{n^{1-\varepsilon}/r^{1+\varepsilon}}$.

Proof. For Max Induced Forest we simply observe that, if $\alpha(G)$ is the size of the largest independent set of a graph, the largest induced forest has size between $\alpha(G)$ (since an independent set is a forest) and $2\alpha(G)$ (since forests are bipartite). The result then follows from Theorem 1.

For Max Induced Tree, we repeat the same argument, after adding a universal vertex connected to everything to the instances of Max Independent Set of Theorem 1. ◀

▶ **Theorem 9.** *Under ETH, for any $\varepsilon > 0$ and $r \leqslant n^{1-\varepsilon}$, an r-approximation for* Max Induced Path *cannot take time* $2^{o(n/r)}$.

Proof. Let ϕ be any instance of 3-SAT. For any positive integer r, we build an instance graph G of Max Induced Path in the following way. For each clause C_i ($i \in [m]$) we add seven vertices $v_{i,1}^1, v_{i,2}^1, \ldots, v_{i,7}^1$ which form a clique C_i^1 and correspond to the seven partial assignments of the three literals of C_i satisfying the clause (if there is only two literals, then there is only three vertices in the clique). We add m vertices $v_1^1, v_2^1, \ldots, v_m^1$, and for all $i \in [2, m]$, we link v_i^1 to all the vertices of the cliques C_{i-1}^1 and all the vertices of the cliques C_i^1. Vertex v_1^1 is only linked to all the vertices of C_1^1. The graph defined at this point is called H_1. We make $r-1$ copies of H_1, denoted by $H_2, \ldots, H_r$. For each $j \in [2, r]$, the vertices of H_j are analogously denoted by $v_{i,1}^j, v_{i,2}^j, \ldots, v_{i,7}^j$ (vertices in the clique C_i^j corresponding to the clause C_i) and v_i^j. For each $j \in [2, r]$, we link vertex v_1^j to all the vertices of the clique C_m^{j-1}, and we add an edge between any two vertices corresponding to contradicting partial assignments, that is assignments attributing different truth values to the same variable (even if those vertices are in distinct H_is). We call such an edge a *contradicting edge*. The edges within the cliques C_i^j can be seen as contradicting edges, but we will not call them so.

If ϕ is satisfiable, let τ be a truth assignment. Let S be the set of the rm vertices in cliques C_i^j agreeing with τ (exactly one vertex per clique). The graph induced by $P = \bigcup_{1 \leqslant i \leqslant m, 1 \leqslant j \leqslant r} \{v_i^j\} \cup S$ is a path with $2rm$ vertices. Indeed, $\forall i \in [2, m], j \in [r]$, the degree of v_i^j in $G[P]$ is 2, since $|P \cap C_i^j| = 1$ and $|P \cap C_{i-1}^j| = 1$. And, $\forall j \in [2, r]$, the degree of v_1^j in $G[P]$ is 2, since $|P \cap C_1^j| = 1$ and $|P \cap C_m^{j-1}| = 1$. Vertex v_1^1 has only degree 1 (one

vertex in C_1^1) and is one endpoint of the path. The degree of the vertices of S in $G[P]$ is also 2, since by construction there is no contradicting edge in the graph induced by P. So, $\forall i \in [1, m-1], j \in [r]$, the only two neighbors of the unique vertex in $S \cap C_i^j$ are v_i^j and v_{i+1}^j. And, $\forall j \in [r-1]$, the only two neighbors of the unique vertex in $S \cap C_m^j$ are v_m^j and v_1^{j+1}. The degree in $G[P]$ of the unique vertex in $S \cap C_m^r$ is only 1; it is the other endpoint of the path.

For each $i \in [m]$, we call *column* R_i the union of the r cliques $C_i^1, C_i^2, \ldots, C_i^r$. Assume there is an induced path $G[Q]$ such that for some column R_i, $|Q \cap R_i| \geqslant 6$. So there are at least four vertices u_1, u_2, u_3, u_4 which are in $Q \cap R_i$ and are not one of the two endpoints of $G[Q]$. We set $U = \{u_1, u_2, u_3, u_4\}$. We say that two vertices in the cliques C_i^j *agree* if they represent non contradicting (or *compatible*) partial assignment. We observe that two vertices in the same column R_i agree iff they represent the same partial assignment. First, we can show that all the vertices in U have to agree with some other vertex. If one vertex $u \in U$ does not agree with any of the other vertices in U, then u has degree at least 3 in $G[Q]$ (there are three contradicting edges linking u to $U \setminus \{u\}$) which is not possible in a path. So, any vertex in U should agree with at least one vertex in $U \setminus \{u\}$. The first possibility is that there are two pairs (u, v) and (w, x) of vertices spanning U, such that the vertices agree within their pair but the two pairs do not agree. But that would create a cycle $uwvx$. The only remaining possibility is that all the vertices in U agree. As those vertices are in the same column, they even represent the *same* partial assignment.

Now, we will describe the path induced by Q by necessary conditions and derive that the formula is satisfiable. Let u_5 and u_6 be two vertices in $(Q \cap R_i) \setminus U$, and $W = U \cup \{u_5, u_6\}$. We observe that u_5 and u_6 should agree with the vertices of U, otherwise their degree in $G[Q]$ would be at least 4. So, all the vertices in W (pairwise) agree. The vertices of W are in pairwise distinct copies H_is. Hence, there are at least 4 copies denoted by $H_{a_1}, H_{a_2}, H_{a_3}, H_{a_4}$ which contain a vertex of W and *do not* contain an endpoint of $G[Q]$. Let $v_{i,h}^{a_1}$ be the unique vertex in $W \cap H_{a_1}$. By the previous remarks, $\forall p \in \{2, 3, 4\}$, $v_{i,h}^{a_p}$ is the unique vertex in $W \cap H_{a_p}$. For each $p \in [4]$, the two neighbors of $v_{i,h}^{a_p}$ in $G[Q]$ have to be $v_i^{a_p}$ and $v_{i+1}^{a_p}$. Vertex $v_{i,h}^{a_p}$ cannot incident to a contradicting edge, otherwise it would create a vertex of degree at least 4 in the path. At its turn, vertex $v_{i+1}^{a_p}$ has degree 2 in $G[Q]$, and its second neighbor has to be in the clique $C_{i+1}^{a_p}$ (if its second neighbor was also in $C_i^{a_p}$, it would form a triangle). Let $w_{p,i+1}$ be the unique vertex in $C_{i+1}^{a_p} \cap Q$. By the same arguments as before, $w_{1,i+1}$, $w_{2,i+1}$, $w_{3,i+1}$, and $w_{4,i+1}$ should all agree. This way we can extend the four fragments of paths to column R_{i+1} up to R_m. Symmetrically, we can extend the fragments of paths to column R_{i-1} to R_1. Now, if we just consider the path induced by $Q \cup H_{a_1}$, it goes through consistent partial assignments for each clause of the instance. The global assignment, built from all those partial assignments, satisfies all the clauses. So, the contrapositive is, if ϕ is not satisfiable, then for all $i \in [m]$, $|R_i \cap Q| < 6$. This implies $|Q| < 10m$.

The number of vertices of G is $8rm$. Recall that, under ETH [16], 3-SAT is not solvable in $2^{o(m)}$. Thus, under ETH, any r-approximation for Max Induced Path cannot take time $2^{o(n/r)}$. ◀

6 Min ATSP and Grundy Coloring

6.1 Min ATSP

In this section we deal with two problems for which the best known hardness of approximation bounds are small constants [18, 19], but no constant-factor approximation is known. We thus only present some algorithmic results.

For Min ATSP, the version of the TSP where we have the triangle inequality but distances may be asymmetric, the best known approximation algorithm has ratio $O(\log n/\log\log n)$ [1]. Here, we show that a classical, simpler $\log n$-approximation [14] can be adapted into an approximation scheme matching its performance in polynomial time. Whether the same can be done for the more recent, improved, algorithm remains as an interesting question.

▶ **Theorem 10.** *For any $r \leqslant n$,* Min ATSP *is $\log r$-approximable in time $O^*(2^{n/r})$.*

Proof. We roughly recall the $\log n$-approximation of Min ATSP detailed in [14]. The idea is to solve the problem of finding a (vertex-)disjoint union of circuits spanning the graph with minimum weight. This can be expressed as a linear program and therefore it can be solved in polynomial time. Let the circuits be $C_1, C_2, \ldots C_h$. We observe that the total length of the circuits is bounded by opt the optimum value for Min ATSP. We choose arbitrarily a vertex v_i in each C_i and recurse on the graph induced by $\{v_1, v_2, \ldots, v_h\}$. By the triangle inequality, we can combine a solution of Min ATSP in $G[\{v_1, v_2, \ldots, v_h\}]$ to the circuits C_is, and get a solution whose value is bounded by the sum of the lengths of the C_is plus the value of the solution for $G[\{v_1, v_2, \ldots, v_h\}]$, which would be 2opt if we solve $G[\{v_1, v_2, \ldots, v_h\}]$ to the optimum. In general, the depth of recursion is a bound on the ratio (see [14]). At each recursion step, the number of vertices in the remaining graph is at least divided by two. So, after at most $\log n$ recursions the algorithm terminates, hence the ratio.

Now, we can afford some superpolynomial computations. After $\log r$ recursions the number of vertices in the remaining graph is no more than $n/2^{\log r} = n/r$. We solve optimally this instance by dynamic programming in time $O^*(2^{n/r})$. The solution that we output has length smaller than $\log r \cdot \text{opt}$. ◀

6.2 Grundy Coloring

Max Grundy Coloring is the problem of ordering the vertices of a graph so that a greedy first-fit coloring applied on that order would use as many colors as possible. Unless NP⊆RP, Max Grundy Coloring admits no PTAS [19], but it is unknown if it can be $o(n)$-approximated.

Observe that, since this is not a subgraph problem, it is not *a priori* obvious that the baseline trade-off performance of Theorem 2 can be achieved. However, we give a simple trade-off scheme that does exactly that by reducing the ordering problem to that of finding an appropriate "witness", which is a set of vertices.

▶ **Theorem 11.** *For any $r > 1$,* Max Grundy Coloring *can be r-approximated in time $O^*(c^{n \log r/r})$, for some constant c.*

Proof. Let $G = (V, E)$ be any instance of Max Grundy Coloring, and r any real value. Here, we call *minimal witness* of G achieving color k, an induced subgraph W of G whose grundy number is k, such that all the induced subgraphs of W different from W have strictly smaller grundy numbers.

Let k be the grundy number of G and W be a minimal witness. Let $C_1 \uplus C_2 \uplus \ldots \uplus C_k$ be a partition of $V(W)$ corresponding to the color classes in an optimal coloring. Let $A_1, A_2, \ldots, A_{\lfloor k/r \rfloor}$ be the $\lfloor k/r \rfloor$ smallest (in terms of number of vertices) color classes among the C_is. Let $S = A_1 \uplus A_2 \uplus \ldots \uplus A_{\lfloor k/r \rfloor}$. Obviously $|V(W)| \leqslant n$, so $|S| \leqslant n/r$.

The algorithm exhausts all the subset of n/r vertices. For each subset of vertices, we run the exact algorithm running in time $O^*(2.246^n)$ on the corresponding induced subgraph. Thus, the algorithm takes time $O^*(2^{n \log r/r} 2.246^{n/r})$. As $|S| \leqslant n/r$, the algorithm considers at some point S or a superset of S. We just have to show that the optimal grundy coloring

of S is an r-approximation. Let us re-index the A_js by increasing values of their index in the C_is, say $B_1, B_2, \ldots, B_{\lfloor k/r \rfloor}$. Then for each $i \in [1, \lfloor k/r \rfloor]$, we can color B_i with color i and achieve color $\lfloor k/r \rfloor$. ◀

7 Set Cover

In this section we focus on the classical Min Set Cover problem, on inputs with n elements and m sets. In terms of n, a $\log r$-approximation is known in time roughly $2^{n/r}$. Moshkovitz [21] gave a reduction from N-variable 3-SAT which, for any $\alpha < 1$ produces instances with universe size $n = N^{O(1/\alpha)}$ and gap $(1-\alpha)\ln n$. Setting $\alpha = \ln(n/r)/\ln n$ translates this result to the terminology of our paper, and shows a running time lower bound of $2^{(n/r)^c}$, for some $c > 0$. Thus, even though the picture for this problem is not as clear as for, say Max Independent Set, it appears likely that the known trade-off scheme is optimal.

We consider here the complexity of the problem as a function of m. This is a well-motivated case, since for many applications m is much smaller than n [23]. Eventually, we would like to investigate whether the known r-approximation in time $2^{m/r}$ can be improved. Though we do not resolve this question, we show that the approximability status of this problem is somewhat unusual.

In polynomial time, the best known approximation algorithm has a guarantee of $\sqrt{m}$ [23]. We first observe that the simple argument of this algorithm can be extended to quasi-polynomial time.

▶ **Theorem 12.** *For any $\delta > 0$ there is an m^δ-approximation algorithm for* Min Set Cover *running in time $O^*(c^{(\log n)^{(1-\delta)/\delta}})$.*

Proof. The argument is similar to that of [23]. We distinguish two cases: if $m^\delta > \ln n$, then we can run the greedy polynomial time algorithm and return a solution with ratio better than m^δ. So assume that $m^\delta < \ln n$.

Now, run the r-approximation of [8], setting $r = m^\delta$. The running time is (roughly) $2^{m/r} = 2^{m^{1-\delta}}$. The result follows since $m < (\ln n)^{1/\delta}$. ◀

The above result is somewhat curious, since it implies that in quasi-polynomial time one can obtain an approximation ratio better than that of the best known polynomial-time algorithm. This leaves open two possibilities: either $\sqrt{m}$ is not in fact the optimal ratio in polynomial time, or there is a jump in the approximability of Min Set Cover from polynomial to quasi-polynomial time. We remark that, though this is rare, there is in fact another problem which displays exactly this behavior: for Graph Pricing the best polynomial-time ratio is $\sqrt{n}$, while n^δ can be achieved in time $O^*(c^{(\log m)^{(1-\delta)/\delta}})$ [6].

We do not settle this question, but observe that a combination of known reductions for Min Set Cover, the ETH and the Projection Games Conjecture of [21] imply that the optimal ratio in polynomial time is m^c for some $c > 0$. Thus, Min Set Cover is indeed likely to behave in a way similar to Graph Pricing. For Theorem 13 we essentially reuse the combination of reductions used in [7] to obtain FPT inapproximability results for Min Set Cover.

▶ **Theorem 13.** *Assume the ETH and the PGC. Then, there exists a $c > 0$ such that there is no m^c-approximation for* Min Set Cover *running in polynomial time.*

Proof. As mentioned, the proof reuses the reduction of [7], which in turn relies on the ETH, the PGC and classical reductions for Min Set Cover. To keep the presentation as short

and self-contained as possible we simply recall Theorem 5 of [7], without giving a detailed proof (or a definition of the PGC).

▶ **Theorem 14.** *[7] If the Projection Games Conjecture holds, for any $r > 1$ there exists a reduction from* 3-SAT *of size N to* Min Set Cover *with the following properties:*

- *YES instances produce* Min Set Cover *instances where the optimal cover has size β, NO instances produce* Min Set Cover *instances where the optimal cover has size at least $r\beta$.*
- *The size n of the universe is $2^{O(r)}\text{poly}(N, r)$.*
- *The number of sets m is $\text{poly}(N) \cdot \text{poly}(r)$.*
- *The reduction runs in time polynomial in n, m.*

Using the above reduction, we can conclude that there exists *some* constant c such that m^c-approximation for Min Set Cover is impossible in polynomial time, under the ETH. The constant c depends on the hidden exponents of the polynomials of the above reduction. The way to do this is to set r to be some polynomial of N, say $r = \sqrt{N}$. Then, the reduction runs in time sub-exponential in N (roughly $2^{\sqrt{N}}$) and produces a gap that is polynomially related to m. If in polynomial time we could r-approximate the new instance, this would give a sub-exponential time algorithm for 3-SAT. ◀

References

1. Arash Asadpour, Michel X. Goemans, Aleksander Madry, Shayan Oveis Gharan, and Amin Saberi. An $O(\log n/\log\log n)$-approximation algorithm for the asymmetric traveling salesman problem. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'10*, pages 379–389. SIAM, 2010. doi:10.1137/1.9781611973075.32.
2. Nicolas Bourgeois, Federico Della Croce, Bruno Escoffier, and Vangelis Th. Paschos. Fast algorithms for min independent dominating set. *Discrete Applied Mathematics*, 161(4-5):558–572, 2013. doi:10.1016/j.dam.2012.01.003.
3. Nicolas Bourgeois, Bruno Escoffier, and Vangelis Th. Paschos. Approximation of min coloring by moderately exponential algorithms. *Information Processing Letters*, 109(16):950–954, 2009. doi:10.1016/j.ipl.2009.05.002.
4. Nicolas Bourgeois, Bruno Escoffier, and Vangelis Th. Paschos. Efficient approximation of min set cover by moderately exponential algorithms. *Theoretical Computer Science*, 410(21-23):2184–2195, 2009. doi:10.1016/j.tcs.2009.02.007.
5. Nicolas Bourgeois, Bruno Escoffier, and Vangelis Th. Paschos. Approximation of max independent set, min vertex cover and related problems by moderately exponential algorithms. *Discrete Applied Mathematics*, 159(17):1954–1970, 2011. doi:10.1016/j.dam.2011.07.009.
6. Parinya Chalermsook, Bundit Laekhanukit, and Danupon Nanongkai. Independent set, induced matching, and pricing: Connections and tight (subexponential time) approximation hardnesses. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS'13*, pages 370–379, 2013. doi:10.1109/FOCS.2013.47.
7. Rajesh Hemant Chitnis, MohammadTaghi Hajiaghayi, and Guy Kortsarz. Fixed-parameter and approximation algorithms: A new look. In G. Gutin and S. Szeider, editors, *Parameterized and Exact Computation – 8th International Symposium, IPEC'13, Revised Selected Papers*, volume 8246 of *Lecture Notes in Computer Science*, pages 110–122. Springer, 2013. doi:10.1007/978-3-319-03898-8_11.
8. Marek Cygan, Lukasz Kowalik, and Mateusz Wykurz. Exponential-time approximation of weighted set cover. *Information Processing Letters*, 109(16):957–961, 2009. doi:10.1016/j.ipl.2009.05.003.

9 Marek Cygan and Marcin Pilipczuk. Exact and approximate bandwidth. *Theoretical Computer Science*, 411(40-42):3701–3713, 2010. doi:10.1016/j.tcs.2010.06.018.

10 Marek Cygan, Marcin Pilipczuk, and Jakub Onufry Wojtaszczyk. Capacitated domination faster than $O(2^n)$. *Information Processing Letters*, 111(23-24):1099–1103, 2011. doi:10.1016/j.ipl.2011.09.004.

11 Irit Dinur. The PCP theorem by gap amplification. *Journal of the ACM*, 54(3), 2007. Article 12. doi:10.1145/1236457.1236459.

12 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In D. B. Shmoys, editor, *Symposium on Theory of Computing, STOC'14*, pages 624–633. ACM, 2014. URL: http://dl.acm.org/citation.cfm?id=2591796, doi:10.1145/2591796.2591884.

13 Rodney G. Downey, Michael R. Fellows, Catherine McCartin, and Frances A. Rosamond. Parameterized approximation of dominating set problems. *Information Processing Letters*, 109(1):68–70, 2008. doi:10.1016/j.ipl.2008.09.017.

14 Alan M. Frieze, Giulia Galbiati, and Francesco Maffioli. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks*, 12(1):23–39, 1982. doi:10.1002/net.3230120103.

15 Magnús M. Halldórsson. Approximating the minimum maximal independence number. *Information Processing Letters*, 46(4):169–172, 1993. doi:10.1016/0020-0190(93)90022-2.

16 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computers and System Sciences*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

17 Viggo Kann. Strong lower bounds on the approximability of some NPO pb-complete maximization problems. In J. Wiedermann and P. Hájek, editors, *Mathematical Foundations of Computer Science 1995, 20th International Symposium, MFCS'95*, volume 969 of *Lecture Notes in Computer Science*, pages 227–236. Springer, 1995. doi:10.1007/3-540-60246-1_129.

18 Marek Karpinski, Michael Lampis, and Richard Schmied. New inapproximability bounds for TSP. In L. Cai, S.-W. Cheng, and T. W. Lam, editors, *Algorithms and Computation – 24th International Symposium, ISAAC'13*, volume 8283 of *Lecture Notes in Computer Science*, pages 568–578. Springer, 2013. doi:10.1007/978-3-642-45030-3_53.

19 Guy Kortsarz. A lower bound for approximating grundy numbering. *Discrete Mathematics & Theoretical Computer Science*, 9(1), 2007.

20 Carsten Lund and Mihalis Yannakakis. The approximation of maximum subgraph problems. In A. Lingas, R. G. Karlsson, and S. Carlsson, editors, *Automata, Languages and Programming, 20nd International Colloquium, ICALP'93*, volume 700 of *Lecture Notes in Computer Science*, pages 40–51. Springer, 1993. doi:10.1007/3-540-56939-1_60.

21 Dana Moshkovitz. The projection games conjecture and the NP-hardness of ln n-approximating set-cover. In A. Gupta, K. Jansen, J.D.P Rolim, and R. Servedio, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques – 15th International Workshop, APPROX'12, and 16th International Workshop, RANDOM'12*, volume 7408 of *Lecture Notes in Computer Science*, pages 276–287. Springer, 2012. doi:10.1007/978-3-642-32512-0_24.

22 Dana Moshkovitz and Ran Raz. Two-query PCP with subconstant error. *Journal of the ACM*, 57(5), 2010. Article 29. doi:10.1145/1754399.1754402.

23 Jelani Nelson. A note on set cover inapproximability independent of universe size. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(105), 2007.

A The problems handled in the paper

Max Independent Set: Given a graph $G = (V, E)$, Max Independent Set consists of finding a set $S \subseteq V$ of maximum size such that for any $(u, v) \in S \times S$, $(u, v) \notin E$.

Min Set Cover: Given a ground set C of cardinality n and a system $\mathcal{S} = \{S_1, \ldots, S_m\} \subset 2^C$, Min Set Cover consists of determining a minimum size subsystem $\mathcal{S}'$ such that $\cup_{S \in \mathcal{S}'} S = C$.

Min Independent Dominating Set: Given a graph $G = (V, E)$, Min Independent Dominating Set consists of finding the smallest independent set of G that is maximal for inclusion.

Max Minimal Vertex Cover: Given a graph $G = (V, E)$, Max Minimal Vertex Cover consists of finding the largest vertex cover of G that is minimal for exclusion.

Min ATSP: This a version of the TSP where we have the triangle inequality but the distance matrix may be asymmetric.

Max Induced Path, Max Induced Tree, Max Induced Forest: Given a graph $G = (V, E)$, we are looking for the largest set of vertices inducing a graph of the respective type.

Max Grundy Coloring: Given a graph $G = (V, E)$, Max Grundy Coloring is the problem of ordering the vertices of a graph so that a greedy first-fit coloring applied on that order would use as many colors as possible.

External Memory Three-Sided Range Reporting and Top-k Queries with Sublogarithmic Updates*

Gerth Stølting Brodal

MADALGO, Department of Computer Science, Aarhus University, Denmark
gerth@cs.au.dk

Abstract

An external memory data structure is presented for maintaining a dynamic set of N two-dimensional points under the insertion and deletion of points, and supporting unsorted 3-sided range reporting queries and top-k queries, where top-k queries report the k points with highest y-value within a given x-range. For any constant $0 < \varepsilon \leq \frac{1}{2}$, a data structure is constructed that supports updates in amortized $O(\frac{1}{\varepsilon B^{1-\varepsilon}} \log_B N)$ IOs and queries in amortized $O(\frac{1}{\varepsilon} \log_B N + K/B)$ IOs, where B is the external memory block size, and K is the size of the output to the query (for top-k queries K is the minimum of k and the number of points in the query interval). The data structure uses linear space. The update bound is a significant factor $B^{1-\varepsilon}$ improvement over the previous best update bounds for these two query problems, while staying within the same query and space bounds.

1998 ACM Subject Classification E.1 Data Structures

Keywords and phrases External memory, priority search tree, 3-sided range reporting, top-k queries

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.23

1 Introduction

In this paper we consider the problem of maintaining a dynamic set of N two-dimensional points from $\mathbb{R}^2$ in external memory, where the set of points can be updated by the insertion and deletion of points, and where two types of queries are supported: unsorted 3-sided range reporting queries and top-k queries. More precisely, we consider how to support the following four operations in external memory (see Figure 1):

Insert(p): Inserts a new point $p \in \mathbb{R}^2$ into the set S of points. If p was already in S, the old copy of p is replaced by the new copy of p (this case is relevant if points are allowed to carry additional information).

Delete(p): Deletes a point $p \in \mathbb{R}^2$ from the current set S of points. The set remains unchanged if p is not in the set.

Report(x_1, x_2, y): Reports all points contained in $S \cap [x_1, x_2] \times [y, \infty]$.

Top(x_1, x_2, k): Report k points contained in $S \cap [x_1, x_2] \times [-\infty, \infty]$ with highest y-value.

1.1 Previous Work

McCreight introduced the priority search tree [14] (for internal memory). The classic result is that priority search trees support updates in $O(\log N)$ time and 3-sided range reporting

* Work supported by the Danish National Research Foundation grant DNRF84 through the Center for Massive Data Algorithmics (MADALGO).

© Gerth Stølting Brodal;
licensed under Creative Commons License CC-BY

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 23; pp. 23:1–23:14

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

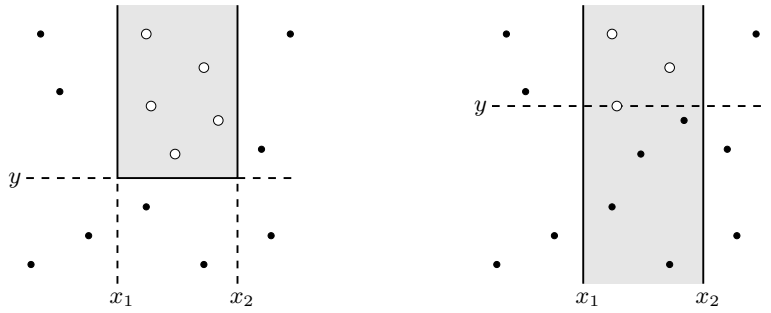


Figure 1 3-sided range reporting queries (left) and top-k queries (right). The reported points are the white points and $k = 3$.

queries in $O(\log N + K)$ time, where K is the number of points reported. Priority search trees are essentially just balanced heap-ordered binary trees where the root stores the point with minimum y-value and the remaining points are distributed among the left and right children so that all points in the left subtree have smaller x-value than points in the right subtree. Frederickson [10] presented an algorithm selecting the k smallest elements from a binary heap in time $O(k)$, which can be applied quite directly to a priority search tree to support top-k queries in $O(\log N + K)$ time.

Icking et al. [12] initiated the study of adapting priority search trees to external memory. Their structure uses linear space, i.e. $O(N/B)$ blocks, and supports 3-sided range reporting queries using $O(\log_2 N + K/B)$ IOs, where B is the external memory block size. Other early linear space solutions were given in [6] and [13] supporting queries with $O(\log_B N + K)$ and $O(\log_B N + K/B + \log_2 B)$ IOs, respectively. Ramaswamy and Subramanian in [18] presented a data structure with optimal query time but using suboptimal space, and in [20] they presented a data structure achieving optimal space but suboptimal queries (see Table 1). The best previous dynamic bounds are obtained by the external memory priority search tree by Arge et al. [4], which supports queries using $O(\log_B N + K/B)$ IOs and updates using $O(\log_B N)$ IOs, using linear space. The space and query bounds of [4] are optimal. External memory top-k queries were studied in [1, 19, 21]. Tao in [21] presented a data structure achieving bounds matching those of the external memory priority search tree of Arge et al. [4], updates being amortized. See Table 1 for an overview of previous results.

We improve the update bounds of both [4] and [21] by a factor $\varepsilon B^{1-\varepsilon}$ by adopting ideas of the buffer trees of Arge [3] to the external memory priority search tree [4].

1D Dictionaries

The classic B-tree of Bayer and McCreight [5] is the external memory counterpart of binary search trees for storing a set of one-dimensional points. A B-tree supports updates and membership/predecessor searches in $O(\log_B N)$ IOs and 1D range reporting queries in $O(\log_B N + K/B)$ IOs, where K is the output size. The query bounds for B-trees are optimal for comparison based external memory data structures, but the update bounds are not.

Arge [3] introduced the buffer tree as a variant of B-trees supporting *batched* sequences of interleaved updates and queries, where a sequence of N operations can be performed using $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ IOs, where M is the internal memory size. The buffer tree can e.g. be used as an external memory priority queue and segment tree, and has applications to external memory graph problems and computational geometry problems. By adapting Arge's technique of buffering updates (insertions and deletions) to a B-tree of degree B^ε, where

Table 1 Previous external-memory 3-sided range reporting and top-k data structures. All query bounds are optimal except [20]. Amortized bounds are marked "†", and ε is a constant satisfying $1 > \varepsilon > 0$. All data structures require space $O(N/B)$, except [18] requiring space $O(\frac{N}{B} \log B \log\log B)$. $\mathcal{IL}^*(x)$ denotes the number of times $\log^*$ must be applied before the result becomes ≤ 2 [20].

Query	Reference	Update	Query	Construction
3-sided	[18]	$O(\log N \cdot \log B)^\dagger$	$O(\log_B N + K/B)$	
	[20]	$O(\log_B N + (\log_B N)^2/B)^\dagger$	$O(\log_B N + K/B + \mathcal{IL}^*(B))$	
	[4]	$O(\log_B N)$	$O(\log_B N + K/B)$	
	New	$O(\frac{1}{\varepsilon B^{1-\varepsilon}} \log_B N)^\dagger$	$O(\frac{1}{\varepsilon} \log_B N + K/B)^\dagger$	$O(\mathrm{Sort}(N))$
Top-k	[1]	(static)	$O(\log_B N + K/B)$	
	[19]	$O(\log_B^2 N)^\dagger$	$O(\log_B N + K/B)$	$O(\mathrm{Sort}(N))$
	[21]	$O(\log_B N)^\dagger$	$O(\log_B N + K/B)$	
	New	$O(\frac{1}{\varepsilon B^{1-\varepsilon}} \log_B N)^\dagger$	$O(\frac{1}{\varepsilon} \log_B N + K/B)^\dagger$	$O(\mathrm{Sort}(N))$

$0 < \varepsilon < 1$ is a constant, and where each node stores a buffer of $O(B)$ buffered updates, one can achieve updates using amortized $O(\frac{1}{\varepsilon B^{1-\varepsilon}} \log_B N)$ IOs and member queries in $O(\frac{1}{\varepsilon} \log_B N)$ IOs.

Brodal and Fagerberg [8] studied the trade-offs between the IO bounds for comparison based updates and membership queries in external memory. They proved the optimality of B-trees with buffers when the amortized update cost is in the range $1/\log^3 N$ to $\log_{B+1} \frac{N}{M}$.

Verbin and Zhang [23] and Iacono and Pătraşcu [11] consider trade-offs between updates and membership queries when hashing is allowed, i.e. elements are not indivisible. In [11] it is proved that updates can be supported in $O(\frac{\lambda}{B})$ IOs and queries in $O(\log_\lambda N)$ IOs, for $\lambda \geq \max\{\log\log N, \log_{M/B}(N/B)\}$. Compared to the comparison based bounds, this essentially removes a factor $\log_B N$ from the update bounds.

Related Top-k Queries

In the RAM model Brodal et al. [9] presented a linear space static data structure provided for the case where x-values were $1, 2, \ldots, N$, i.e. input is an array of y-values. The data structure supports sorted top-k queries in $O(k)$ time, i.e. reports the top K in decreasing y-order one point at a time.

Afshani et al. [1] studied the problem in external memory and proved a trade-off between space and query time for sorted top k queries, and proved that data structures with query time $\log^{O(1)} N + O(cK/B)$ requires space $\Omega\left(\frac{N}{B} \frac{\frac{1}{c}\log_M \frac{N}{B}}{\log(\frac{1}{c}\log_M \frac{N}{B})}\right)$ blocks. It follows that for linear space top-k data structures it is crucial that we focus on unsorted range queries.

Rahul et al. [16] and Rahul and Tao [17] consider the static top-k problem for 2D points with associated real weights where queries report the top-k points with respect to weight contained in an axis-parallel rectangle. Rahul and Tao [17] achieve query time $O(\log_B N + K/B)$ using space $O(\frac{N}{B} \frac{\log N \cdot (\log\log B)^2}{\log\log_B N})$, $O(\frac{N}{B} \frac{\log N}{\log\log_B N})$, and $O(N/B)$ for supporting 4-sided, 3-sided and 2-sided top-k queries respectively.

1.2 Model of Computation

The results of this paper are in the external memory model of Aggarwal and Vitter [2] consisting of a two-level memory hierarchy with an unbounded external memory and an internal memory of size M. An IO transfers $B \leq M/2$ consecutive records between internal and external memory. Computation can only be performed on records in internal memory.

The basic results in the model are that the scanning and sorting an array require $\Theta(\text{Scan}(N))$ and $\Theta(\text{Sort}(N))$ IOs, where $\text{Scan}(N) = \frac{N}{B}$ and $\text{Sort}(N) = \frac{N}{B}\log_{M/B}\frac{N}{B}$respectively [2].

In this paper we assume that the only operation on points is the comparison of coordinates. For the sake of simplicity in the following we assume that all points have distinct x- and y-values. If this is not the case, we can extend the x-ordering to the lexicographical order $\prec_x$ where $(x_1, y_1) \prec_x (x_2, y_2)$ if and only if $x_1 < x_2$, or $x_1 = x_2$ and $y_1 < y_2$, and similarly for the comparison of y-values.

1.3 Our Results

This paper provides the first external memory data structure for 3-sided range reporting queries and top-k queries with amortized sublogarithmic updates.

▶ **Theorem 1.** *For any constant ε, $0 < \varepsilon \leq \frac{1}{2}$, there exists an external memory data structure supporting the insertion and deletion of points in amortized $O(\frac{1}{\varepsilon B^{1-\varepsilon}}\log_B N)$ IOs and 3-sided range reporting queries and top-k queries in amortized $O(\frac{1}{\varepsilon}\log_B N + K/B)$ IOs, where N is the current number of points and K is the size of the query output. Given an x-sorted set of N points, the structure can be constructed with amortized $O(N/B)$ IOs. The space usage of the data structure is $O(N/B)$ blocks.*

To achieve the results in Theorem 1 we combine the external memory priority search tree of Arge et al. [4] with the idea of buffered updates from the buffer tree of Arge [3]. Buffered insertions and deletions move downwards in the priority search tree in batches whereas points with large y-values move upwards in the tree in batches. We reuse the dynamic substructure of [4] for storing $O(B^2)$ points at each node of the priority search tree, except that we reduce its capacity to $B^{1+\varepsilon}$ to achieve amortized $o(1)$ IOs per update. The major technical novelty in this paper lays in the top-k query (Section 7) that makes essential use of Frederickson's binary heap selection algorithm [10] to select an approximate y-value, that allows us to reduce top-k queries to 3-sided range reporting queries combined with standard selection [7].

One might wonder if the bounds of Theorem 1 are the best possible. Both 3-sided range reporting queries and top-k queries can be used to implement a dynamic 1D dictionary with membership queries by storing a value $x \in \mathbb{R}$ as the 2D point $(x, x) \in \mathbb{R}^2$. A dictionary membership query for x can then be answered by the 3-sided query $[x_1, x_2] \times [-\infty, \infty]$ or a top-1 query for $[x, x]$. If our queries had been worst-case instead of amortized, it would follow from [8] that our data structure achieves an optimal trade-off between the worst-case query time and amortized update time for the range where the update cost is between $1/\log^3 N$ to $\log_{B+1}\frac{N}{M}$. Unfortunately, our query bounds are amortized and the argument does not apply. Our query bounds are inherently amortized and it remains an open problem if the bounds in Theorem 1 can be obtained in the worst case. Throughout the paper we assume the amortized analysis framework of Tarjan [22] is applied in the analysis.

Outline of Paper

In Section 2 we describe our data structure for point sets of size $O(B^{1+\varepsilon})$. In Section 3 we define our general data structure. In Section 4 we describe to how support updates, in Section 5 the application of global rebuilding, and in Sections 6 and Section 7 how to support 3-sided range reporting and top-k queries, respectively.

2 $O(B^{1+\varepsilon})$ Structure

In this section we describe a data structure for storing a set of $O(B^{1+\varepsilon})$ points, for a constant $0 \leq \varepsilon \leq \frac{1}{2}$, that supports 3-sided range reporting queries using $O(1 + K/B)$ IOs and the batched insertion and deletion of $s \leq B$ points using amortized $O(1 + s/B^{1-\varepsilon})$ IOs. The structure is very much identical to the external memory priority search structure of Arge et al. [4, Section 3.1] for handling $O(B^2)$ points. The essential difference is that we reduce the capacity of the data structure to obtain amortized $o(1)$ IOs per update, and that we augment the data structure with a sampling operation required by our top-k queries. A sampling intuitively selects the y-value of approximately every Bth point with respect to y-value within a query range $[x_1, x_2] \times [-\infty, \infty]$ and takes $O(1)$ IOs.

In the following we describe how to support the below operations within the bounds stated in Theorem 2.

Insert$(p_1, \ldots, p_s)$ Inserts the points $p_1, \ldots, p_s$ into the structure, where $1 \leq s \leq B$.

Deletes$(p_1, \ldots, p_s)$ Deletes the points $p_1, \ldots, p_s$ from the structure, where $1 \leq s \leq B$.

Report(x_1, x_2, y) Reports all points within the query range $[x_1, x_2] \times [y, \infty]$.

Sample(x_1, x_2) Returns a decreasing sequence of $O(B^\varepsilon)$ y-values $y_1 \geq y_2 \geq \cdots$ such that for each y_i there are between iB and $iB + \alpha B$ points in the range $[x_1, x_2] \times [y_i, \infty]$, for some constant $\alpha \geq 1$. Note that this implies that in the range $[x_1, x_2] \times [y_{i+1}, y_i[$ there are between 0 and $(1 + \alpha)B$ points.

▶ **Theorem 2.** *There exists a data structure for storing $O(B^{1+\varepsilon})$ points, $0 \leq \varepsilon \leq \frac{1}{2}$, where the insertion and deletion of s points requires amortized $O(1 + s/B^{1-\varepsilon})$ IOs. Report queries use $O(1 + K/B)$ IOs, where K is the number of points returned, and Sample queries use $O(1)$ IOs. Given an x-sorted set of N points, the structure can be constructed with $O(N/B)$ IOs. The space usage is linear.*

Data Structure

Our data structure $\mathcal{C}$ consists of four parts. A static data structure $\mathcal{L}$ storing $O(B^{1+\varepsilon})$ points; two buffers $\mathcal{I}$ and $\mathcal{D}$ of delayed insertions and deletions, respectively, each containing at most B points; and a set $\mathcal{S} \subseteq \mathcal{L}$ of $O(B)$ sampled points. A point can appear at most once in $\mathcal{I}$ and $\mathcal{D}$, and at most in one of them. Initially all points are stored in $\mathcal{L}$, and $\mathcal{I}$ and $\mathcal{D}$ are empty.

Let L be the points in the $\mathcal{L}$ structure and let $\ell = \lceil |L|/B \rceil$. The data structure $\mathcal{L}$ consists of $2\ell - 1$ blocks. The points in L are first partitioned left-to-right with respect to x-value into blocks $b_1, \ldots, b_\ell$ each of size B, except possibly for the rightmost block b_ℓ just having size $\leq B$. Next we make a vertical sweep over the points in increasing y-order. Whenever the sweepline reaches a point in a block where the block together with an adjacent block contains exactly B points on or above the sweepline, we replace the two blocks by one block only containing these B points. Since each such block contains exactly the points on or above the sweepline for a subrange $b_i, \ldots, b_j$ of the initial blocks, we denote such a block $b_{i,j}$. The two previous blocks are stored in $\mathcal{L}$ but are no longer part of the vertical sweep. Since each fusion of adjacent blocks causes the sweepline to intersect one block less, it follows that at most $\ell - 1$ such blocks can be created. Figure 2 illustrates the constructed blocks, where each constructed block is illustrated by a horizontal line segment, and the points contained in the block are exactly all the points on or above the corresponding line segment. Finally, we have a "catalog" storing a reference to each of the $2\ell - 1$ blocks of $\mathcal{L}$. For a block b_i we store the minimum and maximum x-values of the points within the block. For blocks $b_{i,j}$

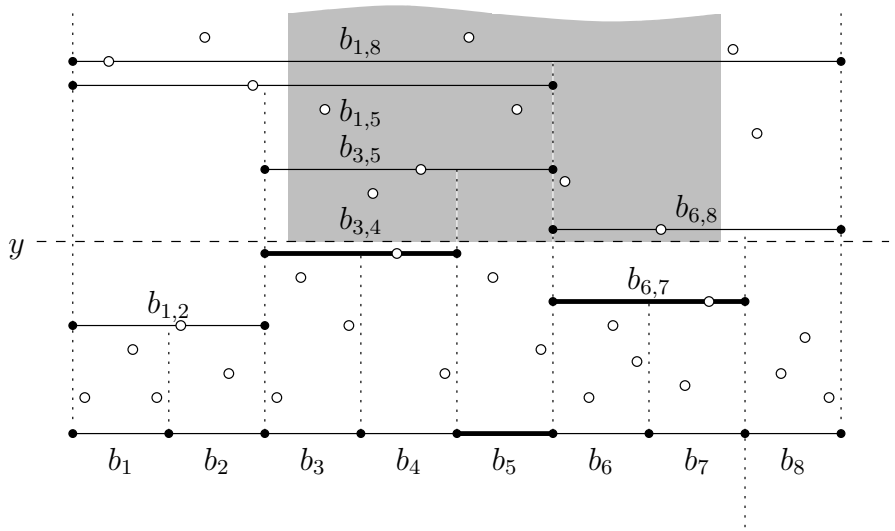


Figure 2 $O(B^{1+\varepsilon})$ structure for $B = 4$. White nodes are the points. Horizontal line segments with black endpoints illustrate the blocks stored. Each block stores the B points on and above the line segment.

we store the interval $[i, j]$ and the minimum y-value of a point in the block, i.e. the y-value where the sweep caused block $b_{i,j}$ to be created.

The set $\mathcal{S} \subseteq \mathcal{L}$ contains from each block $b_1, \ldots, b_\ell$ the points with the $\lceil i \cdot B^\varepsilon \rceil$-th highest y-value for all $1 \leq i \leq B^{1-\varepsilon}$. Since $\ell = O(B^\varepsilon)$, the total number of points in $\mathcal{S}$ is $O(B^\varepsilon \cdot B^{1-\varepsilon}) = O(B)$. The sets $\mathcal{S}$, $\mathcal{I}$, $\mathcal{D}$ and the catalog are stored in $O(1)$ blocks.

Updates

Whenever points are inserted or deleted we store the delayed updates in $\mathcal{I}$ or $\mathcal{D}$, respectively. Before adding a point p to $\mathcal{I}$ or $\mathcal{D}$ we remove any existing occurrence of p in $\mathcal{I}$ and $\mathcal{D}$, since the new update overrides all previous updates of p. Whenever $\mathcal{I}$ or $\mathcal{D}$ overflows, i.e. its size exceeds B, we apply the updates to the set of points in $\mathcal{L}$, and rebuild $\mathcal{L}$ for the updated point set. To rebuild $\mathcal{L}$, we extract the points L in $\mathcal{L}$ in increasing x-order from the blocks $b_1, \ldots, b_\ell$ in $O(\ell)$ IOs, and apply the $O(B)$ updates in $\mathcal{I}$ or $\mathcal{D}$ during the scan of the points to achieve the updated point set L'. We split L' into new blocks $b_1, \ldots, b_{\ell'}$ and perform the vertical sweep by holding in internal memory a priority queue storing for each adjacent pair of blocks the y-value where the blocks potentially should be fusioned. This allows the construction of each of the remaining blocks $b_{i,j}$ of $\mathcal{L}$ in $O(1)$ IOs per block. The reconstruction takes worst-case $O(\ell')$ IOs. Since $|L| = O(B^{1+\varepsilon})$ and the reconstruction of $\mathcal{L}$ whenever a buffer overflow occurs requires $O(|L|/B) = O(B^\varepsilon)$ IOs, the amortized cost of reconstructing $\mathcal{L}$ is $O(1/B^{1-\varepsilon})$ IOs per buffered update.

3-sided Reporting Queries

For a 3-sided range reporting query $Q = [x_1, x_2] \times [y, \infty]$, the t line segments immediately below the bottom segment of the query range Q correspond exactly to the blocks intersected by the sweep when it was at y, and the blocks contain a superset of the points contained in Q. In Figure 2 the grey area shows a 3-sided range reporting query $Q = [x_1, x_2] \times [y, \infty]$, where the relevant blocks are $b_{3,4}$, b_5 and $b_{6,7}$. By construction we know that at the sweepline two

consecutive blocks contain at least B points on or above the sweepline. Since the leftmost and rightmost of these blocks do not necessarily contain any points from Q, it follows that the output to the range query Q is at least $K \geq B\lfloor(t-2)/2\rfloor$. The relevant blocks can be found directly from the catalog using $O(1)$ IOs and the query is performed by scanning these t blocks, and reporting the points contained in Q. The total number of IOs becomes $O(1+t) = O(1+K/B)$.

Sampling Queries

To perform a sampling query for the range $[x_1, x_2]$ we only consider $\mathcal{L}$, i.e. we ignore the $O(B)$ buffered updates. We first identify the two blocks b_i and b_j spanning x_1 and x_2, respectively, by finding the predecessor of x_1 (successor of x_2) among the minimum (maximum) x-values stored in the catalog. The sampled points in $\mathcal{S}$ for the blocks $b_{i+1}, \ldots, b_{j-1}$ are extracted in decreasing y-order, and the $\lceil(s+1)\cdot B^{1-\varepsilon}\rceil$-th y-values are returned from this list for $s = 1, 2, \ldots$. Let $y_1 \geq y_2 \geq \cdots$ denote these returned y-values.

We now bound the number of points in $\mathcal{C}$ contained in the range $Q_s = [x_1, x_2] \times [y_s, \infty]$. By construction there are $\lceil(s+1)\cdot B^{1-\varepsilon}\rceil$ points with y-values $\geq y_s$ in $\mathcal{S}$ from points in $b_{i+1} \cup \cdots \cup b_{j-1}$. In each b_t there are at most $\lceil B^\varepsilon\rceil$ points vertically between each sampled point in $\mathcal{S}$. Assume there are n_t sampled points with y-values $\geq y_s$ in $\mathcal{S}$ from points in b_t, i.e. $n_{i+1} + \cdots + n_{j-1} = \lceil(s+1)\cdot B^{1-\varepsilon}\rceil$. The number of points in b_t with y-value $\geq y_s$ is at least $\lceil n_t B^\varepsilon\rceil$ and less than $\lceil(n_t+1)B^\varepsilon\rceil$, implying that the total number of points in $Q_s \cap (b_{i+1} \cup \cdots \cup b_{j-1})$ is at least $\sum_{t=i+1}^{j-1}\lceil n_t B^\varepsilon\rceil \geq B^\varepsilon \sum_{t=i+1}^{j-1} n_t = B^\varepsilon\lceil(s+1)\cdot B^{1-\varepsilon}\rceil \geq (s+1)B$ and at most $\sum_{t=i+1}^{j-1}(n_t+1)B^\varepsilon = (j-i-1)B^\varepsilon + B^\varepsilon\sum_{t=i+1}^{j-1} n_t = (j-i-1)B^\varepsilon + B^\varepsilon\lceil(s+1)\cdot B^{1-\varepsilon}\rceil \leq (j-i)B^\varepsilon + (s+1)B$. Since the buffered deletions in $\mathcal{D}$ at most cancel B points from $\mathcal{L}$ it follows that there are at least $(s+1)B - B = sB$ points in the range Q_s. Since there are most B buffered insertions in $\mathcal{I}$ and B points in each of the blocks b_i and b_j, it follows that Q_s contains at most $(j-i)B^\varepsilon + (s+1)B + 3B = sB + O(B)$ points, since $j - i = O(B^\varepsilon)$ and $\varepsilon \leq \frac{1}{2}$. It follows that the generated sample has the desired properties.

Since the query is answered by reading only the catalog and $\mathcal{S}$, the query only requires $O(1)$ IOs. Note that the returned y-values might be the y-values of deleted points by buffered deletions in $\mathcal{D}$.

3 The Data Structure

To achieve our main result, Theorem 1, we combine the external memory priority search tree of Arge et al. [4] with the idea of buffered updates from the buffer tree of Arge [3]. As in [4], we have at each node of the priority search tree an instance of the data structure of Section 2 to handle queries on the children efficiently. The major technical novelty lays in the top-k query (Section 7) that makes essential use of Frederickson's binary heap selection algorithm [10] and our samplings queries from Section 2.

Structure

The basic structure is a B-tree [5] T over the x-values of points, where the degree of each internal node is in the range $[\Delta/2, \Delta]$, where $\Delta = \lceil B^\varepsilon\rceil$, except for the root r that is allowed to have degree in the range $[2, \Delta]$. Each node v of T stores three buffers containing $O(B)$ points: a *point buffer* P_v, an *insertion buffer* I_v, and a *deletion buffer* D_v. The intuitive idea is that T together with the P_v sets form an external memory priority search tree, i.e. a point in P_v has larger y-value than all points in P_w for all descendants w of v, and that

the I_v and D_v sets are delayed insertions and deletions on the way down through T that we will handle recursively in batches when buffers overflow. A point $p \in I_v$ ($p \in D_v$) should eventually be inserted in (deleted from) one of the P_w buffers at a descendant w of v. Finally for each internal node v with children $c_1, \ldots, c_\delta$ we will have a data structure $\mathcal{C}_v$ storing $\cup_{i=1}^{\delta} P_{c_i}$, that is an instance of the data structure from Section 2. In a separate block at v we store for each child c_i the minimum y-value of a point in P_{c_i}, or $+\infty$ if P_{c_i} is empty. We assume that all information at the root is kept in internal memory, except for $\mathcal{C}_r$.

Invariants

For a node v, the buffers P_v, I_v and D_v are disjoint and all points have x-values in the x-range spanned by the subtree T_v rooted at v in T. All points in $I_v \cup D_v$ have y-value less than the points in P_v. In particular leaves have empty I_v and D_v buffers. If a point appears in a buffer at a node v and at a descendant w, the update at v is the most recent.

The sets stored at a node v must satisfy one of the below size invariants, guaranteeing that either P_v contains at least $B/2$ points, or all insertion and deletion buffers in T_v are empty and all points in T_v are stored in the point buffer P_v.

1. $B/2 \leq |P_v| \leq B$, $|D_v| \leq B/4$, and $|I_v| \leq B$, or
2. $|P_v| < B/2$, $I_v = D_v = \emptyset$, and $P_w = I_w = D_w = \emptyset$ for all descendants w of v in T.

4 Updates

Consider the insertion or deletion of a point $p = (p_x, p_y)$. First we remove any (outdated) occurrence of p from the root buffers P_r, I_r and D_r. If p_y is smaller than the smallest y-value in P_r then p is inserted into I_r or D_r, respectively. Finally, for an insertion where p_y is larger than or equal to the smallest y-value in P_r then p is inserted into P_r. If P_r overflows, i.e. $|P_r| = B + 1$, we move a point with smallest y-value from P_r to I_r.

During the update above, the I_r and D_r buffers might overflow, which we handle by the five steps described below: (*i*) handle overflowing deletion buffers, (*ii*) handle overflowing insertion buffers, (*iii*) split leaves with overflowing point buffers, (*iv*) recursively split nodes of degree $\Delta + 1$, and (*v*) recursively fill underflowing point buffers. For deletions only (*i*) and (*v*) are relevant, whereas for insertions (*ii*)–(*v*) are relevant.

(*i*) If a deletion buffer D_v overflows, i.e. $|D_v| > B/4$, then by the pigeonhole principle there must exist a child c where we can push a subset $U \subseteq D_v$ of $\lceil |D_v|/\Delta \rceil$ deletions down to. We first remove all points in U from D_v, I_c, D_c, P_c, and $\mathcal{C}_v$. Any point p in U with y-value larger than or equal to the minimum y-value in P_c is removed from U (since the deletion of p cannot cancel further updates). If v is a leaf, we are done. Otherwise, we add the remaining points in U to D_c, which might overflow and cause a recursive push of buffered deletions. In the worst-case, deletion buffers overflow all the way along a path from the root to a single leaf, each time causing at most $\lceil B/\Delta \rceil$ points to be pushed one level down. Updating a $\mathcal{C}_v$ buffer with $O(B/\Delta)$ updates takes amortized $O(1 + (B/\Delta)/B^{1-\varepsilon}) = O(1)$ IOs.

(*ii*) If an insertion buffer I_v overflows, i.e. $|I_v| > B$, then by the pigeonhole principle there must exist a child c where we can push a subset $U \subseteq I_v$ of $\lceil |I_v|/\Delta \rceil$ insertions down to. We first remove all points in U from I_v, I_c, D_c, P_c, and $\mathcal{C}_v$. Any point in U with y-value larger than or equal to the minimum y-value in P_c is inserted into P_c and $\mathcal{C}_v$ and removed from U (since the insertion cannot cancel further updates). If P_c overflows, i.e. $|P_c| > B$, we repeatedly move the points with smallest y-value from P_c to U until $|P_c| = B$. If c is a leaf all points in U are inserted into P_c (which might overflow), and U is now empty. Otherwise, we add the remaining points in U to I_c, which might overflow and cause

a recursive push of buffered insertions. As for deletions, in the worst-case insertion buffers overflow all the way along a path from the root to a single leaf, each time causing $O(B/\Delta)$ points to be pushed one level down. Updating a $\mathcal{C}_v$ buffer with $O(B/\Delta)$ updates takes amortized $O(1 + (B/\Delta)/B^{1-\varepsilon}) = O(1)$ IOs.

(*iii*) If the point buffer P_v at a leaf v overflows, i.e. $|P_v| > B$, we split the leaf v into two nodes v' and v'', and distribute evenly the points P_v among $P_{v'}$ and $P_{v''}$ using $O(1)$ IOs. Note that the insertion and deletion buffers of all the involved nodes are empty. The splitting might cause the parent to get degree $\Delta + 1$.

(*iv*) While some node v has degree $\Delta + 1$, split the node into two nodes v' and v'' and distribute P_v, I_v and D_v among the buffers at the nodes v' and v'' w.r.t. x-value. Finally construct $\mathcal{C}_{v'}$ and $\mathcal{C}_{v''}$ from the children point sets P_c. In the worst-case all nodes along a single leaf-to-root path will have to split, where the splitting of a single node costs $O(\Delta)$ IOs, due to reconstructing $\mathcal{C}$ structures.

(v) While some node v has an underflowing point buffer, i.e. $|P_v| < B/2$, we try to move the $B/2$ top points into P_v from v's children. If all subtrees below v do not store any points, we remove all points from D_v, and repeatedly move the point with maximum y-value from I_v to P_v until either $|P_v| = B$ or $I_v = \emptyset$. Otherwise, we scan the children's point buffers $P_{c_1}, \ldots, P_{c_\delta}$ using $O(\Delta)$ IOs to identify the $B/2$ points with largest y-value, where we only read the children with nonempty point buffers (information about empty point buffers at the children is stored at v, since we store the minimum y-value in each of the children's point buffer). These points X are then deleted from the children's P_{c_i} lists using $O(\Delta)$ IOs and from $\mathcal{C}_v$ using $O(B^\varepsilon) = O(\Delta)$ IOs. All points in $X \cap D_v$ are removed from X and D_v (since they cannot cancel further updates below v). For all points $p \in X \cap I_v$, the occurrence of p in X is removed and the more recent occurrence in I_v is moved to X. While the highest point in I_v has higher y-value than the lowest point in X, we swap these two values to satisfy the ordering among buffer points. Finally all remaining points in X are inserted into P_v using $O(1)$ IOs and into $\mathcal{C}_u$ using $O(B^\varepsilon) = O(\Delta)$ IOs, where u is the parent of v. The total cost for pulling these up to $B/2$ points one level up in T is $O(\Delta)$ IOs. It is crucial that we do the pulling up of points bottom-up, such that we always fill the lowest node in the tree, which will guarantee that children always have non-underflowing point buffers if possible. After having pulled points from the children, we need to check if any of the children's point buffers underflows and should be refilled.

Analysis

The tree T is rebalanced during updates by the splitting of leaves and internal nodes. We do not try to fusion nodes to handle deletions. Instead we apply global rebuilding whenever a linear number of updates have been performed (see Section 5). A leaf v will only be split into two leaves whenever its P_v buffer overflows, i.e. when $|P| > B$. It follows that the total number of leaves created during a total of N insertions can at most be $O(N/B)$, implying that at most $O(\frac{N}{\Delta B})$ internal nodes can be created by the recursive splitting of nodes. It follows that T has height $O(\log_\Delta \frac{N}{B}) = O(\frac{1}{\varepsilon} \log_B N)$.

For every $\Theta(B/\Delta)$ update, in (*i*) and (*ii*) amortized $O(1)$ IOs are spend on each the $O(\log_\Delta \frac{N}{B})$ levels of T, i.e. amortized $O(\frac{\Delta}{B} \log_\Delta \frac{N}{B}) = O(\frac{1}{\varepsilon B^{1-\varepsilon}} \log_B N)$ IOs per update. For a sequence of N updates, in (*iii*) at most $O(N/B)$ leaves are created requiring $O(1)$ IOs each and in (*iv*) at most $O(\frac{N}{B\Delta})$ non-leaf nodes are created. The creation of each non-leaf node costs amortized $O(\Delta)$ IOs, i.e. in total $O(N/B)$ IOs, and amortized $O(1/B)$ IO per update.

The analysis of (v) is more complicated, since the recursive filling can trigger cascaded recursive refillings. Every refilling of a node takes $O(\Delta)$ IOs and moves $\Theta(B)$ points one level

up in the tree's point buffers (some of these points can be eliminated from the data structure during this move). Since each point at most can move $O(\log_\Delta \frac{N}{B})$ levels up, the total number of IOs for the refillings during a sequence of N operations is amortized $O(\frac{N}{B}\Delta \log_\Delta \frac{N}{B})$ IOs, i.e. amortized $O(\frac{1}{\varepsilon B^{1-\varepsilon}} \log_B N)$ IOs per point. The preceding argument ignores two cases. The first case is that during the pull up of points some points from P_c and I_v swap rôles due to their relative y-values. But this does not change the accounting, since the number of points moved one level up does not change due to this change of rôle. The second case is when all children of a node all together have less than $B/2$ points, i.e. we do not move as many points up as promised. In this case we will move to v all points we find at the children of v, such that these children become empty and cannot be read again before new points have been pushed down to these nodes. We can now do a simple amortization argument: By double charging the IOs we previously have counted for pushing points to a child we can ensure that each node with non-empty point buffer always has saved an IO for being emptied. It follows that the above calculations remain valid.

5 Global Rebuilding

We adopt the technique of global rebuilding [15, Chapter 5] to guarantee that T is balanced. We partition the sequence of updates into epochs. If the data structure stores $\bar{N}$ points at the beginning of an epoch the next epoch starts after $\bar{N}/2$ updates have been performed. This ensures that during the epoch the current size satisfies $\frac{1}{2}\bar{N} \leq N \leq \frac{3}{2}\bar{N}$, and that T has height $O(\frac{1}{\varepsilon}\log_B \frac{3\bar{N}}{2}) = O(\frac{1}{\varepsilon}\log_B N)$.

At the beginning of an epoch we rebuild the structure from scratch by constructing a new empty structure and reinsert all the non-deleted points from the previous structure. We identify the points to insert in a top-down traversal of the T, always flushing the insertion and deletion buffers of a node v to its children and inserting all points of P_v into the new tree. The insertion and deletion buffers might temporarily have size $\omega(B)$. To be able to filter out deleted points etc., we maintain the buffers P_v, I_v, and D_v in lexicographically sorted order. Since level i (leaves being level 0) contains at most $\frac{3\bar{N}}{2B(\Delta/2)^i}$ nodes, i.e. stores $O(\frac{\bar{N}}{(\Delta/2)^i})$ points to be reported and buffered updates to be moved i levels down, the total cost of flushing all buffers is $O(\sum_{i=0}^{\infty}(i+1)\frac{\bar{N}}{B(\Delta/2)^i}) = O(\frac{\bar{N}}{B})$ IOs.

The $O(\bar{N})$ reinsertions into the new tree can be done in $O(\frac{\bar{N}}{\varepsilon B^{1-\varepsilon}}\log_B \bar{N})$ IOs. The $\bar{N}/2$ updates during an epoch are each charged a constant factor amortized overhead to cover the $O(\frac{\bar{N}}{\varepsilon B^{1-\varepsilon}}\log_B \bar{N})$ IO cost of rebuilding the structure at the end of the epoch.

6 3-sided Range Reporting Queries

Our implementation of 3-sided range reporting queries $Q = [x_1, x_2] \times [y, \infty]$ consists of three steps: Identify the nodes to *visit* for reporting points, push down buffered insertions and deletions between visited nodes, and finally return the points in the query range Q.

We recursively identify the nodes to visit, as the $O(\frac{1}{\varepsilon}\log_B N)$ nodes on the two root-to-leaf search paths in T for x_1 and x_2, and all nodes v between x_1 and x_2 where all points in P_v are in Q. We can check if we should visit a node w without reading the node, by comparing y with the minimum y-value in P_w that is stored at the parent of w. It follows that all points to be reported by Q are contained in the P_v and I_v buffers of visited nodes v or point buffers at the children of visited nodes, i.e. in $\mathcal{C}_v$. Note that some of the points in the P_v, I_v and $\mathcal{C}_v$ sets might have been deleted by buffered updates at visited ancestor nodes.

A simple worst-case solution for answering queries would be to extract for all visited nodes v all points from P_v, I_v, D_v and $\mathcal{C}_c$ contained in Q. By sorting the $O(K + \frac{B}{\varepsilon}\log_B N)$ extracted points (bound follows from the analysis below) and applying the buffered updates we can answer a query in worst-case $O(\text{Sort}(K + \frac{B}{\varepsilon}\log_B N))$ IOs. In the following we prove the better bound of amortized $O(\frac{1}{\varepsilon}\log_B N + K/B)$ IOs by charging part of the work to the updates.

Our approach is to push buffered insertions and deletions down such that for all visited nodes v, no ancestor u of v stores any buffered updates in D_u and I_u that should go into the subtree of v. We do this by a top-down traversal of the visited nodes. For a visited node v we identify all the children to visit. For a child c to visit, let $U \subseteq D_v \cup I_v$ be all buffered updates belonging to the x-range of c. We delete all points in U from P_c, $\mathcal{C}_v$, I_c and D_c. All updates in U with y-value smaller than the minimum y-value in P_c are inserted into D_c or I_c, respectively. All insertions in U with y-value larger than or equal to the minimum y-value in P_c are merged with P_c. If $|P_c| > B$ we move the points with lowest y-values to I_c until $|P_c| = B$. We update $\mathcal{C}_v$ to reflect the changes to P_c. During this push down of updates, some update buffers at visited nodes might get size $> B$. We temporarily allow this, and keep update buffers in sorted x-order.

The reporting step consists of traversing all visited nodes v and reporting all points in $(P_v \cup I_v) \cap Q$ together with points in $\mathcal{C}_v$ contained in Q but not canceled by deletions in D_v, i.e. $(Q \cap \mathcal{C}_v) \setminus D_v$. Overflowing insertion and deletion buffers are finally handled as described in the update section, Section 4 (i)–(iv), possibly causing new nodes to be created by splits, where the amortized cost is already accounted for in the update analysis. The final step is to refill the P_v buffers of visited nodes, which might have underflowed due to the deletions pushed down among the visited nodes. The refilling is done as described in Section 4 (v).

Analysis

Assume $V + O(\frac{1}{\varepsilon}\log_B N)$ nodes are visited, where V nodes are not on the search paths for x_1 and x_2. Let R be the set of points in the point buffers of the V visited nodes before pushing updates down. Then we know $|R| \geq VB/2$. The number of buffered deletions at the visited nodes is at most $(V + O(\frac{1}{\varepsilon}\log_B N))B/4$, i.e. the number of points reported K is then at least $VB/2 - (V + O(\frac{1}{\varepsilon}\log_B N))B/4 = VB/4 - O(\frac{B}{\varepsilon}\log_B N)$. It follows $V = O(\frac{1}{\varepsilon}\log_B N + K/B)$. The worst-case IO bound becomes $O(V + \frac{1}{\varepsilon}\log_B N + K/B) = O(\frac{1}{\varepsilon}\log_B N + K/B)$, except for the cost of pushing the content of update buffers done at visited nodes and handling overflowing update buffers and underflowing point buffers.

Whenever we push $\Omega(B/\Delta)$ points to a child, the cost is covered by the analysis in Section 4. Only when we push $O(B/\Delta)$ updates to a visited child, with an amortized cost of $O(1)$ IOs, we charge this IO cost to the visited child. Overflowing update buffers and refilling P_v buffers is covered by the cost analyzed in Section 4. It follows that the total amortized cost of a 3-sided range reporting query in amortized $O(\frac{1}{\varepsilon}\log_B N + K/B)$ IOs.

7 Top-k Queries

Our overall approach for answering a top-k query for the range $[x_1, x_2]$ consists of three steps: First we find an approximate threshold y-value $\bar{y}$, such that we can reduce the query to a 3-sided range reporting query. Then we perform a 3-sided range reporting query as described in Section 6 for the range $[x_1, x_2] \times [\bar{y}, \infty]$. Let A be the output the three sided query. If $|A| \leq k$ then we return A. Otherwise, we select and return k points from A with largest y-value using the linear time selection algorithm of Blum et al. [7], that in

external memory uses $O(|A|/B)$ IOs. The correctness of this approach follows if $|A| \geq k$ or A contains all points in the query range, and the IO bound follows if $|A| = O(K + B\log_B N)$ and we can find $\bar{y}$ in $O(\log_B N + K/B)$ IOs. It should be noted that our $\bar{y}$ resembles the approximate k-threshold used by Sheng and Tao [19], except that we allow an additional slack of $O(\log_B N)$.

To compute $\bar{y}$ we (on demand) construct a heap-ordered binary tree $\mathcal{T}$ of sampled y-values, where each node can be generated using $O(1)$ IOs, and apply Frederickson's binary heap-selection to $\mathcal{T}$ to find the $O(k/B + \log_B N)$ largest y-value in $O(K/B + \log_B N)$ time and $O(K/B + \log_B N)$ IOs. This is the returned value $\bar{y}$. For each node v of the B-tree T we construct a path $\mathcal{P}_v$ of $O(\Delta)$ decreasing y values, consisting of the samples returned by Sample(x_1, x_2) for $\mathcal{C}_v$ and merged with the minimum y values of the point buffers P_c, for each child c within the x-range of the query and where $|P_c| \geq B/2$. The root of $\mathcal{P}_v$ is the largest y-value, and the remaining nodes form a leftmost path in decreasing y-value order. For each child c of v, the node in $\mathcal{P}_v$ storing the minimum y-value in P_c has as right child the root of $\mathcal{P}_c$. Finally let $v_1, v_2, \ldots, v_t$ be all the nodes on the two search paths in T for x_1 and x_2. We make a left path $\mathcal{P}$ containing t nodes, each with y-value $+\infty$, and let the root of $\mathcal{P}_{v_i}$ be the right child of the ith node on $\mathcal{P}$. Let $\mathcal{T}$ be the resulting binary tree. The $\bar{y}$ value we select is the $\bar{k} = \lceil 7t + 12k/B \rceil$-th among the nodes in the binary tree $\mathcal{T}$.

Analysis

We can construct the binary tree $\mathcal{T}$ topdown on demand (as needed by Frederickson's algorithm) using $O(1)$ IOs per node, since each $\mathcal{P}_v$ path can be computed using $O(1)$ IOs when Frederickson's algorithm visits the root of $\mathcal{P}_v$.

To lower bound the number of points in T contained in $Q_{\bar{y}} = [x_1, x_2] \times [\bar{y}, \infty]$, we first observe that among the $\bar{k}$ y-values in $\mathcal{T}$ larger than $\bar{y}$ are the t occurrences of $+\infty$, and either $\geq \frac{1}{3}(\bar{k} - t)$ samplings from $\mathcal{C}_v$ sets or $\geq \frac{2}{3}(\bar{k} - t)$ minimum values from P_v sets. Since s samplings from $\mathcal{C}_v$ ensures sB elements from $\mathcal{C}_v$ have larger values than $\bar{y}$ and the $\mathcal{C}_v$ sets are disjoint, the first case ensures that there are $\geq \frac{1}{3}B(\bar{k} - t)$ points from $\mathcal{C}_v$ sets in $Q_{\bar{y}}$. For the second case each minimum y-value of a P_v set represents $\geq B/2$ points in P_v contained in $Q_{\bar{y}}$, i.e. in total $\geq \frac{B}{2}\frac{2}{3}(\bar{k} - t) = \frac{1}{3}B(\bar{k} - t)$ points. Some of these elements will not be reported, since they will be canceled by buffered deletions. These buffered deletions can only be stored at the t nodes on the two search paths and in nodes where all $\geq B/2$ points in P_v are in $Q_{\bar{y}}$. It follows at most $\frac{B}{4}(t + \bar{k})$ buffered deletions can be applied to points in the P_v sets, i.e. in total at least $\frac{B}{3}(\bar{k} - t) - \frac{B}{4}(t + \bar{k}) = \frac{B}{12}\bar{k} - \frac{7B}{12}t = \frac{B}{12}\lceil 7t + 12k/B \rceil - \frac{7B}{12}t \geq k$ points will be reported by the 3-sided range reporting $Q_{\bar{y}}$.

To upper bound the number of points that can be reported by $Q_{\bar{y}}$, we observe that these points are stored in P_v, $\mathcal{C}_v$ and I_v buffers. There are at most $\bar{k}$ nodes where all $\geq B/2$ points in P_v are reported (remaining points in point buffers are reported using $\mathcal{C}_v$ structures), at most from $t+\bar{k}$ nodes we need to consider points from the insertion buffers I_v, and from the at most $t+\bar{k}$ child structures $\mathcal{C}_v$ we report at most $\bar{k}B+(\alpha+1)(t+\bar{k})B$ points, for some constant $\alpha \geq 1$, which follows from the interface of the Sample operation from Section 2. In total the 3-sided query reports at most $\bar{k}B+(t+\bar{k})B+\bar{k}B+(\alpha+1)(t+\bar{k})B = O(B(t+\bar{k})) = O(\frac{1}{\varepsilon}B\log_B N+k)$ points. In the above we ignored the case where we only find $< \bar{k}$ nodes in $\mathcal{T}$, where we just set $\bar{y} = -\infty$ and all points within the x-range will be reported. Note that the IO bounds for finding $\bar{y}$ and the final selection are worst-case, whereas only the 3-sided range reporting query is amortized.

References

1. Peyman Afshani, Gerth Stølting Brodal, and Norbert Zeh. Ordered and unordered top-k range reporting in large data sets. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 390–400. SIAM, 2011.
2. Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988. doi:10.1145/48529.48535.
3. Lars Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003. doi:10.1007/s00453-003-1021-x.
4. Lars Arge, Vasilis Samoladas, and Jeffrey Scott Vitter. On two-dimensional indexability and optimal range search indexing. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS*, pages 346–357. ACM, 1999.
5. Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–189, 1972. doi:10.1007/BF00288683.
6. Gabriele Blankenagel and Ralf Hartmut Güting. XP-trees – external priority search trees. Technical Report Informatik-Bericht Nr. 92, Fern Universität Hagen, 1990.
7. Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.
8. Gerth Stølting Brodal and Rolf Fagerberg. Lower bounds for external memory dictionaries. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 546–554. ACM/SIAM, 2003.
9. Gerth Stølting Brodal, Rolf Fagerberg, Mark Greve, and Alejandro López-Ortiz. Online sorted range reporting. In *Proceedings of the 20th International Symposium Algorithms and Computation, ISAAC*, volume 5878 of *Lecture Notes in Computer Science*, pages 173–182. Springer, 2009.
10. Greg N. Frederickson. An optimal algorithm for selection in a min-heap. *Information and Computation*, 104(2):197–214, 1993.
11. John Iacono and Mihai Pătraşcu. Using hashing to solve the dictionary problem. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 570–582. SIAM, 2012.
12. Christian Icking, Rolf Klein, and Thomas Ottmann. Priority search trees in secondary memory (extended abstract). In *Graph-Theoretic Concepts in Computer Science, International Workshop, WG*, volume 314 of *Lecture Notes in Computer Science*, pages 84–93. Springer, 1987.
13. Paris C. Kanellakis, Sridhar Ramaswamy, Darren Erik Vengroff, and Jeffrey Scott Vitter. Indexing for data models with constraints and classes. *Journal of Computing and System Sciences*, 52(3):589–612, 1996.
14. Edward M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14(2):257–276, 1985. doi:10.1137/0214021.
15. Mark H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes in Computer Science*. Springer, 1983. doi:10.1007/BFb0014927.
16. Saladi Rahul, Prosenjit Gupta, Ravi Janardan, and K. S. Rajan. Efficient top-k queries for orthogonal ranges. In *Proceedings 5th International Workshop on Algorithms and Computation, WALCOM*, volume 6552 of *Lecture Notes in Computer Science*, pages 110–121. Springer, 2011.
17. Saladi Rahul and Yufei Tao. On top-k range reporting in 2D space. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS*, pages 265–275. ACM, 2015. doi:10.1145/2745754.2745777.

18 Sridhar Ramaswamy and Sairam Subramanian. Path caching: A technique for optimal external searching. In *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS*, pages 25–35. ACM, 1994.

19 Cheng Sheng and Yufei Tao. Dynamic top-k range reporting in external memory. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 121–130. ACM, 2012.

20 Sairam Subramanian and Sridhar Ramaswamy. The P-range tree: A new data structure for range searching in secondary memory. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 378–387. ACM/SIAM, 1995.

21 Yufei Tao. A dynamic I/O-efficient structure for one-dimensional top-k range reporting. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 256–265. ACM, 2014. doi:10.1145/2594538.2594543.

22 Robert Endre Tarjan. Amortized computational complexity. *SIAM Journal on Algebraic Discrete Methods*, 6(2):306–318, 1985.

23 Elad Verbin and Qin Zhang. The limits of buffering: A tight lower bound for dynamic membership in the external memory model. *SIAM Journal on Computing*, 42(1):212–229, 2013. doi:10.1137/110842211.

Catalytic Space: Non-determinism and Hierarchy*

Harry Buhrman[1], Michal Koucký[2], Bruno Loff[2], and Florian Speelman[4]

1 **CWI and University of Amsterdam, Amsterdam, The Netherlands**
h.buhrman@cwi.nl
2 **Charles University, Prague, Czech Republic**
koucky@iuuk.mff.cuni.cz
3 **Charles University, Prague, Czech Republic**
bruno.loff@gmail.com
4 **CWI, Amsterdam, The Netherlands**
f.speelman@cwi.nl

Abstract

Catalytic computation, defined by Buhrman, Cleve, Koucký, Loff and Speelman (STOC 2014), is a space-bounded computation where in addition to our working memory we have an exponentially larger auxiliary memory which is full; the auxiliary memory may be used throughout the computation, but it must be restored to its initial content by the end of the computation.

Motivated by the surprising power of this model, we set out to study the non-deterministic version of catalytic computation. We establish that non-deterministic catalytic log-space is contained in ZPP, which is the same bound known for its deterministic counterpart, and we prove that non-deterministic catalytic space is closed under complement (under a standard derandomization assumption). Furthermore, we establish hierarchy theorems for non-deterministic and deterministic catalytic computation.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases catalytic computation, Immerman–Szelepcsényi theorem, space hierarchy

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.24

1 Introduction

Buhrman et al. [3] define the notion of *catalytic computation*, a space-bounded model of computation in which the usual Turing machine has, in addition to its work tape, access to a large auxiliary memory which is full. The auxiliary memory can be used during the computation, but its starting contents must be restored by the end of the computation. The space usage that is counted is the amount of work space s used; the auxiliary memory is for free. In a reasonable setting, the auxiliary memory is of size at most 2^s. One can think of the auxiliary memory as a hard disk full of data. The catch with the auxiliary memory is that it may contain arbitrary content, possibly incompressible, which has to be preserved in some way during the computation. It is not obvious whether such auxiliary memory can be

* The first author was partially supported by NWO gravitation project Networks, EU 7th framework project SIQS and QuSoft. The research leading to these results has received funding from the European Research Council under the European Unions Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement n. 616787. The second author was partially supported also by a grant from Neuron Fund for Support of Science.

© Harry Buhrman, Michal Koucký, Bruno Loff, and Florian Speelman;
licensed under Creative Commons License CC-BY

useful at all. Buhrman et al. show that, surprisingly, there is a non-trivial way of using the full memory; that it is possible to compute in work space $O(\log n)$ (*catalytic log-space,* CL) functions not known to be computable in the usual logarithmic space (*log-space,* L) without the auxiliary memory. Indeed, all of TC^1, which includes NL and LOGCFL, is contained in CL.

This motivated us to explore further: What other problems can be solved in catalytic log-space? Buhrman et al. show $\mathsf{CL} \subseteq \mathsf{ZPP}$, so CL is unlikely to contain the whole of PSPACE (even though this is the case relative to some oracle). The fact that $\mathsf{NL} \subseteq \mathsf{CL}$ suggests an obvious question: what about non-deterministic catalytic log-space? Could it be that non-deterministic computation equipped with auxiliary tape has the same power as deterministic catalytic computation? Non-deterministic catalytic computation could possibly allow us to identify further problems that can benefit from having full memory. The previous work also raises a host of further question about the catalytic model such as: Is there a space hierarchy? Does some kind of Savitch's theorem hold for catalytic log-space? Is non-deterministic catalytic space closed under complement? *etc.* This paper aims to shed light on some of these questions.

In this paper we show that non-deterministic catalytic space is closed under complement under a widely accepted derandomization assumption. We also establish hierarchy theorems for catalytic computation in the deterministic and non-deterministic settings. For our non-deterministic catalytic log-space we can also establish the same ZPP upper bound that was known for CL. Hence there seems to be a closeness between determinism and non-determinism for catalytic computation. Despite that we are unable to establish an equivalent of Savitch's theorem. This remains an intriguing open problem.

We prove the closure under complement using the inductive counting technique of Immerman and Szelepcsényi [4, 9]. However, we had to overcome several difficulties. One challenge is that we might be faced with an exponential-size graph of reachable configurations. We show how to use a pseudorandom generator to avoid such a situation. Another issue is that for inductive counting we need to be able to remember and reason about different configurations. However, the full description of a configuration is exponentially bigger than our work space, so we cannot possibly store it in full. This is one of the hurdles that prevents us from carrying out Savitch's algorithm for catalytic computation. For the inductive counting we resolve this issue by using fingerprints for various configurations.

Our hierarchy theorems are proven in the setting of computation with advice. The catalytic model is a semantic restriction. It is an easy exercise to show that it is algorithmically undecidable whether a machine will restore the full memory on every input to its original content. For semantic models of computation, like bounded-error randomized computation, the only hierarchy theorems that we know of are in the setting with advice. The reason is that essentially all known hierarchy theorems are proven by diagonalization, which requires the ability to enumerate exactly all machines of a given type. We do not know any such enumeration for catalytic machines so we have to settle for the weaker result. The advice is used only to tell the diagonalizing machine whether it is safe to diagonalize against a particular machine. The hierarchy theorems follow from the work of Kinne and van Melkebeek, and van Melkebeek and Pervyshev [7, 10]. For some space bounds we provide more accurate separations that were not explicitly calculated before.

The layout of the paper is as follows. Section 2 contains some preliminaries. In Section 3 we define non-deterministic catalytic computation, and prove that the corresponding log-space class CNL is contained in ZPP. Section 4 is devoted to proving that CNL is closed under complement, and in Section 5 we show hierarchy theorems for catalytic computation.

2 Preliminaries

We assume the reader is familiar with basic computational complexity; a good reference is [2]. The complexity class L denotes the problems solvable in log-space, while PSPACE is the class of those problems that can be solved using a polynomial amount of space. The class NL contains the problems that can be solved *non-deterministically* in log-space, and LOGCFL is the class of problems that are log-space many-one reducible to context-free languages.

The problems in ZPP (zero-error probabilistic polynomial time) are the ones computable by a probabilistic Turing machine that halts in expected polynomial time, while always outputting the correct answer for any input.

We mention the circuit class TC^1, which is the class of boolean functions computable by circuits of depth $O(\log n)$ by AND gates, OR gates and MAJ gates, all with unbounded fan-in – a MAJ gate outputs 1 if and only if most of its input bits are 1. We use $\mathsf{SIZE}(s)$ to denote the class of problems that can be solved by circuits of size s.

The formal definition of catalytic computation [3] is the following:

▶ **Definition 1.** Let $\mathcal{M}$ be a deterministic Turing machine with four tapes: one input and one output tape, one work-tape, and one *auxiliary tape* (or *aux-tape*).

$\mathcal{M}$ is said to be a *catalytic Turing machine* using workspace $s(n)$ and auxiliary space $s_a(n)$ if for all inputs $x \in \{0,1\}^n$ and auxiliary tape contents $w \in \{0,1\}^{s_a(n)}$, the following three properties hold.

1. **Space bound.** The machine $\mathcal{M}(x,w)$ uses space $s(n)$ on its work tape and space $s_a(n)$ on its auxiliary tape.
2. **Catalytic condition.** $\mathcal{M}(x,w)$ halts with w on its auxiliary tape.
3. **Consistency.** The outcome of the computation $\mathcal{M}(x,w)$ is consistent among all initial aux-tape contents w.[1]

From this we obtain an analogue of the usual space-bounded complexity classes:

▶ **Definition 2.** $\mathsf{CSPACE}(s(n), s_a(n))$ is the class of decision problems solvable by a catalytic Turing machine using workspace $s(n)$ and auxiliary space $s_a(n)$. The notational shorthand $\mathsf{CSPACE}(s(n))$ is defined as $\mathsf{CSPACE}(s(n), 2^{s(n)})$. The class CL is $\mathsf{CSPACE}(O(\log n))$.

In the paper [3], it was shown that, surprisingly, CL can make a non-trivial use of the auxiliary tape. Indeed, the paper shows that $\mathsf{TC}^1 \subseteq \mathsf{CL}$, but it is generally believed that $\mathsf{TC}^1 \not\subseteq \mathsf{L}$.

In this paper we will prove a space-hierarchy theorem for catalytic computations. This hierarchy theorem holds for catalytic Turing machines with an advice string.

We define advice added to a catalytic computation in the same way as in the recent line of research that proves hierarchies for certain classes of semantic models, see for example [10, 7]. In our case that means that a computation needs to satisfy the catalytic condition and consistency properties on the *correct* advice, and is allowed to (for example) fail to restore the contents of the aux-tape for other values of the advice. This notion of advice is a variation on the one defined by Karp and Lipton [6], who required that the machine model was robust under all possible values of the advice string. Proving the same hierarchy theorem using the Karp–Lipton definition would be harder, and would indeed imply a hierarchy theorem that also holds without any advice [7].

[1] What this means depends on what we are trying to do. For instance, when solving a decision problem, $\mathcal{M}(x,w)$ should either accept for all choices of w – in which case we say $\mathcal{M}$ accepts x – or it rejects for all possible w – $\mathcal{M}$ rejects x.

We will also prove an analogue of the Immerman–Szelepcsényi theorem. The definition of the non-deterministic version of CL, denoted CNL, will be left for Section 3. Then CNL = coCNL will hold under the same assumption as the following standard derandomization result, whose proof is now standard. (For instance, the pseudo-random generator of [5] has the right properties, or see Appendix C of [8] and Theorem 19 of [1].)

▶ **Lemma 3.** *If there exists a constant $\varepsilon > 0$ such that* $\mathsf{DSPACE}(n) \not\subseteq \mathsf{SIZE}(2^{\varepsilon n})$ *then for all constants c there exists a constant c' and a function $G : \{0,1\}^{c' \log n} \to \{0,1\}^n$ such that for any circuit $\mathcal{C}$ of size n^c*

$$\left| \Pr_{r \in \{0,1\}^n} [\mathcal{C}(r) = 1] - \Pr_{s \in \{0,1\}^{c' \log n}} [\mathcal{C}(G(s)) = 1] \right| < \frac{1}{n}$$

and G is computable in space logarithmic in n.

We will also need a hash family with nice properties. Such a hash family can be easily constructed.

▶ **Lemma 4.** *For every n, there exists a family of hash functions $\{h_k\}_{k=1}^{n^3}$, with each h_k a function $\{0,1\}^n \to \{0,1\}^{4 \log n}$, such that the following properties hold. First, h_k is computable in space $O(\log n)$ for every k, and second, for every set $S \subset \{0,1\}^n$ with $|S| \leq n$ there is a hash function in the family that is injective on S.*

Remarks on notation

For two binary strings x, y of equal length, we use $x \oplus y$ for the bitwise XOR of x and y. The function log always stands for the logarithm of base 2. For simplicity, all Turing machines are assumed to use a binary alphabet – all definitions and proofs would easily generalize to larger alphabet sizes, at the cost of introducing notational clutter.

3 Non-deterministic Catalytic Computation

The model for catalytic computation is defined in terms of deterministic Turing machines. This gives rise to the question: What would the power of a non-deterministic version of CL be? In this section we extend the definitions of catalytic-space computation to the non-deterministic case, and prove basic results about this model.

There are multiple possible ways how to add non-determinism to a catalytic Turing machine. Our definition will require the machine to restore the contents of the auxiliary tape for any given sequence of non-deterministic bits; but at a first glance, it seems we could make this requirement only for those non-deterministic guesses which result in accepting states. We feel that defining the model in this way is less natural for several reasons. For one, we can not run two machines sequentially and accept if one of them accepts: if one of the two machines would reject, the whole computation needs to reject, because the auxiliary tape may have been irreversibly changed; so the class would not be closed under union. This would also prevent amplification of success probability in a probabilistic class defined using such machines. Philosophically speaking, having a catalytic machine which 'sometimes' destroys all data it is guaranteed to preserve, seems to go against the spirit of the model.

Another possible variation would be to require that the accepting sequence of non-deterministic choices is independent of the initial contents of the auxiliary tape, which would give a weaker model. Indeed, this would not look very strange in a certificate definition, effectively requiring that there exists a read-once certificate, independent of the initial

contents of the aux-tape, which can be verified by a deterministic log-space catalytic Turing machine. Even so, when describing the model with non-deterministic Turing machines it seems unnatural to have this restriction. Hence we settle on the following:

▶ **Definition 5.** Let $\mathcal{M}$ be a non-deterministic Turing machine with four tapes: one input and one output tape, one work-tape, and one *auxiliary tape*.

Let $x \in \{0,1\}^n$ be an input, and $w \in \{0,1\}^{s_a(n)}$ be the initial contents of the auxiliary tape. We say that $\mathcal{M}(x,w)$ accepts x if there exists a sequence of nondeterministic choices that makes the machine accept. If for all possible sequences of nondeterministic choices $\mathcal{M}(x,w)$ does not accept, the machine rejects x.

Then $\mathcal{M}$ is said to be a *catalytic non-deterministic Turing machine* using workspace $s(n)$ and auxiliary space $s_a(n)$ if for all inputs, the following three properties hold.

1. **Space bound.** The machine $\mathcal{M}(x,w)$ uses space $s(n)$ on its work tape and space $s_a(n)$ on its auxiliary tape.
2. **Catalytic condition.** $\mathcal{M}(x,w)$ halts with w on its auxiliary tape, irrespective of its nondeterministic choices.
3. **Consistency.** The outcome of the computation $\mathcal{M}(x,w)$ is consistent among all initial aux-tape contents w. This means that for any given input x, $M(x,w)$ should always accept, or always reject, regardless of w; *however*: the specific nondeterministic choices that make $\mathcal{M}(x,w)$ go one way or the other may depend on w.

▶ **Definition 6.** $\mathsf{CNSPACE}(s(n), s_a(n))$ is the class of decision problems solvable by a catalytic Turing machine using workspace $s(n)$ and auxiliary space $s_a(n)$, and $\mathsf{CNSPACE}(s(n))$ is defined as $\mathsf{CNSPACE}(s(n), 2^{s(n)})$. The class CNL is $\mathsf{CNSPACE}(O(\log n))$.

We now have an analogue of non-deterministic space-bounded complexity. In [3], we proved that $\mathsf{CL} \subseteq \mathsf{ZPP}$; we now generalize this to $\mathsf{CNL} \subseteq \mathsf{ZPP}$.

▶ **Definition 7.** Define the directed acyclic graph $\mathcal{G}_{\mathcal{M},x,w}$ to be the configuration graph of a catalytic non-deterministic Turing machine $\mathcal{M}$ on input x and auxiliary tape starting contents w. That is, $\mathcal{G}_{\mathcal{M},x,w}$ has a node for every configuration which is reachable by non-deterministic choices when executing $\mathcal{M}(x,w)$.

We will use $|\mathcal{G}_{\mathcal{M},x,w}|$ to denote the number of nodes of the configuration graph.

▶ **Lemma 8.** *Let $\mathcal{M}$ be a non-deterministic catalytic machine using space $c\log n$ and let $c' = 2c+2$. Then for all x*

$$\mathop{\mathbb{E}}_{w \in_R \{0,1\}^{n^c}} \left[|\mathcal{G}_{\mathcal{M},x,w}| \right] \leq O(n^{c'}) .$$

Proof. Notice that, for any given $x \in \{0,1\}^n$, and for different auxiliary tape contents w, w', the set of configurations in $\mathcal{G}_{\mathcal{M},x,w}$ and in $\mathcal{G}_{\mathcal{M},x,w'}$ have to be disjoint. For the sake of contradiction, consider a configuration q that is reachable both by $\mathcal{M}(x,w)$ and by $\mathcal{M}(x,w')$. Then any halting configuration reachable by q will have the wrong contents on its auxiliary tape for either the computation that started with w or with w'.

The number of bits needed to describe a configuration of $\mathcal{M}$, excluding the contents of the input tape, is bounded by

$$c\log n + n^c + \log n^c + \log n + \log(c\log n) + O(1) \leq (2c+2)\log n + n^c + O(1),$$

where we do include the encoding of the location of the tape heads and the internal state of the Turing Machine. Therefore the total number of reachable configurations, counted over

all possible starting auxiliary tape contents, is at most

$$\sum_{w \in \{0,1\}^{n^c}} |\mathcal{G}_{\mathcal{M},x,w}| \leq 2^{c' \log n + n^c + O(1)} = O(n^{c'}) 2^{n^c}$$

And thus $\mathbb{E}_{w \in_R \{0,1\}^{n^c}} \left[|G_{\mathcal{M},x,w}| \right] \leq O(n^{c'})$. ◀

Now suppose we have CNL machine $\mathcal{M}$, and let $x \in \{0,1\}^n$ be the input string. Consider an algorithm which flips a random string w and searches $\mathcal{G}_{\mathcal{M},x,w}$ for a path from the initial configuration to an accepting configuration. This takes time polynomial in $|G_{\mathcal{M},x,w}|$. By Lemma 8 this graph is polynomial-sized in expectation, and therefore this procedure finishes in expected polynomial time. Thus we obtain:

▶ **Corollary 9.** CNL $\subseteq$ ZPP.

4 An Analog of the Immerman–Szelepcsényi Theorem

This section is devoted to proving that CNL is closed under complement. Our proof strategy is based on the inductive-counting argument to prove the Immerman–Szelepcsényi theorem. In order for the proof to work for catalytic computation, we will need a couple of new ideas.

Suppose we are given a CNL machine $\mathcal{M}$, and wish to construct a CNL-machine $\mathcal{M}'$ to compute the complement $\mathcal{M}$, via an inductive-counting argument on the configuration graph of $\mathcal{M}$.

First of all, notice that whenever $\mathcal{M}'$ wishes to simulate a run of $\mathcal{M}$, it must necessarily use its own aux-tape to simulate the aux-tape of $\mathcal{M}$, because it is the only read-write tape that is big enough.

Now, for some w (initial contents of the aux-tape), $\mathcal{M}$ may visit exponentially many configurations. Then the inductive counting would be impossible to do with only logarithmic space. So the first idea is to use the pseudo-random generator G of Lemma 3 to avoid such *bad* w, by using the binary XOR $w \oplus G(s)$ for different seeds s. Lemma 10 below explains why this works.

Notice also that we must be careful that $\mathcal{M}'$, when simulating a run of $\mathcal{M}$, can always restore the initial contents of its aux-tape. We can make sure this happens correctly by using the catalytic condition applied to $\mathcal{M}$: whenever we need to restore the initial contents of the aux-tape, it will be enough to run the simulation of $\mathcal{M}$ to an arbitrary halting configuration.

Finally, recall that the inductive-counting argument involves storing and comparing configurations of $\mathcal{M}$; but the configurations of $\mathcal{M}$ include the aux-tape, and are too big for $\mathcal{M}'$ to store on its work tape. So the second idea is to use the family of hash functions of Lemma 4, and do inductive-counting by storing and comparing *the hashes* of configurations instead.

Putting the whole thing together, however, is rather delicate, because our pseudo-random generator will still give us *bad seeds* – meaning $w \oplus G(s)$ might visit too many configurations. Furthermore, even if we pick a good seed, we may still happen to pick a *bad hash function* – meaning a hash function which is not collision-free on the set of reachable configurations. So the algorithm needs to be able to handle bad seeds and bad hash functions.

It will happen that a bad seed may lead us to falsely certifying that the accepting configuration is unreachable, when in fact it is reachable. This is solved simply by trying all seeds and doing a majority vote.

For good seeds, the number of reachable configurations is bounded by $c = n^{O(1)}$, but it may still happen that the hash collisions of a bad hash function will lead us to falsely believe

that there are fewer reachable configurations than the actual number (that c is smaller than it actually is) – because configurations with the same hash are only counted once. But fortunately, good hash functions will give us the correct c, and bad hash functions will always give us a smaller value. So we overcome this problem by remembering, for all hash function we try, the largest claimed number of reachable configurations – this will be the true c.

Let us start by showing how to avoid bad w's.

▶ **Lemma 10.** *Assume the derandomization condition of Lemma 3, and let G be as given therein. Let $\mathcal{M}$ be a non-deterministic catalytic Turing machine using workspace $c \log n$. Then, for every input x and aux-tape contents w, at least half of the seeds $s \in \{0,1\}^{O(\log n)}$ will cause the non-deterministic computation $\mathcal{M}(x, G(s) \oplus w)$ to reach at most n^{2c+3} many different configurations.*

Proof. Let $\mathcal{M}$ be a CNL machine using workspace $c \log n$ and auxiliary space n^c. Let $x \in \{0,1\}^n$, $w \in \{0,1\}^{n^c}$ be given.

Let $C_{x,w}$ be a boolean circuit which, on input $r \in \{0,1\}^{n^c}$, does a breadth-first traversal of $\mathcal{G}_{\mathcal{M},x,r\oplus w}$[2], starting on the initial configuration, until either:

(i) More than n^{2c+3} nodes have been found, in which case it outputs 0; or

(ii) The graph has been fully traversed, in which case it outputs 1.

The size of $C_{x,w}$ can be bounded by a polynomial, say n^d. The circuit $C_{x,w}$ outputs 1 on input r if and only if $|\mathcal{G}_{\mathcal{M},x,r\oplus w}| \leq n^{2c+3}$. Therefore, for large enough n, for all $x \in \{0,1\}^n$ and all $w \in \{0,1\}^{n^c}$,

$$\Pr_{r\in_R\{0,1\}^{n^c}}[\mathcal{C}_{x,w}(r) = 0] = \Pr_{r\in_R\{0,1\}^{n^c}}\left[\,|\mathcal{G}_{\mathcal{M},x,r\oplus w}| \geq n^{2c+3}\,\right]$$

$$= \Pr_{r\in_R\{0,1\}^{n^c}}\left[\,|\mathcal{G}_{\mathcal{M},x,r}| \geq n^{2c+3}\,\right] \leq \frac{1}{n^{2c+3}} \mathop{\mathbb{E}}_{r\in_R\{0,1\}^{n^c}}\left[\,|\mathcal{G}_{\mathcal{M},x,r}|\,\right] \leq O\left(\frac{1}{n}\right).$$

Here we have used the fact that, for a fixed w, r and $r \oplus w$ are equidistributed. The last inequality follows from Markov's inequality and Lemma 8.

Now Lemma 3 provides us with a log-space computable function $G : \{0,1\}^{O(\log n)} \to \{0,1\}^{n^c}$ such that, for all $x \in \{0,1\}^n$ and $w \in \{0,1\}^{n^c}$,

$$\left|\Pr_{r\in\{0,1\}^{n^c}}[\mathcal{C}_{x,w}(r) = 0] - \Pr_{s\in\{0,1\}^{O(\log n)}}[\mathcal{C}_{x,w}(G(s)) = 0]\right| \leq \frac{1}{n}.$$

In particular, for all sufficiently large n we get the rough bound:

$$\Pr_{s\in\{0,1\}^{O(\log n)}}[\mathcal{C}_w(x, G(s)) = 0] \leq \frac{1}{n} + O\left(\frac{1}{n}\right) < \frac{1}{2}.$$

Therefore, for any x and w, at least half of the seeds s will ensure that the configuration graph $\mathcal{G}_{\mathcal{M},x,G(s)\oplus w}$ has at most n^{2c+3} nodes. ◀

Our goal is now to use an inductive counting argument on $\mathcal{G}_{\mathcal{M},x,G(s)\oplus w}$. Like we mentioned earlier, inductive counting requires us to write down configurations in the work tape, but the tape is not big enough. To circumvent this, we will instead write down the *hash values* of the configurations, via the hash family of Lemma 4. The proof below puts it all together.

[2] Recall that $\mathcal{G}_{\mathcal{M},x,r\oplus w}$ is the configuration graph of $\mathcal{M}$, for input x and aux-tape contents given by the bit-wise XOR of r and w.

▶ **Theorem 11** (Immerman–Szelepcsényi for catalytic computation). *If there exists a constant* $\varepsilon > 0$ *such that* $\mathsf{DSPACE}(n) \not\subseteq \mathsf{SIZE}(2^{\varepsilon n})$ *then* $\mathsf{CNL} = \mathsf{coCNL}$.

Proof. Let $\mathcal{M}$ be a nondeterministic Turing machine that uses $d \log n$ work space, and has an auxiliary tape of size n^d. We wish to construct a nondeterministic catalytic Turing machine $\mathcal{M}'$, using workspace $O(\log n)$, such that for any n and any input $x \in \{0,1\}^n$ our computation accepts x if $\mathcal{M}$ rejects x, and vice-versa.

Without loss of generality, assume that for any given $w \in \{0,1\}^{n^d}$, $\mathcal{M}(x,w)$ has a unique accepting configuration acc_w. Let $start_w$ be the initial configuration of $\mathcal{M}(x,w)$ and let $e = 2d + 3$.

By the consistency property, either there exists a path from $start_w$ to acc_w for all w, or it is impossible to reach acc_w from $start_w$, for any w. We prove Theorem 11 by describing a way of certifying that there exists no path between $start_w$ and acc_w in $\mathcal{G}_{\mathcal{M},x,w}$.

Fix some input x, and let w' denote the initial contents of the aux-tape of $\mathcal{M}'$. By Lemma 10, we know that for at least half of the possible seeds $s \in \{0,1\}^{O(\log n)}$, we have

$$|\mathcal{G}_{\mathcal{M},x,G(s)\oplus w'}| \leq n^e \,. \tag{1}$$

If (1) holds, we say s is a *good seed*.

Lemma 4 gives us a family of hash functions $\{h_k\}_{k=1}^{n^{3e}}$, with the property that, for every good seed s, there is at least one hash function in the family which is one-to-one on the nodes of $\mathcal{G}_{\mathcal{M},x,w}$.

In page 9, we give the pseudo-code for $\mathcal{M}'$'s algorithm. Let us now do a guided reading of this code. We begin by breaking the code into three sections, for the lines 2–6, 7–26, and 27–32.

In lines 2–6, we initialize a variable N to 0 (line 2), cycle through every seed s (line 3), XOR the contents of the aux-tape with $G(s)$ (line 4), and initialize two variables g and ℓ to 0 (lines 5 and 6).

Then, in lines 7–26, we have an inner loop that cycles through every hash function (line 7). Below we will prove:

Property I. If the seed s is good, then **(I.a)** some sequence of non-deterministic bits will cause the inner loop to exit normally at line 27, with the promise that $g = |\mathcal{G}_{\mathcal{M},x,w}|$, and that h_ℓ is one-to-one on $\mathcal{G}_{\mathcal{M},x,w}$; and **(I.b)** any sequence of non-deterministic bits that fails this promise will exit the inner loop by jumping directly to line 30.

At line 27, we use the value of g and ℓ we have obtained to try and certify that acc_w is not reachable. If we succeed to do so, we increment N (line 28). Below we will also prove:

Property II. If the seed s is good, $g = |\mathcal{G}_{\mathcal{M},x,w}|$, and h_ℓ is one-to-one on $\mathcal{G}_{\mathcal{M},x,w}$, then some sequence of non-deterministic bits will cause us to successfully certify that acc_w is not reachable if and only if $M(x,w)$ rejects.[3]

Before we move on to the next seed, we first restore the initial contents of the aux-tape, by once again XORing them with $G(s)$ (line 30).

Finally, the procedure accepts if and only if $N > S/2$ in line 32. Let us prove that, assuming Properties I and II, the procedure accepts if and only if $M(x,w)$ rejects. Lemma 10 ensures that more than half the seeds are good, and hence:

[3] But if s, g or h_ℓ are not as assumed, we might get a false-positive, claiming that acc_w is not reachable when in fact it is.

Algorithm 1 Pseudo-code for $\mathcal{M}'$.

Here G is the log-space PRG of Lemma 3, S is the number of seeds, $M = n^e$ stands for the maximum number of configurations allowed in the configuration graph, and H is the size of the hash family given by Lemma 4. The aux-tape is represented by a variable w, whose initial value is w'. The lines that use non-determinism are marked with a $(*)$.

```
1: procedure coCNL-Simulation(Input x, Aux-Tape w ← w')
2:     N ← 0
3:     for s = 0...S do
4:         w ← G(s) ⊕ w
5:         g ← 0
6:         ℓ ← 0
7:         for k = 1...H do
8:             c ← 1
9:             for i = 1...M do
10:                c' ← 0
11:                for v = 0...M do
12:                    if canReach(v, i, h_k) then                    ▷ (*)
13:                        c' ← c' + 1
14:                    else if cannotReach(v, i, c, h_k) then         ▷ (*)
15:                        Do nothing
16:                    else
17:                        Jump to line 30
18:                    end if
19:                end for
20:                c ← c'
21:            end for
22:            if c > g then
23:                g ← c
24:                ℓ ← k
25:            end if
26:        end for
27:        if cannotReach(h_ℓ(acc_w), M + 1, g, h_ℓ) then             ▷ (*)
28:            N ← N + 1
29:        end if
30:        w ← G(s) ⊕ w
31:    end for
32:    Accept if N > S/2, and Reject otherwise
33: end procedure
```

1. If $M(x, w)$ rejects: Property I ensures that, for each good seed s, some non-deterministic guess will cause us to reach line 27 with $g = |\mathcal{G}_{\mathcal{M},x,w}|$ and h_ℓ one-to-one on $\mathcal{G}_{\mathcal{M},x,w}$; then Property II ensures that some further guess will result in N being incremented; hence some overall non-deterministic guess will give $N > S/2$, and the procedure will accept in line 32.
2. If $M(x, w)$ accepts: Property I ensures that, for each good seed s, if we reach line 27, then $g = |\mathcal{G}_{\mathcal{M},x,w}|$ and h_ℓ one-to-one on $\mathcal{G}_{\mathcal{M},x,w}$, and thus, by Property II, N will not be incremented in line 28. If some non-deterministic guess fails to get us to line 27, then Property I tells us that the execution jumped directly to line 30, so N was again not incremented. Because no good seed will ever cause N to be incremented, $N < S/2$ and the procedure rejects in line 32.

So all we need to do is prove properties I and II. We first need to specify the canReach and cannotReach subroutines. Their correctness is easy to see from the description and pseudo-code. The canReach(v, i, h_k) subroutine (see page 10) checks whether there is a node w in $\mathcal{G}_{\mathcal{M},xw}$, reachable within i steps, with $h_k(w) = v$.

Behavior of the canReach subroutine. If such a w exists, then some non-deterministic guess will cause the procedure to return TRUE, and, otherwise, every non-deterministic guess will return FALSE.

canReach non-deterministically works as follows: we guess a length $L \leq i$, and simulate $\mathcal{M}$ for L steps. After this, we hash the configuration $\mathcal{M}$ is currently in, and compare it to v. We will then return TRUE if and only if the two hashes are the same, but before we return, we finish the simulation of $\mathcal{M}$ until we reach a halting state, in order to restore the contents of the aux-tape.

Algorithm 2 The canReach subroutine.

The subroutine to check that a node hashing to v is reachable in at most i steps, given some hash function h_k.

```
1:  procedure canReach(v, i, h_k)
2:      z ← 0                          ▷ Workspace and internal state of simulated machine
3:      Non-deterministically guess L ≤ i                                        ▷ (∗)
4:      Simulate M(x, w) using z as workspace for L steps                        ▷ (∗)
5:      if h_k(z, w') = v then
6:          r ← TRUE
7:      else
8:          r ← FALSE
9:      end if
10:     Continue simulation of M(x, w) using z and reach any halting state
11:     return r
12: end procedure
```

The cannotReach(v, i, c, h_k) subroutine (see page 11) checks that there is no node in $\mathcal{G}_{\mathcal{M},x,w}$ hashing to v and reachable within i steps, as long as c and h_k fulfill the promise that there are exactly c nodes in $\mathcal{G}_{\mathcal{M},x,w}$ that are reachable within $i - 1$ steps, and that h_k is one-to-one on $\mathcal{G}_{\mathcal{M},x,w}$.

Algorithm 3 The CANNOTREACH subroutine.

The subroutine checking that a node hashing to v is *not* reachable within i steps, for hash function h_k, when given c, the number of nodes reachable in $i-1$ steps.

1: **procedure** CANNOTREACH(v, i, c, h_k)
2: $h' \leftarrow -1$ ▷ Hash of previously seen node
3: **for** $j = 1 \dots c$ **do**
4: $z \leftarrow 0$ ▷ Workspace and internal state of simulated machine
5: Non-deterministically guess $L \leq i-1$ ▷ $(*)$
6: Simulate $\mathcal{M}(x,w)$ using z as workspace for L steps ▷ $(*)$
7: **if** $h_k(z,w) \leq h'$ **then** ▷ Visited the nodes in wrong order
8: Simulate $\mathcal{M}(x,w)$ using z and reach any halting state
9: **return** FALSE
10: **end if**
11: $h' \leftarrow h_k(z,w)$
12: **while** there are unvisited neighbours **do**
13: Step $\mathcal{M}(x,w)$ with workspace z into a neighbour configuration
14: **if** $h_k(z,w') = v$ **then** ▷ v is reachable in i steps
15: Simulate $\mathcal{M}(x,w)$ using z and reach any halting state
16: **return** FALSE
17: **end if**
18: Revert simulation with one step back
19: **end while**
20: Continue simulation of $\mathcal{M}(x,w)$ using z and reach any halting state
21: **end for**
22: **return** TRUE
23: **end procedure**

Behavior of the cannotReach subroutine. If the hash v is unreachable within i steps and the given c, h_k obey the promise, then some non-deterministic guess will cause the procedure to return TRUE. If v *is* reachable and c, h_k obey the promise, every guess will return FALSE. *Furthermore*, if the hash v is unreachable within i steps, and c is smaller than the number of nodes in $\mathcal{G}_{\mathcal{M},x,w}$ that are reachable within $i-1$ steps, then there is a non-deterministic guess that causes the procedure to return TRUE, even if h_k is not one-to-one.

The CANNOTREACH subroutine visits c different nodes of $\mathcal{G}_{\mathcal{M},x,w}$ in order of ascending hash value, and for each of them checks that none of their neighbors hash to v. Since a single step of a computation only makes a local change, it is possible to remember this step and revert it afterward to continue with the next neighbor. If one of the neighbors hash to v or if a wrong non-deterministic guess has been made somewhere, we restore the aux-tape and return FALSE. Otherwise finish the simulation of $\mathcal{M}$ until a halting configuration is reached, to restore the orginal value of w. If we have visited c distinct nodes without finding v as a neighbor, then we return TRUE.

Property II follows easily from the correctness of the CANNOTREACH subroutine: indeed, if $M(x,w)$ rejects, then acc_w is not reachable, and hence with the promise made on g and h_ℓ, some guess will cause CANNOTREACH$(h_\ell(acc_w), M+1, g, h_\ell)$ to return TRUE.

We now complete the proof of the theorem by proving Property I. Let us focus on the k-loop (lines 7–26) which goes through every hash function h_k. For each h_k a value c is computed (see lines 8, 10, 13 and 20).

It might happen that the k-loop is aborted (in line 17), but if this never happens, then c will be compared to g (line 22), so that by the time the k-loop terminates, g will hold the maximum c produced for any value of k (line 23), and ℓ will hold the first value of k which produced this maximum (line 24).

Now we make the following two claims:

(i) If s is good, and h_k is one-to-one on $\mathcal{G}_{\mathcal{M},x,w}$, the i-loop (lines 9–21) will either abort, or set $c = |\mathcal{G}_{\mathcal{M},x,w}|$. Furthermore, some non-deterministic choice within the i-loop will not abort.

(ii) If s is good, but h_k is not one-to-one on $\mathcal{G}_{\mathcal{M},x,w}$, the i-loop will either abort, or set c to a value strictly smaller than $|\mathcal{G}_{\mathcal{M},x,w}|$. As above, some non-deterministic choice within the i-loop will not abort.

From these, it follows that if s is good, then for every k there is a non-deterministic guess which does not abort, and using any such non-aborting guess, g will be set to $|\mathcal{G}_{\mathcal{M},x,w}|$, and ℓ will be the smallest k for which h_k is one-to-one. This gives us Property I.

Let us prove claim (i). Suppose that h_k is one-to-one, and that the i-loop does not abort. Then we may prove inductively that in every iteration of the i-loop, c is the number of nodes in $\mathcal{G}_{\mathcal{M},x,w}$ reachable by $M(x,w)$ within $i-1$ steps. Now, c, h_k satisfy the promise required by CANNOTREACH, and hence, for any non-aborting guess, the v-loop will set c' to the number of nodes in $\mathcal{G}_{\mathcal{M},x,w}$ reachable within i steps; this value is then copied to c (line 20) for the next iteration of the i-loop. When the i-loop ends, c has been set to the number of nodes reachable within M steps, which is exactly $|\mathcal{G}_{\mathcal{M},x,w}|$. The fact that there always exists such a non-aborting guess follows from the behavior of the CANREACH procedure, and from the behavior of the CANNOTREACH procedure in the case when c, h_k fulfill the promise.

To prove claim (ii), notice that the value of c' is incremented in line 13, and is thus bounded by the the size of image $h_k(\mathcal{G}_{\mathcal{M},x,w})$. So if h_k is not one-to-one, c' will always be strictly less than $|\mathcal{G}_{\mathcal{M},x,w}|$. On the other hand, it is always possible to find a non-deterministic guess which does not abort, even when h_k is not one-to-one. Whenever hash v is reachable in i steps, we can take the guess which makes CANREACH in line 12 return TRUE; when hash v is not reachable in i steps, we know from the behavior of CANNOTREACH, that we can find a guess that makes CANNOTREACH return true, provided that the argument c given to CANNOTREACH in iteration i is not more than the number of nodes reachable within $i-1$ steps. This follows from the fact that, in iteration $i-1$, c' is bounded by the number of such nodes (because it is incremented only conditional on CANREACH of line 12. ◀

5 Hierarchies for Catalytic Computation

In this section we prove space-hierarchy theorems for deterministic and non-deterministic catalytic computation. Hierarchy theorems are usually proven using diagonalization. Since catalytic computation is a semantic model we do not know how to use diagonalization directly. Similarly to other semantic models (such as bounded-error randomized computation) we have to settle for hierarchy theorems with advice. This advice is used to tell the diagonalizing machine which machines can be safely simulated and diagonalized against, and which should not be simulated (so that the diagonalizing machine remains in the model).

The hierarchy theorem can be proven using the technique of Van Melkebeek and Pervyshev [10], which are sophisticated variations of [11]. Separations for certain space bounds follow directly from previous results on generic hierarchy theorems for semantic models of computation [7, 10]. We omit the proofs due to space constraints.

▶ **Theorem 12.** *Let $a \geq 1$ be an integer and $s'(n)$ and $s(n)$ be space-constructible functions. There is a function in* $\mathsf{CNSPACE}(s(n))/1$ *that is not in* $\mathsf{CNSPACE}(s'(n))/a$*, and there is a function in* $\mathsf{CSPACE}(s(n))/1$ *that is not in* $\mathsf{CSPACE}(s'(n))/a$ *if any of the following is satisfied:*

1. $s'(n) = O(\log n)$ *and* $s(n) = \omega(\log n)$.
2. $s'(n) = O(\log^{k'} n)$ *and* $s(n) = \Omega(2^{(\log\log n)^{k'}})$*, for some constant* $k' > 1$.
3. $s'(n) = O(n^{k'})$ *and* $s(n) = \Omega(n^k)$*, where* $0 < k' < k/2$ *and* $k' < 1/(1+a)$.
4. $s'(n) = O(n^{k'})$ *and* $s(n) = \Omega(n^k)$*, where* $k, k' > 0$ *are such that* $k \geq 2a$ *and* $k \geq \lceil 4^{ak'^2} \rceil$.

The following corollary follows by using a padding argument (see [10], §4.4).

▶ **Corollary 13.** *Let $a \geq 1$ be an integer and $k > k'$ be positive reals. Then there is a function in $CNSPACE(n^k)/a$ that is not in $CNSPACE(n^{k'})/a$.*

References

1 E. Allender, H. Buhrman, M. Koucký, D. van Melkebeek, and D. Ronneburger. Power from random strings. *SIAM J. Comput.*, 35(6):1467–1493, June 2006. doi:10.1137/050628994.

2 S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

3 H. Buhrman, R. Cleve, M. Koucký, B. Loff, and F. Speelman. Computing with a full memory: Catalytic space. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC'14, pages 857–866, New York, NY, USA, 2014. ACM. doi:10.1145/2591796.2591874.

4 N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988. doi:10.1137/0217058.

5 R. Impagliazzo and A. Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC'97, pages 220–229, New York, NY, USA, 1997. ACM. doi:10.1145/258533.258590.

6 R. Karp and R. Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28:191–209, 1982.

7 J. Kinne and D. van Melkebeek. Space hierarchy results for randomized and other semantic models. *Computational Complexity*, 19(3):423–475, 2010. doi:10.1007/s00037-009-0277-1.

8 A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002. doi:10.1137/S0097539700389652.

9 R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988. doi:10.1007/BF00299636.

10 D. van Melkebeek and K. Pervyshev. A generic time hierarchy with one bit of advice. *Computational Complexity*, 16(2):139–179, 2007. doi:10.1109/CCC.2006.7.

11 S. Žák. A Turing machine time hierarchy. *Theoretical Computer Science*, 26(3):327–333, 1983. doi:10.1016/0304-3975(83)90015-4.

Testing Shape Restrictions of Discrete Distributions*

Clément L. Canonne[1], Ilias Diakonikolas[2], Themis Gouleakis[3], and Ronitt Rubinfeld[4]

1 **Columbia University, 500 W 120th Street, New York NY, USA**
ccanonne@cs.columbia.edu
2 **University of Edinburgh, 10 Crichton Street, Edinburgh, Scotland, UK**
ilias.d@ed.ac.uk
3 **CSAIL, MIT, 32 Vassar Street, Cambridge MA, USA**
tgoule@mit.edu
4 **CSAIL, MIT, 32 Vassar Street, Cambridge MA, USA; and**
the Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv, Israel
ronitt@csail.mit.edu

Abstract

We study the question of testing *structured* properties (classes) of discrete distributions. Specifically, given sample access to an arbitrary distribution D over $[n]$ and a property $\mathcal{P}$, the goal is to distinguish between $D \in \mathcal{P}$ and $\ell_1(D, \mathcal{P}) > \varepsilon$. We develop a general algorithm for this question, which applies to a large range of "shape-constrained" properties, including monotone, log-concave, t-modal, piecewise-polynomial, and Poisson Binomial distributions. Moreover, for all cases considered, our algorithm has near-optimal sample complexity with regard to the domain size and is computationally efficient. For most of these classes, we provide the first non-trivial tester in the literature. In addition, we also describe a generic method to prove lower bounds for this problem, and use it to show our upper bounds are nearly tight. Finally, we extend some of our techniques to tolerant testing, deriving nearly–tight upper and lower bounds for the corresponding questions.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.3 Probability and Statistics

Keywords and phrases property testing, probability distributions, statistics, lower bounds

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.25

1 Introduction

Inferring information about the probability distribution that underlies a data sample is an essential question in Statistics, and one that has ramifications in every field of the natural sciences and quantitative research. In many situations, it is natural to assume that this data exhibits some simple structure because of known properties of the origin of the data, and in fact these assumptions are crucial in making the problem tractable. Such assumptions translate as constraints on the probability distribution – e.g., it is supposed to be Gaussian, or to meet a smoothness or "fat tail" condition (see e.g. [26, 24, 34]).

As a result, the problem of deciding whether a distribution possesses such a structural property has been widely investigated both in theory and practice, in the context of *shape*

* The full version of this paper is available at http://arxiv.org/abs/1507.03558.

© Clément L. Canonne, Ilias Diakonikolas, Themis Gouleakis, and Ronitt Rubinfeld;
licensed under Creative Commons License CC-BY

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 25; pp. 25:1–25:14

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

restricted inference [7, 33] and *model selection* [27]. Here, it is guaranteed or thought that the unknown distribution satisfies a shape constraint, such as having a monotone or log-concave probability density function [32, 6, 39, 19]. From a different perspective, a recent line of work in Theoretical Computer Science, originating from the papers of Batu et al. [10, 9, 22] has also been tackling similar questions in the setting of property testing (see [29, 30, 31, 12] for surveys on this field). This very active area has seen a spate of results and breakthroughs over the past decade, culminating in very efficient (both sample and time-wise) algorithms for a wide range of distribution testing problems [8, 23, 4, 18, 16, 1, 21]. In many cases, this led to a tight characterization of the number of samples required for these tasks as well as the development of new tools and techniques, drawing connections to learning and information theory [35, 36, 37].

In this paper, we focus on the following property testing problem: given a class (property) of distributions $\mathcal{P}$ and sample access to an *arbitrary* distribution D, one must distinguish between the case that (a) $D \in \mathcal{P}$, versus (b) $\|D - D'\|_1 > \varepsilon$ for all $D' \in \mathcal{P}$ (i.e., D is either in the class, or far from it). While many of the previous works have focused on the testing of specific properties of distributions or obtained algorithms and lower bounds on a case-by-case basis, an emerging trend in distribution testing is to design general frameworks that can be applied to *several* property testing problems [38, 21]. This direction, the testing analog of a similar movement in distribution learning [13, 15, 14, 3], aims at abstracting the minimal assumptions that are shared by a large variety of problems, and giving algorithms that can be used for any of these problems. In this work, we make significant progress in this direction by providing a unified framework for the question of testing various properties of probability distributions. More specifically, we describe a generic technique to obtain upper bounds on the sample complexity of this question, which applies to a broad range of structured classes. Our technique yields sample near-optimal and computationally efficient testers for a wide range of distribution families. Conversely, we also develop an approach – generic as well – to prove lower bounds on these sample complexities, and use it to derive tight or nearly tight bounds for many of these classes.

Related work

Batu et al. [11] initiated the study of efficient property testers for monotonicity and obtained (nearly) matching upper and lower bounds for this problem; while [1] later considered testing the class of Poisson Binomial Distributions, and settled the sample complexity of this problem (up to the precise dependence on ε). Indyk, Levi, and Rubinfeld [25], focusing on distributions that are piecewise constant on t intervals ("t-histograms") described a $\tilde{O}(\sqrt{tn}/\varepsilon^5)$-sample algorithm for testing membership to this class. Another body of work by [8], [11], and [18] shows how assumptions on the shape of the distributions can lead to significantly more efficient algorithms. They describe such improvements in the case of identity and closeness testing as well as for entropy estimation, under monotonicity or k-modality constraints. Specifically, Batu et al. show in [11] how to obtain a $O\big(\log^3 n/\varepsilon^3\big)$-sample tester for closeness in this setting, in stark contrast to the $\Omega\big(n^{2/3}\big)$ general lower bound. Daskalakis et al. [18] later gave $O(\sqrt{\log n})$ and $O(\log^{2/3} n)$-sample testing algorithms for testing respectively identity and closeness of monotone distributions, and obtained similar results for k-modal distributions. Finally, we briefly mention two related results, due respectively to [8] and [17]. The first one states that for the task of getting a multiplicative *estimate* of the entropy of a distribution, assuming monotonicity enables exponential savings in sample complexity – $O\big(\log^6 n\big)$, instead of $\Omega(n^c)$ for the general case. The second describes how to test if an unknown k-modal distribution is in fact monotone, using only $O\big(k/\varepsilon^2\big)$ samples. Note that the latter line of

work differs from ours in that it *presupposes* the distributions satisfy some structural property, and uses this knowledge to test something else about the distribution; while we are given *a priori* arbitrary distributions, and must *check* whether the structural property holds. Except for these two cases of monotonicity and PBDs nothing was known on testing the properties we study.[1]

Moreover, for the specific problems of identity and closeness testing, recent results of [21, 20] describe a general algorithm which applies to a large range of shape or structural constraints, and yields optimal identity testers for classes of distributions that satisfy them. We observe that while the question they answer can be cast as a specialized instance of membership testing, our results are incomparable to theirs, both because of the distinction above (testing *with* versus testing *for* structure) and as the structural assumptions they rely on are fundamentally different from ours.

1.1 Results and techniques

A natural way to tackle our membership testing problem would be to first learn the unknown distribution D *as if* it satisfied the property, before checking if the hypothesis obtained is indeed both close to the original distribution and to the property. Taking advantage of the purported structure, the first step could presumably be conducted with a small number of samples; things break down, however, in the second step. Indeed, most approximation results leading to the improved learning algorithms one would apply in the first stage only provide very weak guarantees, in the ℓ_1 sense. For this reason, they lack the robustness that would be required for the second part, where it becomes necessary to perform *tolerant* testing between the hypothesis and D – a task that would then entail a number of samples almost linear in n. To overcome this difficulty, we need to move away from these global ℓ_1 closeness results and instead work with stronger requirements, this time in ℓ_2 norm.

At the core of our approach is an idea of Batu et al. [11], which show that monotone distributions can be well-approximated by piecewise constant densities on a suitable partition of the domain; and leverage this fact to reduce monotonicity testing to uniformity testing on each interval of this partition. While their argument is tailored specifically for the setting of monotonicity testing, we are able to abstract the key ingredients, and generalize this idea to many classes of probability distributions. In more detail, we provide a generic testing algorithm which applies indifferently to any class of distributions which admit succinct decompositions – that is, which are well-approximated (in a strong ℓ_2 sense) by piecewise constant densities on a small number of intervals (we hereafter refer to this approximation property, formally defined in Definition 15, as (Succinctness); and extend the notation to apply to any *class* $\mathcal{C}$ of distributions for which all $D \in \mathcal{C}$ satisfy (Succinctness)). Crucially, the algorithm does not care about *how* these decompositions can be obtained: for the purpose of testing these structural properties we only need to establish their *existence*. Specific examples are given in the corollaries below. Informally, our main result is as follows:

▶ **Theorem 1** (Main Theorem). *There exists an algorithm* TestSplittable *which, given sampling access to an unknown distribution D over $[n]$ and parameter $\varepsilon \in (0,1]$, can distinguish with probability $2/3$ between (a) $D \in \mathcal{P}$ versus (b) $\ell_1(D,\mathcal{P}) > \varepsilon$, for* any *property $\mathcal{P}$ that satisfies the above natural structural criterion* (Succinctness). *Moreover, for many*

[1] Following the communication of a preliminary version of this paper, Acharya, Daskalakis, and Kamath [2] obtained near-optimal testers for some of the classes we consider. Their work builds on ideas from [1] and, to the best of our knowledge, their techniques are orthogonal to ours.

Table 1 Summary of results.

Class	Upperbound	Lowerbound
Monotone	$\tilde{O}\left(\frac{\sqrt{n}}{\varepsilon^6}\right)$ [11], $\tilde{O}\left(\frac{\sqrt{n}}{\varepsilon^{7/2}}\right)$ (Corollary 2)	$\Omega\left(\frac{\sqrt{n}}{\varepsilon^2}\right)$ [11], $\Omega\left(\frac{\sqrt{n}}{\varepsilon^2}\right)$ (Corollary 6)
Unimodal	$\tilde{O}\left(\frac{\sqrt{n}}{\varepsilon^{7/2}}\right)$ (Corollary 2)	$\Omega\left(\frac{\sqrt{n}}{\varepsilon^2}\right)$ (Corollary 6)
t-modal	$\tilde{O}\left(\frac{\sqrt{tn}}{\varepsilon^{7/2}}\right)$ (Corollary 3)	$\Omega\left(\frac{\sqrt{n}}{\varepsilon^2}\right)$ (Corollary 6)
Log-concave, concave, convex	$\tilde{O}\left(\frac{\sqrt{n}}{\varepsilon^{7/2}}\right)$ (Corollary 2)	$\Omega\left(\frac{\sqrt{n}}{\varepsilon^2}\right)$ (Corollary 6)
Monotone Hazard Rate (MHR)	$\tilde{O}\left(\frac{\sqrt{n}}{\varepsilon^{7/2}}\right)$ (Corollary 2)	$\Omega\left(\frac{\sqrt{n}}{\varepsilon^2}\right)$ (Corollary 6)
Binomial, Poisson Binomial (PBD)	$\tilde{O}\left(\frac{n^{1/4}}{\varepsilon^2}+\frac{1}{\varepsilon^6}\right)$ [1], $\tilde{O}\left(\frac{n^{1/4}}{\varepsilon^{7/2}}\right)$ (Corollary 5)	$\Omega\left(\frac{n^{1/4}}{\varepsilon^2}\right)$ ([1], Corollary 7)
t-histograms	$\tilde{O}\left(\frac{\sqrt{tn}}{\varepsilon^5}\right)$ [25], $\tilde{O}\left(\frac{\sqrt{tn}}{\varepsilon^3}\right)$ (Corollary 4)	$\Omega\left(\sqrt{tn}\right)$ for $t \leq \frac{1}{\varepsilon}$ [25], $\Omega\left(\frac{\sqrt{n}}{\varepsilon^2}\right)$ (Corollary 6)
t-piecewise degree-d	$\tilde{O}\left(\frac{\sqrt{t(d+1)n}}{\varepsilon^{7/2}}+\frac{t(d+1)}{\varepsilon^3}\right)$ (Corollary 4)	$\Omega\left(\frac{\sqrt{n}}{\varepsilon^2}\right)$ (Corollary 6)
k-SIIRV		$\Omega\left(k^{1/2}n^{1/4}\right)$ (Corollary 8)

such properties this algorithm is computationally efficient, and its sample complexity is optimal (up to logarithmic factors and the exact dependence on ε).

We then instantiate this result to obtain "out-of-the-box" *computationally efficient* testers for several classes of distributions, by showing that they satisfy the premise of our theorem (the definition of these classes is given in Section 2.1):

▶ **Corollary 2.** *The algorithm* TestSplittable *can test the classes of monotone, unimodal, log-concave, concave, convex, and monotone hazard rate (MHR) distributions, with $\tilde{O}\left(\sqrt{n}/\varepsilon^{7/2}\right)$ samples.*

▶ **Corollary 3.** *The algorithm* TestSplittable *can test the class of t-modal distributions, with $\tilde{O}\left(\sqrt{tn}/\varepsilon^{7/2}\right)$ samples.*

▶ **Corollary 4.** *The algorithm* TestSplittable *can test the classes of t-histograms and t-piecewise degree-d distributions, with $\tilde{O}\left(\sqrt{tn}/\varepsilon^3\right)$ and $\tilde{O}\left(\sqrt{t(d+1)n}/\varepsilon^{7/2}+t(d+1)/\varepsilon^3\right)$ samples respectively.*

▶ **Corollary 5.** *The algorithm* TestSplittable *can test the classes of Binomial and Poisson Binomial Distributions, with $\tilde{O}\left(n^{1/4}/\varepsilon^{7/2}\right)$ samples.*

We stress that prior to our work, no non-trivial testing bound was known for many of these classes – including t-modal, log-concave, convex, concave, MHR, and piecewise polynomials. Moreover, although three of these results are not new (the $\tilde{O}\left(\sqrt{n}/\varepsilon^6\right)$ upper and $\Omega\left(\sqrt{n}/\varepsilon^2\right)$ lower bounds on testing monotonicity can be found in [11], while the $\Theta\left(n^{1/4}\right)$ sample complexity of testing PBDs is pinned down[2] in [1] and the task of testing t-histograms is considered in [25]), the crux here is that we are able to derive them in a *unified* way, by applying the same generic algorithm to these different questions. Our result for t-histograms (Theorem 4) also improves on the previous $\tilde{O}\left(\sqrt{tn}/\varepsilon^5\right)$-sample tester, as long as $t = \tilde{O}\left(1/\varepsilon^2\right)$. As an addition to its generality, we emphasize that our framework yields a much cleaner and simpler proof of the results from [1], for both the upper and lower bounds.

[2] Specifically, [1] obtain an $n^{1/4} \cdot \tilde{O}\left(1/\varepsilon^2\right) + \tilde{O}\left(1/\varepsilon^6\right)$ sample complexity, to be compared with our $\tilde{O}\left(n^{1/4}/\varepsilon^{7/2}\right) + O\left(\log^4 n/\varepsilon^4\right)$ upper bound; as well as an $\Omega\left(n^{1/4}/\varepsilon^2\right)$ lower bound.

Lower bounds

As a counterpart to our testing algorithm, we give a generic framework for proving lower bounds against testing classes of distributions. In more detail, we describe how to reduce – under a mild assumption on the property $\mathcal{C}$ – the problem of testing *membership to* $\mathcal{C}$ ("does $D \in \mathcal{C}$?") to testing *identity to* D^* ("does $D = D^*$?"), for any explicit distribution D^* in $\mathcal{C}$. While these two problems need not in general be related,[3] we show that our approach applies to a large number of natural properties and obtain nearly matching lower bounds for all of them. Moreover, this lets us derive a simple proof of the lower bound of [1] on testing the class of Poisson Binomial Distributions.

▶ **Corollary 6.** *Testing log-concavity, convexity, concavity, MHR, unimodality, t-modality, t-histograms, and t-piecewise degree-d distributions each require* $\Omega\big(\sqrt{n}/\varepsilon^2\big)$ *samples (the last three for* $t = o(\sqrt{n})$ *and* $t(d+1) = o(\sqrt{n})$*, respectively), for any* $\varepsilon \geq 1/n^{O(1)}$.

▶ **Corollary 7.** *Testing the classes of Binomial and Poisson Binomial Distributions each require* $\Omega\big(n^{1/4}/\varepsilon^2\big)$ *samples, for any* $\varepsilon \geq 1/n^{O(1)}$.

▶ **Corollary 8.** *There exist absolute constants* $c > 0$ *and* $\varepsilon_0 > 0$ *such that testing the class of k-SIIRV distributions requires* $\Omega\big(k^{1/2}n^{1/4}\big)$ *samples, for any* $k = o(n^c)$ *and* $\varepsilon \leq \varepsilon_0$.

Tolerant testing upper and lower bounds

Using our techniques, we establish the first non-trivial upper and lower bounds on tolerant testing for shape restrictions. Similarly, our upper and lower bounds are matching as a function of the domain size. More specifically, we give a generic upper bound approach (which is a simple variant of our non-tolerant testing upper bound approach). Our lower bound approach for the non-tolerant case applies to *tolerant* testing as well. In more detail, our results are as follows (see Sections 6 and 7 of the full version):

▶ **Corollary 9.** *Tolerant testing of log-concavity, convexity, concavity, MHR, unimodality, and t-modality can be performed with* $O\big(\frac{1}{(\varepsilon_2-\varepsilon_1)^2}\frac{n}{\log n}\big)$ *samples, for* $\varepsilon_2 \geq C\varepsilon_1$ *(where* $C > 2$ *is a constant).*

▶ **Corollary 10.** *Tolerant testing of the classes of Binomial and Poisson Binomial Distributions can be performed with* $O\big(\frac{1}{(\varepsilon_2-\varepsilon_1)^2}\frac{\sqrt{n\log(1/\varepsilon_1)}}{\log n}\big)$ *samples, for* $\varepsilon_2 \geq C\varepsilon_1$ *(where* $C > 2$ *is a constant).*

▶ **Corollary 11.** *Tolerant testing of log-concavity, convexity, concavity, MHR, unimodality, and t-modality each require* $\Omega\Big(\frac{1}{(\varepsilon_2-\varepsilon_1)}\frac{n}{\log n}\Big)$ *samples (the latter for* $t = o(n)$*).*

▶ **Corollary 12.** *Tolerant testing of the classes of Binomial and Poisson Binomial Distributions each require* $\Omega\Big(\frac{1}{(\varepsilon_2-\varepsilon_1)}\frac{\sqrt{n}}{\log n}\Big)$ *samples.*

[3] As a simple example, consider the class $\mathcal{C}$ of *all* distributions, for which testing membership is trivial.

[3] *Tolerant testing* of a property $\mathcal{P}$ is defined as follows: given $0 \leq \varepsilon_1 < \varepsilon_2 \leq 1$, one must distinguish between (a) $\ell_1(D,\mathcal{P}) \leq \varepsilon_1$ and (b) $\ell_1(D,\mathcal{P}) \geq \varepsilon_2$. This turns out to be, in general, a much harder task than that of "regular" testing (where we take $\varepsilon_1 = 0$).

On the scope of our results

We observe that our main theorem is likely to apply to many other classes of distributions, due to the mild structural assumptions it requires. However, we did not attempt here to be comprehensive; but rather to illustrate the generality of our results. Moreover, for all properties considered in this paper the generic upper and lower bounds we derive through our methods turn out to be optimal up to polylogarithmic factors (with regard to the support size). The reader is referred to Table 1 for a summary of our results and related work.

1.2 Organization of the paper

We start by giving the necessary background and definitions in Section 2, before turning to our main result, the proof of Theorem 1 (our general testing algorithm) in Section 3, and sketching our lower bound framework in Section 4. Due to space constraints, we focus in this extended abstract on proving this main theorem; and defer its applications, as well as the other sections of the paper – details on lower bounds and tolerant testing, to the full version.

2 Notations and preliminaries

We first restate a result of Batu et al. relating closeness to uniformity in ℓ_2 and ℓ_1 norms to "overall flatness" of the probability mass function, and which will be one of the ingredients of the proof of Theorem 1:

▶ **Lemma 13** ([10, 9]). *Let D be a distribution on a domain S. (a) If $\max_{i\in S} D(i) \leq (1+\varepsilon)\min_{i\in S} D(i)$, then $\|D\|_2^2 \leq (1+\varepsilon^2)/|S|$. (b) If $\|D\|_2^2 \leq (1+\varepsilon^2)/|S|$, then $\|D-\mathcal{U}_S\|_1 \leq \varepsilon$.*

To check condition (b) above we shall rely on the following, which one can derive from the techniques in [21] and whose proof we defer to the full version:

▶ **Lemma 14** (Adapted from [21, Theorem 11]). *There exists an algorithm* Check-Small-ℓ_2 *which, given parameters $\varepsilon, \delta \in (0,1)$ and $c \cdot \sqrt{|I|}/\varepsilon^2 \log(1/\delta)$ independent samples from a distribution D over I (for some absolute constant $c > 0$), outputs either* yes *or* no*, and satisfies the following.*

- *If $\|D-\mathcal{U}_I\|_2 > \varepsilon/\sqrt{|I|}$, then the algorithm outputs* no *with probability at least $1-\delta$;*
- *If $\|D-\mathcal{U}_I\|_2 \leq \varepsilon/2\sqrt{|I|}$, then the algorithm outputs* yes *with probability at least $1-\delta$.*

2.1 Definitions

We give here the descriptions of the classes of distributions involved in this work (the formal definitions can be found in the full version). Recall that a distribution D over $[n]$ is *monotone* (non-increasing) if its probability mass function (pmf) satisfies $D(1) \geq D(2) \geq \dots D(n)$. A natural generalization of the class $\mathcal{M}$ of monotone distributions is the set of t-modal distributions, i.e. distributions whose pmf can go "up and down" or "down and up" up to t times.

We will also consider the classes of convex, concave, and log-concave (discrete) distributions, denoted respectively $\mathcal{K}^+, \mathcal{K}^-$, and $\mathcal{L}$ (a distribution is said to be log-concave if the logarithm of its pmf defines a concave function). It is not hard to see that convex and concave distributions are unimodal; moreover, every concave distribution is also log-concave, i.e. $\mathcal{K}^- \subseteq \mathcal{L}$.

Another class of interest is that of *monotone hazard rate* (MHR), which we write $\mathcal{MHR}$: this is the class of distributions D whose *hazard rate* $H(i) = D(i)/\sum_{j=i}^{n} D(j)$ is a non-decreasing function. It is known that every log-concave distribution is both unimodal and MHR (see e.g. [5, Proposition 10]), and that monotone distributions are MHR.

Two other families of distributions have elicited significant interest in the context of density estimation, that of *histograms* (piecewise constant) and *piecewise polynomial densities*. Finally, we will also consider the two following classes, which both extend the family of Binomial distributions $\mathcal{BIN}_n$: on one hand, the *Poisson Binomial Distributions* ($\mathcal{PBD}_n$), and on the other the *k-Sum of Independent Integer Random Variables (k-SIIRV)* (whose class we denote k-$\mathcal{SIIRV}_n$).

3 The general algorithm

In this section, we obtain our main result, restated below:

▶ **Theorem 1** (Main Theorem). *There exists an algorithm* TestSplittable *which, given sampling access to an unknown distribution D over $[n]$ and parameter $\varepsilon \in (0,1]$, can distinguish with probability $2/3$ between (a) $D \in \mathcal{P}$ versus (b) $\ell_1(D,\mathcal{P}) > \varepsilon$, for* any *property $\mathcal{P}$ that satisfies the above natural structural criterion* (Succinctness). *Moreover, for many such properties this algorithm is computationally efficient, and its sample complexity is optimal (up to logarithmic factors and the exact dependence on ε).*

Intuition. Before diving into the proof of this theorem, we first provide a high-level description of the argument. The algorithm proceeds in 3 stages: the first, the *decomposition step*, attempts to recursively construct a partition of the domain in a small number of intervals, with a very strong guarantee. If the decomposition succeeds, then the unknown distribution D will be close (in ℓ_1 distance) to its "flattening" on the partition; while if it fails (too many intervals have to be created), this serves as evidence that D does not belong to the class and we can reject. The second stage, the *approximation step*, then learns this flattening of the distribution – which can be done with few samples since by construction we do not have many intervals. The last stage is purely computational, the *projection step*: where we verify that the flattening we have learned is indeed close to the class $\mathcal{C}$. If all three stages succeed, then by the triangle inequality it must be the case that D is close to $\mathcal{C}$; and by the structural assumption on the class, if $D \in \mathcal{C}$ then it will admit succinct enough partitions, and all three stages will go through.

Turning to the proof, we start by defining formally the "structural criterion" we shall rely on, before describing the algorithm at the heart of our result in Section 3.1. (We note that a modification of this algorithm is described in the full version, which allows us to derive Corollary 5.)

▶ **Definition 15** (Decompositions). Let $\gamma > 0$ and $L = L(\gamma, n) \geq 1$. A class of distributions $\mathcal{C}$ on $[n]$ is said to be (γ, L)-*decomposable* if for every $D \in \mathcal{C}$ there exists $\ell \leq L$ and a partition $\mathcal{I}(\gamma, D) = (I_1, \ldots, I_\ell)$ of the interval $[1, n]$ such that, for all $j \in [\ell]$, one of the following holds:

(i) $D(I_j) \leq \frac{\gamma}{L}$; or
(ii) $\max_{i \in I_j} D(i) \leq (1+\gamma) \cdot \min_{i \in I_j} D(i)$.

Further, if $\mathcal{I}(\gamma, D)$ is *dyadic* (i.e., each I_k is of the form $[j \cdot 2^i + 1, (j+1) \cdot 2^i]$ for some integers i, j, corresponding to the leaves of a recursive bisection of $[n]$), then $\mathcal{C}$ is said to be (γ, L)-*splittable*.

▶ **Lemma 16.** *If $\mathcal{C}$ is (γ, L)-decomposable, then it is $(\gamma, O(L \log n))$-splittable.*

3.1 The algorithm

Theorem 1, and with it Corollary 2 and Corollary 3 will follow from the theorem below, combined with the structural theorems (left to the full version) on the corresponding classes:

▶ **Theorem 17.** *Let $\mathcal{C}$ be a class of distributions over $[n]$ for which the following holds.*

1. *$\mathcal{C}$ is $(\gamma, L(\gamma, n))$-splittable;*
2. *there exists a procedure* ProjectionDist$_{\mathcal{C}}$ *which, given as input a parameter $\alpha \in (0,1)$ and the explicit description of a distribution D over $[n]$, returns* yes *if the distance $\ell_1(D, \mathcal{C})$ to $\mathcal{C}$ is at most $\alpha/10$, and* no *if $\ell_1(D, \mathcal{C}) \geq 9\alpha/10$ (and either* yes *or* no *otherwise).*

Then, the algorithm TestSplittable *(Algorithm 1) is a $O\Big(\max\Big(\sqrt{nL}\log n/\varepsilon^3, L/\varepsilon^2\Big)\Big)$-sample tester for $\mathcal{C}$, for $L = L(\varepsilon, n)$. (Moreover, if* ProjectionDist$_{\mathcal{C}}$ *is computationally efficient, then so is* TestSplittable.*)*

For all classes we consider, we are able to provide such computationally efficient procedures (see the full version for details).

Algorithm 1 TestSplittable

Require: Domain I (interval), sample access to D over I; subroutine ProjectionDist$_{\mathcal{C}}$
Input: Parameters ε and function $L_{\mathcal{C}}(\cdot,\cdot)$.

Setting Up
  Define $\gamma \stackrel{\text{def}}{=} \frac{\varepsilon}{80}$, $L \stackrel{\text{def}}{=} L_{\mathcal{C}}(\gamma, |I|)$, $\kappa \stackrel{\text{def}}{=} \frac{\varepsilon}{160L}$, $\delta \stackrel{\text{def}}{=} \frac{1}{10L}$; and $c > 0$ be as in Lemma 14.
  Set $m \stackrel{\text{def}}{=} C \cdot \max\left(\frac{1}{\kappa}, \frac{\sqrt{L|I|}}{\varepsilon^3}\right) \cdot \log|I| = \tilde{O}\left(\frac{\sqrt{L|I|}}{\varepsilon^3} + \frac{L}{\varepsilon}\right)$ ▷ C is an absolute constant.
  Obtain a sequence $\mathbf{s}$ of m independent samples from D. ▷ For any $J \subseteq I$, let m_J be the number of samples falling in J.

Decomposition
  while $m_I \geq \max\left(c \cdot \frac{\sqrt{|I|}}{\varepsilon^2}\log\frac{1}{\delta}, \kappa m\right)$ and at most L splits have been performed **do**
    Run Check-Small-ℓ_2 (from Lemma 14) with parameters $\frac{\varepsilon}{40}$ and δ, using the samples of $\mathbf{s}$ belonging to I.
    if Check-Small-ℓ_2 outputs no **then**
      Bisect I, and recurse on both halves (using the same samples).
    end if
  end while
  if more than L splits have been performed **then**
    return REJECT
  else
    Let $\mathcal{I} \stackrel{\text{def}}{=} (I_1, \ldots, I_\ell)$ be the partition of $[n]$ from the leaves of the recursion. ▷ $\ell \leq L$.
  end if

Approximation
  Learn the flattening $\Phi(D, \mathcal{I})$ of D to ℓ_1 error $\frac{\varepsilon}{20}$ (with probability 1/10), using $O\big(\ell/\varepsilon^2\big)$ new samples. Let $\tilde{D}$ be the resulting hypothesis. ▷ $\tilde{D}$ is a ℓ-histogram.

Offline Check
  return ACCEPT if and only if ProjectionDist$_{\mathcal{C}}(\varepsilon, \tilde{D})$ returns yes. ▷ No sample needed.

3.2 Proof of Theorem 17

We now give the proof of our main result (Theorem 17), first analyzing the sample complexity of Algorithm 1 before arguing its correctness. For the latter, we will need the following simple lemma from [25], restated below:

▶ **Fact 18** ([25, Fact 1]). *Let D be a distribution over $[n]$, and $\delta \in (0,1]$. Given $m \geq C \cdot \frac{\log \frac{n}{\delta}}{\eta}$ independent samples from D (for some absolute constant $C > 0$), with probability at least $1-\delta$ we have that, for every interval $I \subseteq [n]$:*
(i) *if $D(I) \geq \frac{\eta}{4}$, then $\frac{D(I)}{2} \leq \frac{m_I}{m} \leq \frac{3D(I)}{2}$;*
(ii) *if $\frac{m_I}{m} \geq \frac{\eta}{2}$, then $D(I) > \frac{\eta}{4}$;*
(iii) *if $\frac{m_I}{m} < \frac{\eta}{2}$, then $D(I) < \eta$;*
where $m_I = |\{\, j \in [m] : x_j \in I \,\}|$ is the number of the samples falling into I.

Sample complexity

The sample complexity is immediate, and comes from Steps 4 and 20. The total number of samples is

$$m + O\left(\frac{\ell}{\varepsilon^2}\right) = O\left(\frac{\sqrt{|I| \cdot L}}{\varepsilon^3} \log |I| + \frac{L}{\varepsilon} \log |I| + \frac{L}{\varepsilon^2}\right) = O\left(\frac{\sqrt{|I| \cdot L}}{\varepsilon^3} \log |I| + \frac{L}{\varepsilon^2}\right).$$

Correctness

Say an interval I considered during the execution of the "Decomposition" step is *heavy* if m_I is big enough on Step 7, and *light* otherwise; and let $\mathcal{H}$ and $\mathcal{L}$ denote the sets of heavy and light intervals respectively. By choice of m and a union bound over all $|I|^2$ possible intervals, we can assume on one hand that with probability at least $9/10$ the guarantees of Fact 18 hold simultaneously for all intervals considered. We hereafter condition on this event.

We first argue that if the algorithm does not reject in Step 13, then with probability at least $9/10$ we have $\|D - \Phi(D,\mathcal{I})\|_1 \leq \varepsilon/20$. Indeed, we can write

$$\begin{aligned}\|D - \Phi(D,\mathcal{I})\|_1 &= \sum_{k\colon I_k \in \mathcal{L}} D(I_k) \cdot \|D_{I_k} - \mathcal{U}_{I_k}\|_1 + \sum_{k\colon I_k \in \mathcal{H}} D(I_k) \cdot \|D_{I_k} - \mathcal{U}_{I_k}\|_1 \\ &\leq 2 \sum_{k\colon I_k \in \mathcal{L}} D(I_k) + \sum_{k\colon I_k \in \mathcal{H}} D(I_k) \cdot \|D_{I_k} - \mathcal{U}_{I_k}\|_1 .\end{aligned}$$

Let us bound the two terms separately.

- If $I' \in \mathcal{H}$, then by our choice of threshold we can apply Lemma 14 with $\delta = \frac{1}{10L}$; conditioning on all of the (at most L) events happening, which overall fails with probability at most $1/10$ by a union bound, we get

$$\|D_{I'}\|_2^2 = \|D_{I'} - \mathcal{U}_{I'}\|_2^2 + \frac{1}{|I'|} \leq \left(1 + \frac{\varepsilon^2}{1600}\right) \frac{1}{|I'|}$$

  as Check-Small-ℓ_2 returned yes; and by Lemma 13 this implies $\|D_{I'} - \mathcal{U}_{I'}\|_1 \leq \varepsilon/40$.

- If $I' \in \mathcal{L}$, then we claim that $D(I') \leq \max(\kappa, 2c \cdot \frac{\sqrt{|I'|}}{m\varepsilon^2} \log \frac{1}{\delta})$. Clearly, this is true if $D(I') \leq \kappa$, so it only remains to show that $D(I') \leq 2c \cdot \frac{\sqrt{|I'|}}{m\varepsilon^2} \log \frac{1}{\delta}$. But this follows from Fact 18(i), as if we had $D(I') > 2c \cdot \frac{\sqrt{|I'|}}{m\varepsilon^2} \log \frac{1}{\delta}$ then $m_{I'}$ would have been big enough,

and $I' \notin \mathcal{L}$. Overall,

$$\sum_{I' \in \mathcal{L}} D(I') \leq \sum_{I' \in \mathcal{L}} \left(\kappa + 2c \cdot \frac{\sqrt{|I'|}}{m\varepsilon^2} \log \frac{1}{\delta} \right) \leq L\kappa + 2 \sum_{I' \in \mathcal{L}} c \cdot \frac{\sqrt{|I'|}}{m\varepsilon^2} \log \frac{1}{\delta} \leq \frac{\varepsilon}{160} \left(1 + \sum_{I' \in \mathcal{L}} \sqrt{\frac{|I'|}{|I|\, L}} \right) \leq \frac{\varepsilon}{80}$$

for a sufficiently big choice of constant $C > 0$ in the definition of m; where we first used that $|\mathcal{L}| \leq L$, and then that $\sum_{I' \in \mathcal{L}} \sqrt{\frac{|I'|}{|I|}} \leq \sqrt{L}$ by Jensen's inequality.

Putting it together, this yields

$$\|D - \Phi(D, \mathcal{I})\|_1 \leq 2 \cdot \frac{\varepsilon}{80} + \frac{\varepsilon}{40} \sum_{I' \in \mathcal{H}} D(I_k) \leq \varepsilon/40 + \varepsilon/40 = \varepsilon/20.$$

Soundness. By contrapositive, we argue that if the test returns ACCEPT, then (with probability at least 2/3) D is ε-close to $\mathcal{C}$. Indeed, conditioning on $\tilde{D}$ being $\varepsilon/20$-close to $\Phi(D, \mathcal{I})$, we get by the triangle inequality that

$$\begin{aligned} \|D - \mathcal{C}\|_1 &\leq \|D - \Phi(D, \mathcal{I})\|_1 + \|\Phi(D, \mathcal{I}) - \tilde{D}\|_1 + \operatorname{dist}\big(\tilde{D}, \mathcal{C}\big) \\ &\leq \frac{\varepsilon}{20} + \frac{\varepsilon}{20} + \frac{9\varepsilon}{10} = \varepsilon. \end{aligned}$$

Overall, this happens except with probability at most $1/10 + 1/10 + 1/10 < 1/3$.

Completeness. Assume $D \in \mathcal{C}$. Then the choice of of γ and L ensures the existence of a good dyadic partition $\mathcal{I}(\gamma, D)$ in the sense of Definition 15. For any I in this partition for which (i) holds ($D(I) \leq \frac{\gamma}{L} < \frac{\kappa}{2}$), I will have $\frac{m_I}{m} < \kappa$ and be kept as a "light leaf" (this by contrapositive of Fact 18(ii). For the other ones, (ii) holds: let I be one of these (at most L) intervals.

- If m_I is too small on Step 7, then I is kept as "light leaf."
- Otherwise, then by our choice of constants we can use Lemma 13 and apply Lemma 14 with $\delta = \frac{1}{10L}$; conditioning on all of the (at most L) events happening, which overall fails with probability at most 1/10 by a union bound, CHECK-SMALL-ℓ_2 will output yes, as

$$\|D_I - \mathcal{U}_I\|_2^2 = \|D_I\|_2^2 - \frac{1}{|I|} \leq \left(1 + \frac{\varepsilon^2}{6400} \right) \frac{1}{|I|} - \frac{1}{|I|} = \frac{\varepsilon^2}{6400\,|I|}$$

and I is kept as "flat leaf."

Therefore, as $\mathcal{I}(\gamma, D)$ is dyadic the DECOMPOSITION stage is guaranteed to stop within at most L splits (in the worst case, it goes on until $\mathcal{I}(\gamma, D)$ is considered, at which point it succeeds).[4] Thus Step 13 passes, and the algorithm reaches the APPROXIMATION stage. By the foregoing discussion, this implies $\Phi(D, \mathcal{I})$ is $\varepsilon/20$-close to D (and hence to $\mathcal{C}$); $\tilde{D}$ is then (except with probability at most 1/10) ($\frac{\varepsilon}{20} + \frac{\varepsilon}{20} = \frac{\varepsilon}{10}$)-close to $\mathcal{C}$, and the algorithm returns ACCEPT.

[4] In more detail, we want to argue that if D is in the class, then a decomposition with at most L pieces is found by the algorithm. Since there *is* a dyadic decomposition with at most L pieces (namely, $\mathcal{I}(\gamma, D) = (I_1, \ldots, I_t)$), it suffices to argue that the algorithm will never split one of the I_j's (as every single I_j will eventually be considered by the recursive binary splitting, unless the algorithm stopped recursing in this "path" before even considering I_j, which is even better). But this is the case by the above argument, which ensures each such I_j will be recognized as satisfying one of the two conditions for "good decomposition" (being either close to uniform in ℓ_2, or having very little mass).

4 Lower bounds

We now turn to proving converses to our positive results – namely, that many of the upper bounds we obtain cannot be significantly improved upon. As in our algorithmic approach, we describe for this purpose a *generic framework* for obtaining lower bounds: roughly speaking, a "testing-by-narrowing" reduction argument, that shows that under very modest assumptions testing *membership* to a class is at least as hard as testing *identity* to any fixed distribution it contains. (Due to space constraints, we only provide here a sketch of our results – and refer the interested reader to the full version.)

High-level idea

The motivation for our result is the observation of [11] that "monotonicity is at least as hard as uniformity." Unfortunately, their specific argument does not generalize to other classes of distributions, making it impossible to extend it readily. The starting point of our approach is to observe that while uniformity testing is hard in general, it becomes very easy *under the promise that the distribution is monotone, or even only close to monotone*. This gives an alternate proof of the lower bound for monotonicity testing, via a different reduction: first, test if the unknown distribution is monotone; if it is, test whether it is uniform, now assuming closeness to monotone.

More generally, this idea applies to any class $\mathcal{C}$ which (a) contains the uniform distribution, and (b) for which we have a $o(\sqrt{n})$-sample agnostic learner $\mathcal{L}$,[5] as follows. Assuming we have a tester $\mathcal{T}$ for $\mathcal{C}$ with sample complexity $o(\sqrt{n})$, define a uniformity tester as below.

- test if $D \in \mathcal{C}$ using $\mathcal{T}$; if not, reject (as $\mathcal{U} \in \mathcal{C}$, D cannot be uniform);
- otherwise, agnostically learn D with $\mathcal{L}$ (since D is close to $\mathcal{C}$), and obtain hypothesis $\hat{D}$;
- check offline if $\hat{D}$ is close to uniform.

By assumption, $\mathcal{T}$ and $\mathcal{L}$ each use $o(\sqrt{n})$ samples, so does the whole process; but this contradicts the lower bound of [10, 28] on uniformity testing. Hence, $\mathcal{T}$ must use $\Omega(\sqrt{n})$ samples.

This immediately will imply Corollary 6. Moreover, this argument can be further extended to other reductions than to uniformity, which we use to derive Corollaries 7 and 8.

The lower bound theorem

▶ **Theorem 19.** *Let $\mathcal{C}$ be a class of distributions over $[n]$ for which the following holds:*

(i) *there exists a semi-agnostic learner $\mathcal{L}$ for $\mathcal{C}$, with sample complexity $q_L(n, \varepsilon, \delta)$ and "agnostic constant" c;*

(ii) *there exists a subclass $\mathcal{C}_{\text{Hard}} \subseteq \mathcal{C}$ such that testing $\mathcal{C}_{\text{Hard}}$ requires $q_H(n, \varepsilon)$ samples.*

Suppose further that $q_L(n, \varepsilon, 1/10) = o(q_H(n, \varepsilon))$. Then, any tester for $\mathcal{C}$ must use $\Omega(q_H(n, \varepsilon))$ samples.

Proof. The above theorem relies on the reduction outlined above, which we rigorously detail here. Assuming $\mathcal{C}$, $\mathcal{C}_{\text{Hard}}$, $\mathcal{L}$ as above (with semi-agnostic constant $c \geq 1$), and a tester $\mathcal{T}$ for

[5] Recall that an algorithm is said to be a *semi-agnostic learner* for a class $\mathcal{C}$ if it satisfies the following. Given sample access to an arbitrary distribution D and parameter ε, it outputs a hypothesis $\hat{D}$ which (with high probability) does "almost as well as it gets": $\|D - \hat{D}\|_1 \leq c \cdot \text{OPT}_{\mathcal{C},D} + O(\varepsilon)$ where $\text{OPT}_{\mathcal{C},D} \stackrel{\text{def}}{=} \inf_{D' \in \mathcal{C}} \ell_1(D', D)$, and $c \geq 1$ is some absolute constant (if $c = 1$, the learner is said to be agnostic).

$\mathcal{C}$ with sample complexity $q_T(n,\varepsilon)$, we define a tester $\mathcal{T}_{\text{Hard}}$ for $\mathcal{C}_{\text{Hard}}$. On input $\varepsilon \in (0,1]$ and given sample access to a distribution D on $[n]$, $\mathcal{T}_{\text{Hard}}$ acts as follows:

1. call $\mathcal{T}$ with parameters n, $\frac{\varepsilon'}{c}$ (where $\varepsilon' \stackrel{\text{def}}{=} \frac{\varepsilon}{3}$) and failure probability 1/6, to $\frac{\varepsilon'}{c}$-test if $D \in \mathcal{C}$. If not, reject.
2. otherwise, agnostically learn a hypothesis $\hat{D}$ for D, with $\mathcal{L}$ called with parameters n, ε' and failure probability 1/6;
3. check offline if $\hat{D}$ is ε'-close to $\mathcal{C}_{\text{Hard}}$, accept if and only if this is the case.

We condition on both calls (to $\mathcal{T}$ and $\mathcal{L}$) to be successful, which overall happens with probability at least 2/3 by a union bound. The completeness is immediate: if $D \in \mathcal{C}_{\text{Hard}} \subseteq \mathcal{C}$, $\mathcal{T}$ accepts, and the hypothesis $\hat{D}$ satisfies $\|\hat{D} - D\|_1 \leq \varepsilon'$. Therefore, $\ell_1(\hat{D}, \mathcal{C}_{\text{Hard}}) \leq \varepsilon'$, and $\mathcal{T}_{\text{Hard}}$ accepts.

For the soundness, we proceed by contrapositive. Suppose $\mathcal{T}_{\text{Hard}}$ accepts; it means that each step was successful. In particular, $\ell_1(\hat{D}, \mathcal{C}) \leq \varepsilon'/c$; so that the hypothesis outputted by the agnostic learner satisfies $\|\hat{D} - D\|_1 \leq c \cdot \text{OPT} + \varepsilon' \leq 2\varepsilon'$. In turn, since the last step passed and by a triangle inequality we get, as claimed, $\ell_1(D, \mathcal{C}_{\text{Hard}}) \leq 2\varepsilon' + \ell_1(\hat{D}, \mathcal{C}_{\text{Hard}}) \leq 3\varepsilon' = \varepsilon$.

Observing that the overall sample complexity is $q_T(n, \frac{\varepsilon'}{c}) + q_L(n, \varepsilon', \frac{1}{10}) = q_T(n, \frac{\varepsilon'}{c}) + o(q_H(n, \varepsilon'))$ concludes the proof. ◀

References

1 Jayadev Acharya and Constantinos Daskalakis. Testing Poisson Binomial Distributions. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1829–1840, 2015.

2 Jayadev Acharya, Constantinos Daskalakis, and Gautam C. Kamath. Optimal Testing for Properties of Distributions. In C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, R. Garnett, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3577–3598. Curran Associates, Inc., 2015. URL: `http://papers.nips.cc/paper/5839-optimal-testing-for-properties-of-distributions.pdf`.

3 Jayadev Acharya, Ilias Diakonikolas, Jerry Zheng Li, and Ludwig Schmidt. Sample-optimal density estimation in nearly-linear time. *CoRR*, abs/1506.00671, 2015. URL: `http://arxiv.org/abs/1506.00671`.

4 Noga Alon, Alexandr Andoni, Tali Kaufman, Kevin Matulef, Ronitt Rubinfeld, and Ning Xie. Testing k-wise and almost k-wise independence. In *Proceedings of the 39th ACM Symposium on Theory of Computing, STOC 2007, San Diego, California, USA, June 11-13, 2007*, pages 496–505, New York, NY, USA, 2007. `doi:10.1145/1250790.1250863`.

5 Mark Y. An. Log-concave probability distributions: theory and statistical testing. Technical report, Centre for Labour Market and Social Research, Denmark, 1996. URL: `http://EconPapers.repec.org/RePEc:fth:clmsre:96-01`.

6 Mark Bagnoli and Ted Bergstrom. Log-concave probability and its applications. *Economic Theory*, 26(2):445–469, 2005. URL: `http://dx.doi.org/10.1007/s00199-004-0514-4`, `doi:10.1007/s00199-004-0514-4`.

7 Richard E. Barlow, Bartholomew D.J, J.M Bremner, and H.D Brunk. *Statistical Inference Under Order Restrictions: The Theory and Application of Isotonic Regression*. Wiley Series in Probability and Mathematical Statistics. J. Wiley, London, New York, 1972.

8 Tuğkan Batu, Sanjoy Dasgupta, Ravi Kumar, and Ronitt Rubinfeld. The complexity of approximating the entropy. *SIAM Journal on Computing*, 35(1):132–150, 2005.

9 Tuğkan Batu, Eldar Fischer, Lance Fortnow, Ravi Kumar, Ronitt Rubinfeld, and Patrick White. Testing random variables for independence and identity. In *42nd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2001, Las Vegas, Nevada, USA, October 14-17 2001*, pages 442–451, 2001.

10 Tuğkan Batu, Lance Fortnow, Ronitt Rubinfeld, Warren D. Smith, and Patrick White. Testing that distributions are close. In *41st Annual IEEE Symposium on Foundations of Computer Science, FOCS 2000, Redondo Beach, California, USA, November 12-14 2000*, pages 259–269, 2000.

11 Tuğkan Batu, Ravi Kumar, and Ronitt Rubinfeld. Sublinear algorithms for testing monotone and unimodal distributions. In *Proceedings of the 36th ACM Symposium on Theory of Computing, STOC 2004, Chicago, IL, USA, June 13-16, 2004*, pages 381–390, New York, NY, USA, 2004. ACM. doi:10.1145/1007352.1007414.

12 Clément L. Canonne. A Survey on Distribution Testing: your data is Big. But is it Blue? *Electronic Colloquium on Computational Complexity (ECCC)*, 22:63, April 2015.

13 Siu-on Chan, Ilias Diakonikolas, Rocco A. Servedio, and Xiaorui Sun. Learning mixtures of structured distributions over discrete domains. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1380–1394, 2013.

14 Siu-on Chan, Ilias Diakonikolas, Rocco A. Servedio, and Xiaorui Sun. Efficient density estimation via piecewise polynomial approximation. In *Proceedings of the 45th ACM Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 – June 03, 2014*, pages 604–613. ACM, 2014.

15 Siu-on Chan, Ilias Diakonikolas, Rocco A. Servedio, and Xiaorui Sun. Near-optimal density estimation in near-linear time using variable-width histograms. In *Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1844–1852, 2014.

16 Siu-on Chan, Ilias Diakonikolas, Gregory Valiant, and Paul Valiant. Optimal algorithms for testing closeness of discrete distributions. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1193–1203, 2014.

17 Constantinos Daskalakis, Ilias Diakonikolas, and Rocco A. Servedio. Learning k-modal distributions via testing. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1371–1385. Society for Industrial and Applied Mathematics (SIAM), 2012.

18 Constantinos Daskalakis, Ilias Diakonikolas, Rocco A. Servedio, Gregory Valiant, and Paul Valiant. Testing k-modal distributions: Optimal algorithms via reductions. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1833–1852. Society for Industrial and Applied Mathematics (SIAM), 2013. URL: http://dl.acm.org/citation.cfm?id=2627817.2627948.

19 Ilias Diakonikolas. Learning structured distributions. In *Handbook of Big Data*. CRC Press, 2016.

20 Ilias Diakonikolas, Daniel M. Kane, and Vladimir Nikishkin. Optimal algorithms and lower bounds for testing closeness of structured distributions. In *56th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2015*, 2015.

21 Ilias Diakonikolas, Daniel M. Kane, and Vladimir Nikishkin. Testing Identity of Structured Distributions. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, 2015.

22 Oded Goldreich and Dana Ron. On testing expansion in bounded-degree graphs. Technical Report TR00-020, Electronic Colloquium on Computational Complexity (ECCC), 2000.

23 Sudipto Guha, Andrew McGregor, and Suresh Venkatasubramanian. Streaming and sublinear approximation of entropy and information distances. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 733–742, Philadelphia, PA, USA, 2006. Society

for Industrial and Applied Mathematics (SIAM). URL: http://dl.acm.org/citation.cfm?id=1109557.1109637.

24 Philip Hougaard. Survival models for heterogeneous populations derived from stable distributions. *Biometrika*, 73:397–96, 1986.

25 Piotr Indyk, Reut Levi, and Ronitt Rubinfeld. Approximating and Testing k-Histogram Distributions in Sub-linear Time. In *Proceedings of PODS*, pages 15–22, 2012.

26 Benoit Mandelbrot. New methods in statistical economics. *Journal of Political Economy*, 71(5):pp. 421–440, 1963.

27 Pascal Massart and Jean Picard. *Concentration inequalities and model selection*. Lecture Notes in Mathematics. Springer, 33, 2003, Saint-Flour, Cantal, 2007. URL: http://opac.inria.fr/record=b1122538.

28 Liam Paninski. A coincidence-based test for uniformity given very sparsely sampled discrete data. *IEEE Transactions on Information Theory*, 54(10):4750–4755, 2008.

29 Dana Ron. Property Testing: A Learning Theory Perspective. *Foundations and Trends in Machine Learning*, 1(3):307–402, 2008.

30 Dana Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5:73–205, 2010.

31 Ronitt Rubinfeld. Taming Big Probability Distributions. *XRDS*, 19(1):24–28, September 2012. doi:10.1145/2331042.2331052.

32 Debasis Sengupta and Asok K. Nanda. Log-concave and concave distributions in reliability. *Naval Research Logistics (NRL)*, 46(4):419–433, 1999. doi:10.1002/(SICI)1520-6750(199906)46:4<419::AID-NAV5>3.0.CO;2-B.

33 Mervyn J. Silvapulle and Pranab K. Sen. *Constrained Statistical Inference*. John Wiley & Sons, Inc., 2001. doi:10.1002/9781118165614.fmatter.

34 Constantino Tsallis, Silvio V. F. Levy, André M. C. Souza, and Roger Maynard. Statistical-mechanical foundation of the ubiquity of Lévy distributions in nature. *Phys. Rev. Lett.*, 75:3589–3593, Nov 1995. doi:10.1103/PhysRevLett.75.3589.

35 Gregory Valiant and Paul Valiant. A CLT and tight lower bounds for estimating entropy. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:179, 2010.

36 Gregory Valiant and Paul Valiant. Estimating the unseen: An $n/\log n$-sample estimator for entropy and support size, shown optimal via new clts. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 685–694, 2011.

37 Gregory Valiant and Paul Valiant. An automatic inequality prover and instance optimal identity testing. In *55th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, 2014.

38 Paul Valiant. Testing symmetric properties of distributions. *SIAM Journal on Computing*, 40(6):1927–1968, 2011.

39 Guenther Walther. Inference and modeling with log-concave distributions. *Statistical Science*, 24(3):319–327, 2009. URL: http://projecteuclid.org/DPubS?service=UI&version=1.0&verb=Display&handle=euclid.ss/1270041258.

Deciding Circular-Arc Graph Isomorphism in Parameterized Logspace

Maurice Chandoo

Leibniz Universität Hannover, Theoretical Computer Science, Appelstr. 4, 30167 Hannover, Germany
chandoo@thi.uni-hannover.de

Abstract

We compute a canonical circular-arc representation for a given circular-arc (CA) graph which implies solving the isomorphism and recognition problem for this class. To accomplish this we split the class of CA graphs into uniform and non-uniform ones and employ a generalized version of the argument given by Köbler et al. (2013) that has been used to show that the subclass of Helly CA graphs can be canonized in logspace. For uniform CA graphs our approach works in logspace and in addition to that Helly CA graphs are a strict subset of uniform CA graphs. Thus our result is a generalization of the canonization result for Helly CA graphs. In the non-uniform case a specific set Ω of ambiguous vertices arises. By choosing the parameter k to be the cardinality of Ω this obstacle can be solved by brute force. This leads to an $\mathcal{O}(k + \log n)$ space algorithm to compute a canonical representation for non-uniform and therefore all CA graphs.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases graph isomorphism, canonical representation, parameterized algorithm

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.26

1 Introduction

An arc is a connected set of points on the circle. A graph G is called a CA graph if every vertex v can be assigned to an arc $\rho(v)$ such that two vertices u, v are adjacent iff their respective arcs intersect, i.e. $\rho(u) \cap \rho(v) \neq \emptyset$. We call such a (bijective) mapping ρ a CA representation of G and the set of arcs $\rho(G) = \{\rho(v) \mid v \in V(G)\}$ a CA model. The recognition problem for CA graphs is to decide whether a given graph G is a CA graph. The canonical representation problem for CA graphs consists of computing a CA representation ρ_G for a given CA graph G with the additional canonicity constraint that whenever two CA graphs G and H are isomorphic the CA models $\rho_G(G)$ and $\rho_H(H)$ are identical.

Similarly, a graph is an interval graph if every vertex can be assigned to an interval on a line such that two vertices share an edge iff their intervals intersect. It is easy to see that every interval graph is a CA graph since every interval model is a CA model.

The class of CA graphs started to gain attraction after a series of papers in the 1970's by Alan Tucker. However, there is still no known better upper bound for deciding CA graph isomorphism than for graph isomorphism in general even though considerable effort has been done to this end. There have been two claimed polynomial-time algorithms in [9], [3] which have been disproven in [2], [1] respectively. For the subclass of interval graphs a linear-time algorithm for isomorphism has been described in [7]. A series of newer results show that canonical representations for interval graphs, proper CA graphs and Helly CA graphs (a superset of interval graphs) can be computed in logspace [5, 4, 6]. Furthermore, recognition and isomorphism for interval graphs is logspace-hard[5] and these two hardness results carry over to the class of CA graphs; for recognition the reduction requires a little additional work.


© Maurice Chandoo;
licensed under Creative Commons License CC-BY

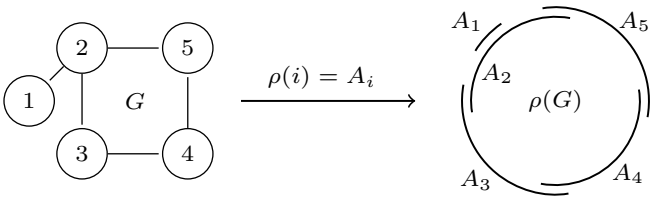


Figure 1 A CA graph and representation.

Our main contribution is that we extend the argument used in [6] to compute canonical representations for HCA graphs to all CA graphs. We split the class of CA graphs into uniform and non-uniform ones and show that the mentioned argument can be applied in both cases using only $\mathcal{O}(k + \log n)$ space. The parameter k describes the cardinality of an obstacle set Ω that occurs only in the non-uniform case. This means $k = 0$ in the uniform case and hence we also obtain a logspace algorithm for uniform CA graphs which are a superclass of HCA graphs. To the best of our knowledge this is the first non-trivial algorithm to decide isomorphism specifically for the class of CA graphs.

This paper is structured as follows. In section 2 we define CA graphs along with their representations and recall the concept of normalized representations from [3]. In section 3 we explain what we mean by flip trick and formalize this with the notions of flip sets and candidate functions. This idea has been used by [8] to compute a CA representation for CA graphs in linear time and [6] has modified it to compute canonical (Helly) CA representations for HCA graphs. In section 4 and 5 the flip trick is applied to uniform and non-uniform CA graphs respectively.

2 Preliminaries

Given two sets A, B we say A and B intersect if $A \cap B \neq \emptyset$. We say A, B overlap, in symbols $A \between B$, if $A \cap B, A \setminus B$ and $B \setminus A$ are non-empty. Let $A = (a_{u,v})_{u,v \in V(A)}, B = (b_{u,v})_{u,v \in V(B)}$ be two square matrices over vertex sets $V(A), V(B)$. We say A and B are isomorphic, in symbols $A \cong B$, if there exists a bijection $\pi \colon V(A) \to V(B)$ such that $a_{u,v} = b_{\pi(u),\pi(v)}$ for all $u, v \in V(A)$; π is called an isomorphism. For two graphs G, H with adjacency matrices A_G, A_H we say that $G \cong H$ if $A_G \cong A_H$. We consider only undirected graphs without self-loops. A graph class $\mathcal{C}$ is a subset of all graphs which is closed under isomorphism, i.e. if $G \in \mathcal{C}$ and $H \cong G$ then $H \in \mathcal{C}$. We define the graph isomorphism problem for a graph class $\mathcal{C}$ as $\mathrm{GI}(\mathcal{C}) = \{(G, H) \mid G, H \in \mathcal{C} \text{ and } G \cong H\}$. For a graph G and a vertex $v \in V(G)$ we define the open neighborhood $N(v)$ of v as the set of vertices that are adjacent to v and the closed neighborhood $N[v] = N(v) \cup \{v\}$. For a subset of vertices $V' \subseteq V(G)$ we define the common neighborhood of V' as $N[V'] = \bigcap_{v \in V'} N[v]$. We also write $N[u, v, \dots]$ instead of $N[\{u, v, \dots\}]$. A vertex v is called universal if $N[v] = V(G)$. For two vertices $u \neq v \in V(G)$ we say that u and v are twins if $N[u] = N[v]$. A twin class is an equivalence class induced by the twin relation. A graph is said to be without twins if every twin class has cardinality one. For a graph G and $S' \subseteq S \subseteq V(G)$ we define the exclusive neighborhood $N_S(S')$ as all vertices $v \in V(G) \setminus S$ such that v is adjacent to all vertices in S' and to none in $S \setminus S'$.

▶ **Definition 1** (Label-independent)**.** Let f be a function which maps graphs to a subset of subsets of vertices, i.e. $f(G) \subseteq \mathcal{P}(V(G))$. We say f is label-independent if for every pair of isomorphic graphs G, H and all isomorphisms π from G to H it holds that

$$f(H) = \{\pi(X) \mid X \in f(G)\} \text{ with } \pi(X) = \{\pi(v) \mid v \in X\} .$$

2.1 Circular-Arc Graphs and Representations

A CA model is a set of arcs $\mathcal{A} = \{A_1, \ldots, A_n\}$ on the circle. Let $p \neq p'$ be two points on the circle. Then the arc A specified by $[p, p']$ is given by the part of the circle that is traversed when starting from p going in clockwise direction until p' is reached. We say that p is the left and p' the right endpoint of A and write $l(\cdot), r(\cdot)$ to denote the left and right endpoint of an arc in general. If $A = [p, p']$ then the arc obtained by swapping the endpoints $\overline{A} = [p', p]$ covers the opposite part of the circle. We say $\overline{A}$ is obtained by flipping A. When considering a CA model with respect to its intersection structure only the relative position of the endpoints to each other matter. W.l.o.g. all endpoints can be assumed to be pairwise different and no arc covers the full circle. Therefore a CA model $\mathcal{A}$ with n arcs can be described as a unique string as follows. Pick an arbitrary arc $A \in \mathcal{A}$ and relabel the arcs with $1, \ldots, n$ in order of appearance of their left endpoints when traversing the circle clockwise starting from the left endpoint of A. Then write down the endpoints in order of appearance when traversing the circle clockwise starting from the left endpoint of the chosen arc A. Do this for every arc and pick the lexicographically smallest resulting string as representation for $\mathcal{A}$. For example, the smallest such string for the CA model in Fig. 1 would result from choosing A_1 $(l(1), r(1), l(2), r(5), l(3), r(2), \ldots)$. In the following we identify $\mathcal{A}$ with its string representation.

Let G be a graph and $\rho = (\mathcal{A}, f)$ consists of a CA model $\mathcal{A}$ and a bijective mapping f from the vertices of G to the arcs in $\mathcal{A}$. Then ρ is called a CA representation of G if for all $u \neq v \in V(G)$ it holds that $\{u, v\} \in E(G) \Leftrightarrow f(u) \cap f(v) \neq \emptyset$. We write $\rho(x)$ to mean the arc $f(x)$ corresponding to the vertex x, $\rho(G)$ for the CA model $\mathcal{A}$ and for a subset $V' \subseteq V(G)$ let $\rho[V'] = \{\rho(v) \mid v \in V'\}$. Given a set $X \subseteq V(G)$ we write $\rho^+(X)$ to denote $\cup_{v \in X} \rho(v)$. A graph is a CA graph if it has a CA representation.

We say a CA model $\mathcal{A}$ has a hole if there exists a point on the circle which is not contained in any arc in $\mathcal{A}$. Every such CA model can be understood as interval model by straightening the arcs. Therefore a graph is an interval graph if it admits a CA representation with a hole. A CA graph G is called Helly (HCA graph) if it has a CA representation ρ such that for all maxcliques, i.e. inclusion-maximal cliques, C in G it holds that $\bigcap_{v \in C} \rho(v) \neq \emptyset$. Every interval model has the Helly property and therefore every interval graph is an HCA graph.

2.2 Normalized Representation

In [3] it was observed that the intersection type of two arcs $A \neq B$ can be one of the following five types: A and B are disjoint (`di`), A is contained in B (`cd`), A contains B (`cs`), A and B jointly cover the circle (circle cover `cc`) or A and B overlap (`ov`) but do not jointly cover the circle. Using these types we can associate a matrix with every CA model. An intersection matrix is a square matrix with entries $\{\texttt{di}, \texttt{ov}, \texttt{cs}, \texttt{cd}, \texttt{cc}\}$. Given a CA model $\mathcal{A}$ we define its intersection matrix $\mu_{\mathcal{A}}$ such that $(\mu_{\mathcal{A}})_{a,b}$ reflects the intersection type of the arcs $a \neq b \in \mathcal{A}$. An intersection matrix μ is called a CA (interval) matrix if it is the intersection matrix of some CA (interval) model.

When trying to construct a CA representation for a CA graph G it is clear that whenever two vertices are non-adjacent their corresponding arcs must be disjoint. If two vertices u, v are adjacent the intersection type of their corresponding arcs might be ambiguous. It would be convenient if the intersection type for every pair of vertices would be uniquely determined by G itself. This can be achieved by associating a graph G with an intersection matrix λ_G

called the neighborhood matrix which is defined for all $u \neq v \in V(G)$ as

$$(\lambda_G)_{u,v} = \begin{cases} \texttt{di} & \text{, if } \{u,v\} \notin E(G) \\ \texttt{cd} & \text{, if } N[u] \subsetneq N[v] \\ \texttt{cs} & \text{, if } N[v] \subsetneq N[u] \\ \texttt{cc} & \text{, if } N[u] \not\parallel N[v] \text{ and } N[u] \cup N[v] = V(G) \\ & \quad \text{and } \forall w \in N[u] \setminus N[v] : N[w] \subset N[u] \\ & \quad \text{and } \forall w \in N[v] \setminus N[u] : N[w] \subset N[v] \\ \texttt{ov} & \text{, otherwise} \end{cases}$$

and the first case applies whose condition is satisfied.

Let μ be an intersection matrix over the vertex set V and $\rho = (\mathcal{A}, f)$ where $\mathcal{A}$ is a CA model and f is a bijective mapping from V to $\mathcal{A}$. We say ρ is a CA representation of the matrix μ if μ is isomorphic to the intersection matrix of $\mathcal{A}$ via f and denote the set of such CA representations with $\mathcal{N}(\mu)$. Then we say ρ is a normalized CA representation of a graph G if ρ is a CA representation of the neighborhood matrix λ_G of G. An example of a normalized representation can be seen in Fig. 1. Let us denote the set of all normalized CA representations of G with $\mathcal{N}(G) = \mathcal{N}(\lambda_G)$.

▶ **Lemma 2** ([3]). *Every CA graph G without twins and universal vertices has a normalized CA representation, that is $\mathcal{N}(G) \neq \emptyset$.*

For our purpose it suffices to consider only graphs without twins and universal vertices for the same reasons as in [6]. The point is that a universal vertex can be removed from the graph and later added as arc which covers the whole circle in the representation. For each twin class an arbitrary representative vertex can be chosen and colored with the cardinality of its twin class; the other twins are removed.

▶ **Lemma 3.** *The canonical CA representation problem for CA graphs is logspace reducible to the canonical CA representation problem for colored CA graphs without twins and universal vertices.*

Henceforth we assume every graph to be twin-free and without universal vertices and shall only consider normalized representations. For two vertices $u \neq v$ in a graph G we write $u\,\alpha\,v$ instead of $(\lambda_G)_{u,v} = \alpha$. For example, $u\,\texttt{cd}\,v$ indicates that the arc of u must be contained in the arc of v for every (normalized) representation of G.

3 Flip Trick

Let $\mathcal{A}$ be a CA model and $X \subseteq \mathcal{A}$ is a subset of arcs to be flipped. We define the resulting CA model $\mathcal{A}^{(X)} = \left\{\overline{A} \mid A \in X\right\} \cup \mathcal{A} \setminus X$. Consider a point x on the circle and let X be the set of arcs that contain this point. Then after flipping the arcs in X no other arc contains the point x and thus $\mathcal{A}^{(X)}$ has a hole and therefore must be an interval model. Let μ and $\mu^{(X)}$ be the intersection matrices of $\mathcal{A}$ and $\mathcal{A}^{(X)}$ respectively. It was observed in [8] that the interval matrix $\mu^{(X)}$ can be easily computed using μ and X as input via Table 1. With this the problem of computing a canonical CA representation for colored CA graphs can be reduced to the canonical interval representation problem for colored interval matrices, which can be solved in logspace[6] (the colored part is not mentioned explicitly but can be easily incorporated into the proof by adding the colors to the leaves of the colored Δ tree).

Table 1 Effects of flipping arcs in the intersection matrix

$\mu_{A,B}$	di	cd	cs	cc	ov
$\mu_{\bar{A},B}$	cs	cc	di	cd	ov
$\mu_{A,\bar{B}}$	cd	di	cc	cs	ov
$\mu_{\bar{A},\bar{B}}$	cc	cs	cd	di	ov

The idea is that given a CA graph G if we can compute a set of vertices X as described above then we can obtain a canonical CA representation for G by the following argument. The neighborhood matrix λ_G of G is a CA matrix and the matrix $\lambda_G^{(X)}$ must be an interval matrix. Compute a canonical interval representation for $\lambda_G^{(X)}$ and flip the arcs in X back. This leads to a representation for λ_G and thus G. The required set X can be specified as follows.

▶ **Definition 4.** Let G be a CA graph. Then a non-empty $X \subseteq V(G)$ is a flip set iff there exists a representation $\rho \in \mathcal{N}(G)$ and a point x on the circle such that $v \in X \Leftrightarrow x \in \rho(v)$.

In fact, for the argument to obtain a canonical representation to hold it is only required that X is chosen such that $\lambda_G^{(X)}$ is an interval matrix. However, it can be shown that this is equivalent to the above definition. Now, we can reframe the argument given in [6] as follows.

▶ **Definition 5** (Candidate function)**.** Let $\mathcal{C}$ be a subset of all CA graphs and f is a function which maps graphs to a subset of subsets of their vertices, i.e. $f(G) \subseteq \mathcal{P}(V(G))$. We call f a candidate function for $\mathcal{C}$ if the following conditions hold:

1. For every $G \in \mathcal{C}$ there exists an $X \in f(G)$ such that X is a flip set.
2. f is label-independent.

▶ **Theorem 6.** *If f is a candidate function for all CA graphs that can be computed in logspace then the canonical representation problem for CA graphs can be solved in logspace.*

Proof. Let G be a graph. To decide the recognition problem for CA graphs observe that there is a flip set in $f(G)$ iff G is a CA graph. To verify if a set $X \subseteq V(G)$ is a flip set one can check if $\lambda_G^{(X)}$ is an interval matrix by trying to compute an interval representation.

For the representation problem let G be a CA graph. Let F be the subset of $f(G)$ such that every $X \in F$ is a flip set. By the first condition it holds that F is non-empty. For every $X \in F$ a CA representation ρ_X of G can be computed by the previous argument. We return a CA representation with the lexicographically smallest underlying model

$$\operatorname*{argmin}_{\{\rho_X \mid X \in F\}} \rho_X(G)$$

as canonical CA representation. To see that this is indeed a canonical representation consider two isomorphic CA graphs G, H with F_G, F_H defined as F for G previously. Let π be an isomorphism from G to H and $M_G = \{\rho_X(G) \mid X \in F_G\}$ is the set of CA models induced by the flip sets F_G, similarly define M_H. Then canonicity follows by showing $M_G = M_H$. We show that $M_G \subseteq M_H$ as the argument for the other direction is analogous. Let $\mathcal{A}$ be a model in M_G. Let X be a flip set in F_G which induces $\mathcal{A}$, i.e. $\rho_X(G) = \mathcal{A}$. It must hold that $\pi(X) \in f(H)$ since f is label-independent. From $G \cong H$ it follows that $\lambda_G \cong \lambda_H$ and therefore the interval matrices $\lambda_G^{(X)}$ and $\lambda_H^{(\pi(X))}$ are isomorphic meaning that $\pi(X) \in F_H$ since it is a flip set as well. As the interval representations of both interval matrices have

identical underlying models due to canonicity it follows that they remain so after flipping X resp. $\pi(X)$. Therefore it holds that $\mathcal{A} \in M_H$.

This works in logspace since f and the representation ρ_X for every flip set X can be computed in logspace. ◀

So, we have reduced the problem of computing a canonical CA representation for CA graphs to the problem of computing a candidate function for CA graphs.

Let $\mathcal{C}$ and $\mathcal{C}'$ be two graph classes that partition all CA graphs. If $f_{\mathcal{C}}, f_{\mathcal{C}'}$ are candidate functions for $\mathcal{C}, \mathcal{C}'$ respectively then $f(G) = f_{\mathcal{C}}(G) \cup f_{\mathcal{C}'}(G)$ is a candidate function for all CA graphs. That f is label-independent follows from label-independent functions being closed under taking unions. The crux here is that we do not need to be able to distinguish if a CA graph G is in $\mathcal{C}$ or $\mathcal{C}'$. Hence, in the next two sections we consider two such classes that partition all CA graphs while avoid dealing with recognition of these two classes.

We complete this section by stating the candidate function used in [6] to canonize HCA graphs and explain why it is a candidate function for this subclass of CA graphs.

$$f_{\mathrm{HCA}}(G) = \bigcup_{u,v \in V(G)} \{N[u,v]\}.$$

f_{HCA} always returns at least one flip set for an HCA graph because all maxcliques in an HCA graph are flip sets due to the Helly property and there exists at least one maxclique in every HCA graph that can be characterized as the common neighborhood of two vertices[6]. However, neither of these two properties hold for CA graphs in general.

It remains to argue that f_{HCA} can be computed in logspace and is label-independent. Since the same arguments have to be made for the two candidate functions devised in the next sections we introduce a tool that facilitates this and demonstrate it for f_{HCA}.

▶ **Definition 7.** Let φ be a first-order (FO) formula over graph structures with $k+1$ free variables. Then we define the function f_φ for a graph G as:

$$f_\varphi(G) = \bigcup_{(v_1,\ldots,v_k) \in V(G)^k} \{\, \{u \in V(G) \mid G \models \varphi(v_1,\ldots,v_k,u)\} \,\}.$$

▶ **Lemma 8.** *For every FO formula φ over graph structures the function f_φ is computable in logspace and label-independent.*

To compute f_φ we can successively evaluate φ and take the union of the results, which can be both done in logspace. To show that f_φ is label-independent it suffices to apply structural induction to φ. Then f_{HCA} is computed by the FO formula $\varphi(u,v,x)$ that states that $x \in N[u,v]$. By Lemma 8 it follows that f_{HCA} is logspace-computable and label-independent.

4 Uniform CA graphs

The difficulty when trying to compute flip sets for CA graphs in general is that for a CA graph G there might be different normalized CA representations such that a set of vertices shares a common point in one representation but not in an other one. We show a subset of CA graphs, namely the uniform CA graphs, where this issue does not occur therefore making it easy to compute flip sets.

For a CA graph G consider an arbitrary vertex u. Looking at the neighbors of u we can try to compute the flip sets $X_{u,1}, X_{u,2}$ specified in Fig. 2. Both sets contain u and all vertices that contain u or form a circle cover with u. The vertices that overlap with u belong

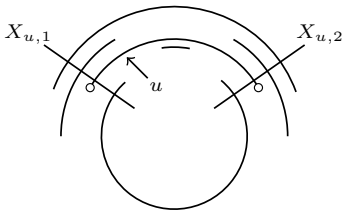


Figure 2 Uniform target flip sets.

to either $X_{u,1}$ or $X_{u,2}$ depending on the side they overlap from with u. Since we cannot determine left and right from the neighborhood matrix we want to express an equivalence relation $\sim_u$ which states that two vertices overlap from the same side with u. With the two equivalence classes induced by $\sim_u$ the two flip sets can be expressed.

Given two vertices x, y that both overlap with u it is for instance easy to see that they must overlap from different sides with u if they are disjoint. The only intersection type between x, y for which the situation is not immediately clear is ov as further distinctions are required. An ov-triangle is a set of three pairwise overlapping vertices. If x and y overlap then x, y, u form such an ov-triangle. Consider the possible normalized representations for an ov-triangle. The three vertices can either all jointly cover the circle or be a set of overlapping intervals. In the first case they overlap pairwise but their overall intersection is empty thus we call this a non-Helly triangle and the second case an interval triangle. For an interval triangle there are three different possible representations up to reflection depending on which of the three vertices is placed in-between the other two. If x, y, u is an ov-triangle then x and y overlap from the same side with u iff x, y, u form an interval triangle and u is not in-between x, y. We show that it is easy to derive this information in the case of a uniform CA graph G as it does not depend on a representation of G.

▶ **Definition 9.** Let G be a graph. An ov-triangle T is in the set Δ_G if the following holds:

1. $\bigcup_{v \in T} N[v] = V(G)$
2. For all $x \in T$ it holds that if a vertex $v \in N_T(x)$ then v cd x

▶ **Definition 10** (Uniform CA graph)**.** A CA graph G is uniform if for all ov-triangle T in G and $\rho \in \mathcal{N}(G)$ it holds that:

$$T \in \Delta_G \Rightarrow \rho[T] \text{ is a non-Helly triangle}$$

The idea behind the definition of Δ_G is that it captures the properties that an ov-triangle must satisfy in the graph if it can be represented as non-Helly triangle. The definition of uniform CA graphs guarantees us that an ov-triangle $T \in \Delta_G$ can never be represented as interval triangle. Now, we can show that for the class of uniform CA graphs the property of being a non-Helly triangle and the in-between predicate for interval triangles is invariant across all normalized representations.

▶ **Lemma 11.** *Let G be a uniform CA graph. Then the following statements are equivalent for every* ov*-triangle T:*

1. $T \in \Delta_G$
2. $\exists \rho \in \mathcal{N}(G) : \rho[T]$ *is a non-Helly triangle*
3. $\forall \rho \in \mathcal{N}(G) : \rho[T]$ *is a non-Helly triangle*

Proof. This immediately follows from the definition of Δ_G and uniform CA graphs. ◀

As a contrasting example of a (non-uniform) CA graph for which being a non-Helly triangle depends on the representation consider the graph G obtained by taking the complement of the disjoint union of three K_2's (a triforce with a circle around the outer corners). Every edge in G is an $\texttt{ov}$-entry in λ_G and a CA model for G is given by precisely two non-Helly triangles. The possible assignments of the vertices to the arcs that follow from the automorphisms of G yield the different representations.

▶ **Definition 12.** Let G be a graph and $T = \{u, v, w\}$ is an $\texttt{ov}$-triangle with $T \notin \Delta_G$. We say v is in-between u, w if at least one of the following holds:

1. $N_T(u), N_T(w) \neq \emptyset$
2. $N_T(u, w) \neq \emptyset$ and there exists $z \in N_T(u, w)$ such that $\{u, w, z\} \in \Delta_G$

▶ **Lemma 13.** *Let G be a uniform CA graph and $T = \{u, v, w\}$ is an $\texttt{ov}$-triangle with $T \notin \Delta_G$. Then the following statements are equivalent:*

1. *v in-between u, w*
2. $\exists \rho \in \mathcal{N}(G) : \rho(v) \subset \rho(u) \cup \rho(w)$
3. $\forall \rho \in \mathcal{N}(G) : \rho(v) \subset \rho(u) \cup \rho(w)$

Proof. "2 ⇒ 1": There exists $\rho \in \mathcal{N}(G)$ such that $\rho(v) \subset \rho(u) \cup \rho(w)$. For all $x \neq y \in T$ it must hold that $N[x] \setminus N[y] \neq \emptyset$ due to the fact that $x \,\texttt{ov}\, y$ implies $N[x] \not\subset N[y]$. For this to be true $N_T(x)$ or $N_T(x, z)$ must be non-empty for $z \in T \setminus \{x, y\}$. It follows that $N_T(u)$ and $N_T(w)$ or $N_T(u, w)$ must be non-empty since $N_T(v) = \emptyset$ due to $\rho(v) \subset \rho(u) \cup \rho(w)$. If $z \in N_T(u, w)$ then for $\rho(z)$ to intersect with $\rho(u)$ and $\rho(w)$ but not with $\rho(v)$ implies that $\rho[u, w, z]$ forms a non-Helly triangle and therefore $\{u, w, z\} \in \Delta_G$ by Lemma 11.
"1 ⇒ 3": Assume there exists a $\rho \in \mathcal{N}(G)$ such that $\rho(u) \subset \rho(v) \cup \rho(w)$. This implies that $N_T(u) = \emptyset$ and therefore there exists a $z \in N_T(u, w)$ such that $\{u, w, z\} \in \Delta_G$. By Lemma 11 it follows that $\rho[u, w, z]$ must be a non-Helly triangle and $\rho(z)$ must be disjoint from $\rho(v)$, contradiction.
"3 ⇒ 2": is clear. ◀

Lemma 11 and 13 state a fact of the form that if a property holds in one representation then it holds in all representations hence the name uniform CA graphs.

The α-neighborhood of a vertex u in a graph G is defined as $N^\alpha(u) = \{v \in N(u) \mid u \,\alpha\, v\}$ for $\alpha \in \{\texttt{ov}, \texttt{cd}, \texttt{cs}, \texttt{cc}\}$. Now, we can define the aforementioned equivalence relation $\sim_u$ and state the candidate function for uniform CA graphs.

▶ **Definition 14.** Given a graph G and vertex $u \in V(G)$ we define the relation $\sim_u$ on $N^{\texttt{ov}}(u)$ such that $x \sim_u y$ holds if one of the following applies:

1. $x = y$
2. x contains or is contained in y
3. x and y overlap, $\{x, y, u\} \notin \Delta_G$ and u is not in-between x, y

▶ **Lemma 15.** *Let G be a uniform CA graph and $u \in V(G)$. Then $x \sim_u y$ holds iff x and y overlap from the same side with u in every $\rho \in \mathcal{N}(G)$.*

Proof. "⇒": If x, y are in a contained/contains relation this is clear. For the third condition it holds that for all $\rho \in \mathcal{N}(G)$ $\rho[x, y, u]$ is an interval triangle and x or y is in-between the other two. It follows that x, y overlap from the same side with u.
"⇐": If $\lambda_{x,y} \in \{\texttt{di}, \texttt{cc}\}$ this is clear. If x, y overlap then they either form a non-Helly triangle or u is in-between x, y. In both cases x, y overlap from different sides with u. ◀

▶ **Theorem 16.** *The following mapping is a candidate function for uniform CA graphs and can be computed in logspace:*

$$f_U(G) = \bigcup_{\substack{u \in V(G) \\ x \in N^{\mathtt{ov}}(u)}} \left\{ \{u\} \cup N^{\mathtt{cd}}(u) \cup N^{\mathtt{cc}}(u) \cup \{y \in N^{\mathtt{ov}}(u) | x \sim_u y\} \right\}.$$

Proof. To show that $f_{\mathrm{U}}(G)$ is a candidate function we have to prove that for every uniform CA graph G there always exists a flip set $X \in f_{\mathrm{U}}(G)$ and that f_{U} is label-independent. In fact, the even stronger claim holds that for all uniform CA graphs G every set in $f_{\mathrm{U}}(G)$ is a flip set. Let $X \in f_{\mathrm{U}}(G)$ via some $u \in V(G)$ and $x \in N^{\mathtt{ov}}(u)$. Then the set of vertices in X correspond to one of the two flip sets shown in Fig. 2. The correctness for the subset of vertices in X overlapping with u follows from Lemma 15.

To show that $f_{\mathrm{U}}(G)$ can be computed in logspace and is label-independent we apply Lemma 8. We can rewrite f_{U} as

$$f_{\mathrm{U}}(G) = \bigcup_{u,x \in V(G)} \left\{ \{z \in V(G) \mid G \models \varphi(u,x,z)\} \right\}$$

with $G \models \varphi(u,x,z)$ iff $x \in N^{\mathtt{ov}}(u)$ and $z \in \{u\} \cup N^{\mathtt{cd}}(u) \cup N^{\mathtt{cc}}(u) \cup \{y \in N^{\mathtt{ov}}(u) | x \sim_u y\}$. It remains to check that the entries in the neighborhood matrix, α-neighborhoods, exclusive neighborhoods, Δ_G, in-between and $\sim_u$ can be expressed in FO logic. ◀

▶ **Corollary 17.** *A canonical CA representation for uniform CA graphs can be computed in logspace.*

▶ **Theorem 18.** *Helly CA graphs are a strict subset of uniform CA graphs.*

Proof. First, we show "$\subseteq$" by contradiction. Assume there exists a Helly CA graph G which is non-uniform. For G to be non-uniform there must exist an $\mathtt{ov}$-triangle $T \in \Delta_G$ and a representation $\rho \in \mathcal{N}(G)$ such that T is represented as interval triangle in ρ. Let $T = \{u, v, w\}$ and assume w.l.o.g. that $\rho(v) \subset \rho(u) \cup \rho(w)$ (v is in-between u, w). It follows that $N[u] \cup N[w] = V(G)$. Since $\lambda_{u,w} \neq \mathtt{cc}$ there must be a $u' \in N[u] \setminus N[w]$ such that $N[u'] \setminus N[u] \neq \emptyset$. This means there exists a $w' \in N[u', w] \setminus N[u]$. As $N[u'] \not\subset N[w']$ it follows that u' and w' overlap. For u' it must hold that it is either in $N_T(u)$ or $N_T(u, v)$. If it is in $N_T(u)$ then by the second condition of Δ_G it follows that u' must be contained in u, contradiction. For the same reason w' is in $N_T(v, w)$. It follows that u', v, w' form an $\mathtt{ov}$-triangle and must be represented as non-Helly triangle in ρ. This contradicts that G is a Helly CA graph.

To see that this inclusion is strict consider the graph G obtained by taking a triangle T and attaching a new vertex to each vertex in T (also known as net graph). In every representation $\rho \in \mathcal{N}(G)$ it must hold that T is represented as non-Helly triangle since $N_T(v) \neq \emptyset$ for all $v \in T$. For this reason G cannot be a Helly or a non-uniform CA graph. ◀

5 Non-Uniform CA graphs

From the definition of uniform CA graphs it follows that a CA graph G is non-uniform if there exists an $\mathtt{ov}$-triangle T in Δ_G and a $\rho \in \mathcal{N}(G)$ such that T is represented as interval triangle in ρ. We call the pair (T, ρ) a witness for the non-uniformity of G and also say G is non-uniform via (T, ρ). Additionally, for such a witness pair (T, ρ) we call T maximal if there exists no $T' \neq T$ such that G is non-uniform via (T', ρ) and $\rho^+(T) \subset \rho^+(T')$. Such a maximal T for a given ρ must always exist. For the purpose of computing a candidate

function for this class we can assume that for a given non-uniform CA graph G we are supplied with an $\mathtt{ov}$-triangle T such that there exists a $\rho \in \mathcal{N}(G)$ with (T, ρ) being a witness for G and T is maximal. This is justified by the fact that we can iterate over all $\mathtt{ov}$-triangle $T \in \Delta_G$ trying to compute flip sets knowing that for at least one such T these conditions are met. Additionally, we write T as ordered triple (u, v, w) to indicate that v is in-between u and w in ρ.

Let G be non-uniform via (T, ρ). Consider for a vertex $x \in V(G) \setminus T$ what the possible relations between $\rho(x)$ and $\rho^+(T)$ are. For instance, $\rho(x)$ cannot be disjoint from $\rho^+(T)$ because this implies that $x \notin \cup_{t \in T} N[t]$ and therefore $T \notin \Delta_G$. Also, $\rho(x)$ cannot contain $\rho^+(T)$ as this would mean that x is a universal vertex.

▶ **Definition 19.** Let G be a non-uniform CA graph via (T, ρ). The set of normalized representations that agree with ρ on T is:

$$\mathcal{N}_\rho^T(G) = \{\rho' \in \mathcal{N}(G) \mid \rho'[T] = \rho[T]\} \ .$$

▶ **Definition 20.** Let G be a non-uniform CA graph via (T, ρ) and $\alpha \in \{\mathtt{ov}, \mathtt{cc}, \mathtt{cd}\}$. We say $x \in V(G) \setminus T$ is an α-arc in ρ' if

$$\rho'(x) \, \alpha \, \rho^+(T)$$

for some $\rho' \in \mathcal{N}_\rho^T(G)$. We call x an unambiguous α-arc if the above condition holds for all $\rho' \in \mathcal{N}_\rho^T(G)$.

▶ **Definition 21.** Let G be a graph and $T = (u, v, w) \in \Delta_G$. Then we define the following sets w.r.t. T:

$$\begin{aligned}
\Gamma_{\mathtt{ov},u} &= \{x \in V(G) \setminus T \mid x \ \mathtt{ov} \ u, v \ , \ x \ \mathtt{di} \ w\} \\
\Gamma_{\mathtt{ov},w} &= \{x \in V(G) \setminus T \mid x \ \mathtt{ov} \ v, w \ , \ x \ \mathtt{di} \ u\} \\
\Gamma_{\mathtt{ov}} &= \Gamma_{\mathtt{ov},u} \cup \Gamma_{\mathtt{ov},w} \\
\Gamma_{\mathtt{cc}} &= \left\{ x \in V(G) \setminus T \;\middle|\; \begin{array}{l} x \ \mathtt{ov} \ u, w \ , \ x \ \mathtt{di} \ v \text{ or} \\ \exists a \in T : x \ \mathtt{cc} \ a \end{array} \right\} \\
\Gamma_{\mathtt{cd}} &= \left\{ x \in V(G) \setminus T \;\middle|\; \begin{array}{l} x \ \mathtt{ov} \ u, w \ , \ x \ \mathtt{cs} \ v \text{ or} \\ \exists a \in T : x \ \mathtt{cd} \ a \end{array} \right\} \\
\Omega &= \{x \in V(G) \setminus T \mid x \ \mathtt{ov} \ u, v, w\}
\end{aligned}$$

▶ **Lemma 22.** *If G is a non-uniform CA graph via (T, ρ) such that T is maximal then $T, \Gamma_{\mathtt{ov}}, \Gamma_{\mathtt{cc}}, \Gamma_{\mathtt{cd}}, \Omega$ partition $V(G)$.*

Proof. It is not hard to see that these sets do not overlap. To show that every vertex in G belongs to one of these sets we need to check all possible positions of the endpoints of a vertex x not in T relative to T, consider Fig. 3. We know every vertex $x \notin T$ must be represented as α-arc for some $\alpha \in \{\mathtt{ov}, \mathtt{cd}, \mathtt{cc}\}$. For $\alpha \in \{\mathtt{cd}, \mathtt{cc}\}$ it must hold that both endpoints of x must be in one of the intervals 1–5. The exemplary x depicted in Fig. 3 is a $\mathtt{cd}$-arc in the given representation and overlaps with u, v, w. If x is a $\mathtt{cd}$-arc then the number of the interval in which its left endpoint is situated must be less than or equal that of its right endpoint (in our example $2 \leq 5$). If it is a $\mathtt{cc}$-arc then the right endpoint must come before the left. The graph on the right encodes all possible placements of the two endpoints and it can be verified case-by-case that an α-arc for $\alpha \in \{\mathtt{cd}, \mathtt{cc}\}$ will occur in either Γ_α or

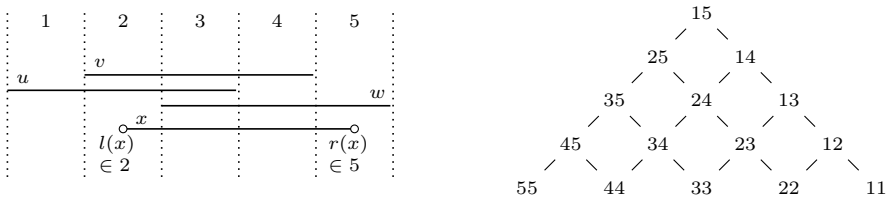


Figure 3 Possible positions of the endpoints of x relative to $T = \{u, v, w\}$

Ω. It remains to argue that an ov-arc can only have the intersection structure denoted by $\Gamma_{\texttt{ov}}$. W.l.o.g. assume that x is an ov-arc that overlaps from u's side with T. It holds that the right endpoint of x is in one of the five intervals and the left endpoint must be in none of these five intervals. If $r(x) \in 1$ then $x \in N_T(u)$ and by the second condition of Δ_G it must hold that x is contained in u, contradiction. If $r(x) \in 2$ then x overlaps with u, v and is disjoint with w and therefore $x \in \Gamma_{\texttt{ov},u}$. If $r(x) \in \{3, 4\}$ then $T = \{u, v, w\}$ is not maximal since $\{x, v, w\} \in \Delta_G$. If $r(x) \in 5$ then it can be shown that there must be a circle cover entry in the neighborhood matrix for x, w which contradicts that they must overlap. ◀

▶ **Lemma 23.** *Let G be a non-uniform CA graph via (T, ρ) and T is maximal. All vertices in Γ_α are unambiguous α-arcs for $\alpha \in \{\texttt{ov}, \texttt{cd}, \texttt{cc}\}$.*

Proof. The intersection structure of a vertex $x \in \Gamma_\alpha$ with T dictates the positioning of the endpoints relative to T in every $\rho' \in \mathcal{N}_\rho^T(G)$. It follows that this placement of the endpoints of x must hold for all $\rho' \in \mathcal{N}_\rho^T(G)$ and therefore x is an unambiguous α-arc. ◀

For a vertex $x \in \Omega$ the possible placements of its endpoints to satisfy the intersection structure with T can be one of the following four types $(\texttt{cd}, 14), (\texttt{cd}, 25), (\texttt{cc}, 14), (\texttt{cc}, 25)$. For example x in Fig. 3 is type $(\texttt{cd}, 25)$ and flipping x would lead to type $(\texttt{cc}, 25)$.

Let G be non-uniform via (T, ρ) and $T = (u, v, w)$ is maximal. Then the two target flip sets X_u, X_w we want to compute in the non-uniform case are immediately before the left endpoint and immediately after the right endpoint of $\rho^+(T)$ and must be of the form

$$X_i = \Gamma_{\texttt{ov},i} \cup \Gamma_{\texttt{cc}} \cup \Omega'$$

for $i \in \{u, w\}$ and some subset $\Omega' \subseteq \Omega$.

▶ **Definition 24.** Let G be a non-uniform CA graph via (T, ρ) and T is maximal. We call a subset $\Omega' \subseteq \Omega$ cc-realizable w.r.t. (T, ρ) if there exists a $\rho' \in \mathcal{N}_\rho^T(G)$ such that for all $x \in \Omega$ it holds that x is a cc-arc in ρ' iff $x \in \Omega'$.

In other words, a cc-realizable set is a subset of Ω such that all of its vertices can be represented as cc-arc in a normalized representation. By finding such a set we can construct two flip sets by adding the cc-vertices in $\Gamma_{\texttt{cc}}$ and one side of the ov-vertices as described above. Now, the challenge consists in finding such a cc-realizable subset. A way to solve this is to parameterize our input by the cardinality of Ω and try all possibilities. Since Ω and its cardinality depend on the particular ov-triangle T chosen, which we do not know a priori, we can use the following set which is a superset of every possible Ω and thus bounds the cardinality:

$$K_G = \{x \in V(G) \mid \exists \{u, v, w\} \in \Delta_G \text{ s.t. } x \,\texttt{ov}\, u, v, w\} \ .$$

▶ **Theorem 25.** *The following mapping is a candidate function for non-uniform CA graphs and can be computed in $\mathcal{O}(k + \log n)$ space for $k = |K_G|$:*

$$f_N(G) = \bigcup_{\substack{(u,v,w)\in\Delta_G \\ \Omega'\subseteq\Omega}} \{\Gamma_{\mathtt{ov},u} \cup \Gamma_{\mathtt{cc}} \cup \Omega'\}$$

where $\Gamma_{\mathtt{ov},u}, \Gamma_{\mathtt{cc}}, \Omega$ are taken w.r.t. $T = (u, v, w)$.

Proof. To show that there always exists a flip set $X \in f_N(G)$ for a non-uniform CA graph G we argue as follows. Let G be non-uniform via (T, ρ) and $T = (u, v, w)$ is maximal. Let Ω' be a cc-realizable set w.r.t. (T, ρ), which must exist since G is non-uniform. Then $X = \Gamma_{\mathtt{ov},u} \cup \Gamma_{\mathtt{cc}} \cup \Omega'$ w.r.t. T is one of the target flip sets described previously. This means there exists a $\rho \in \mathcal{N}(G)$ such that X describes the set of arcs that contain a point right before the left endpoint of $\rho^+(T)$ or a point right after the right endpoint of $\rho^+(T)$.

To see that $f_N(G)$ is label-independent a formula $\varphi(\Omega', u, v, w, x)$ can be constructed that is true iff $x \in \Gamma_{\mathtt{ov},u} \cup \Gamma_{\mathtt{cc}} \cup \Omega'$ where Ω' is a second-order set variable. Note, that $|\Omega| \leq |K_G|$. Therefore this works in $\mathcal{O}(k + \log n)$ space since one can iterate over all 2^k subsets of Ω using k bits and then apply the argument in Lemma 8 via φ which requires additional $\mathcal{O}(\log n)$ space. ◀

6 Conclusion

We showed how to canonically, or in our terms label-independently, compute flip sets for CA graphs to acquire canonical CA representations. The properties of uniform CA graphs enable us to do this easily in logspace. In the case of non-uniform CA graphs, however, it seems that the cc-realizable sets pose a non-trivial obstacle when trying to compute flip sets. The only simple remedy appears to be the proposed parameterization that enables us to use brute force. Changing the target flip sets does not seem to improve upon this situation. As a consequence, we suggest to investigate the space of cc-realizable sets. Given the restricted structure of non-uniform CA graphs this could be a reasonable first step towards deciding isomorphism for CA graphs in polynomial time.

Additionally, in [8] it was shown how to compute flip sets for CA graphs in linear time without the canonicity constraint. Can this be done in logspace as well? This would mean that recognition of CA graphs is logspace-complete.

Acknowledgments. We thank the anonymous reviewers for their helpful comments on earlier drafts of this paper.

References

1 Andrew Curtis, Min Chih Lin, Ross McConnell, Yahav Nussbaum, Francisco Soulignac, Jeremy Spinrad, and Jayme Szwarcfiter. Isomorphism of graph classes related to the circular-ones property. *Discrete Mathematics and Theoretical Computer Science*, 15(1), 2013. URL: http://www.dmtcs.org/dmtcs-ojs/index.php/dmtcs/article/view/2298.

2 Elaine Marie Eschen. *Circular-arc Graph Recognition and Related Problems.* PhD thesis, Vanderbilt University, Nashville, TN, USA, 1998. UMI Order No. GAX98-03921.

3 Wen-Lian Hsu. $O(M \cdot N)$ algorithms for the recognition and isomorphism problems on circular-arc graphs. *SIAM J. Comput.*, 24(3):411–439, June 1995. doi:10.1137/S0097539793260726.

4 Johannes Köbler, Sebastian Kuhnert, and Oleg Verbitsky. Solving the Canonical Representation and Star System Problems for Proper Circular-Arc Graphs in Logspace. In *FSTTCS*, volume 18, pages 387–399. Schloss Dagstuhl, 2012.

5 Johannes Köbler, Sebastian Kuhnert, Bastian Laubner, and Oleg Verbitsky. Interval graphs: Canonical representations in logspace. *SIAM J. Comput.*, 40(5):1292–1315, 2011. doi:10.1137/10080395X.

6 Johannes Köbler, Sebastian Kuhnert, and Oleg Verbitsky. Helly circular-arc graph isomorphism is in logspace. In *MFCS 2013*, volume 8087 of *Lecture Notes in Computer Science*, pages 631–642. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-40313-2_56.

7 George S. Lueker and Kellogg S. Booth. A linear time algorithm for deciding interval graph isomorphism. *J. ACM*, 26(2):183–195, April 1979. doi:10.1145/322123.322125.

8 Ross M. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica*, 37(2):93–147, 2003. doi:10.1007/s00453-003-1032-7.

9 Tsong-Ho Wu. *An $O(n^3)$ Isomorphism Test for Circular-Arc Graphs*. PhD thesis, SUNY Stony Brook, New York, NY, USA, 1983.

Bottleneck Paths and Trees and Deterministic Graphical Games

Shiri Chechik[1], Haim Kaplan[2], Mikkel Thorup[3], Or Zamir[4], and Uri Zwick[5]

1 **Blavatnik School of Computer Science, Tel Aviv University, Israel**
shiri.chechik@gmail.com
2 **Blavatnik School of Computer Science, Tel Aviv University, Israel**
haimk@tau.ac.il
3 **Department of Computer Science, University of Copenhagen, Denmark**
mikkel2thorup@gmail.com
4 **Blavatnik School of Computer Science, Tel Aviv University, Israel**
orzamir@mail.tau.ac.il
5 **Blavatnik School of Computer Science, Tel Aviv University, Israel**
zwick@tau.ac.il

Abstract

Gabow and Tarjan showed that the *Bottleneck Path* (BP) problem, i.e., finding a path between a given source and a given target in a weighted directed graph whose largest edge weight is minimized, as well as the *Bottleneck spanning tree* (BST) problem, i.e., finding a directed spanning tree rooted at a given vertex whose largest edge weight is minimized, can both be solved deterministically in $O(m \log^* n)$ time, where m is the number of edges and n is the number of vertices in the graph. We present a slightly improved randomized algorithm for these problems with an expected running time of $O(m\beta(m,n))$, where $\beta(m,n) = \min\{k \geq 1 \mid \log^{(k)} n \leq \frac{m}{n}\} \leq \log^* n - \log^*(m/n) + 1$. This is the first improvement for these problems in over 25 years. In particular, if $m \geq n \log^{(k)} n$, for some constant k, the expected running time of the new algorithm is $O(m)$. Our algorithm, as that of Gabow and Tarjan, work in the *comparison model*. We also observe that in the word-RAM model, both problems can be solved deterministically in $O(m)$ time. Finally, we solve an open problem of Andersson et al., giving a deterministic $O(m)$-time comparison-based algorithm for solving deterministic 2-player turn-based zero-sum terminal payoff games, also known as *Deterministic Graphical Games* (DGG).

1998 ACM Subject Classification G.2.2 Graph Theory – Graph Algorithms

Keywords and phrases bottleneck paths, comparison model, deterministic graphical games

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.27

1 Introduction

The *Bottleneck Path* (BP) problem, also known as the *min-max path* problem, is the problem of finding a path from a given source s to a given target t in a weighted directed graph $G = (V, E)$ in which the maximum edge weight is minimized. In the closely related *Bottleneck Spanning Tree* (BST) problem, also known as the *min-max arborescence* problem, we are asked to find a directed spanning tree of a given weighted directed graph $G = (V, E)$, rooted at a given root vertex s, such that the maximum edge weight in the tree is minimized.

Deterministic Graphical games (DGG) form a simple and interesting family of 2-player turn-based zero-sum games. A DGG is played by two players, players 0 and 1, on a directed


© Shiri Chechik, Haim Kaplan, Mikkel Thorup, Or Zamir, and Uri Zwick;
licensed under Creative Commons License CC-BY
33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 27; pp. 27:1–27:13



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

graph $G = (V, E)$ whose vertex set V is partitioned into $V = V_0 \uplus V_1 \uplus T$, [1] where V_i is the set of vertices controlled by player i, where $i = 0, 1$, and T is the set of *terminals*. A *payoff* function $p : T \to \mathbb{R}$ assigns a payoff to each terminal. A token is placed at a *start* vertex $s \in V_0 \cup V_1$. If the token is currently at a vertex $u \in V_i$, then player i chooses an edge $(u, v) \in E$ and moves the token to v. Each non-terminal vertex is assumed to have at least one outgoing edge. Terminals have no outgoing edges. If the token reaches a terminal t, the game ends and player 0 pays player 1 the payoff $p(t)$. Player 0, also known as min, wants to minimize this payoff, while player 1, also known as max, wants to maximize it. If the game never ends, no payment is made. Player 0 prefers an infinite play over a positive payoff, while player 1 prefers an infinite play over a negative payoff. A DGG, with start s, is solved by finding its min-max value and optimal strategies for the two players.

BP, BST and DGG share several similar features. First, they are both min-max optimization problems, though under different interpretations. Second, they can all be solved by trivial linear time algorithms if the edge weights, or the terminal payoffs, are given in *sorted* order. Third, they can all be solved using a *threshold method* that goes back to Edmonds and Fulkerson [9] when the edge weights, or the terminal payoffs, are not given in sorted order. In the case of the BP problem, for example, we find the *median* of the edge weights and partition the edges accordingly into *light* and *heavy* edges. We now check whether there is a directed path from s to t that uses only light edges. If so, all heavy edges can be discarded. If there is no such light path, we can set the weight of all light edges to $-\infty$. In the next iteration, we compute the median of all edges whose weight is not $-\infty$. This easily leads on $O(m \log n)$-time algorithms for the BP, BST and DGG problems. We also note that if the graph is undirected, we can *contract* the light edges. This leads to simple $O(m)$-time algorithms for the BP and BST problems in undirected graphs.

Gabow and Tarjan [16] used a more sophisticated version of the threshold method to obtain an $O(m \log^* n)$-time algorithm for the BP and BST problems. We present an improved randomized algorithm for these problems whose running time is $O(m\beta(m, n))$. As mentioned, for $m \geq n \log^{(k)} n$, the expected running time is $O(m)$, i.e., best possible. We also show that BP and BST are equivalent under randomized reductions. As $\log^* n$ and $\beta(m, n)$ are both extremely slowly growing functions, our improved bound has no practical importance. We believe, however, that understanding the exact complexity of fundamental problems such as BP and BST is an important endeavor.

Andersson et al. [1] used the technique of Gabow and Tarjan [16] to obtain an $O(m\beta(m, k))$-time algorithm for solving DGGs, where k is the number of terminals. We show, perhaps surprisingly, that the DGG problem is *easier* than the BP and BST problems. By combining the viewpoints of the two players, we obtain a simple deterministic $O(m)$-time algorithm for solving DGGs.

1.1 Bottleneck Paths and Bottleneck Spanning Trees

Both the BP and BST in *directed* graphs are well-motivated problems that were studied by many researchers. The BP problem, for example, models the problem of finding a route from one city to another minimizing the maximum distance travelled between two consecutive cities. The equivalent problem of finding a max-min path from s to t is the problem of finding a maximum *capacity* path in a flow network. Edmonds and Karp [10] obtained an

[1] Here $A \uplus B$ stands for the *disjoint union* of A and B, i.e., the union $A \cup B$ where is it assumed that $A \cap B = \emptyset$.

efficient, though not strongly polynomial, maximum flow algorithm that repeatedly computes maximum capacity augmenting paths.

Edmonds and Fulkerson [9] introduced the *threshold method* that yields a simple $O(m \log n)$-time algorithm for the BP problem. They also note that the min-max s-t path problem has a max-min *dual*, i.e., the problem of finding a s-t *cut* whose minimal edge weight is maximized.

Dijkstra's [7] single-source shortest paths algorithm can be easily adapted to solve the BP and BST problems. The resulting algorithm solves, in fact, the *Single-Source Bottleneck Paths* (SS-BP) problem in which a min-max path is sought from the source s to any vertex of the graph. It is easy to see that the tree of min-max paths returned by the algorithm is also a min-max spanning tree. If Fibonacci heaps [13] are used, an $O(m + n \log n)$-time algorithm is obtained.

Gabow and Tarjan [16] obtained an improved algorithm for the BP and BST problems that runs in $O(m \log^* n)$ time, where $\log^* n = \min\{k \geq 1 \mid \log^{(k)} n \leq 1\}$, where $\log^{(1)} n = \log n$ and $\log^{(k)} n = \log \log^{(k-1)} n$, for $k > 1$. We improve on this 25 year old result and obtain a randomized algorithm whose running time is $O(m\beta(m,n))$, where $\beta(m,n) = \min\{k \geq 1 \mid \log^{(k)} n \leq \frac{m}{n}\} \leq \log^* n - \log^*(m/n) + 1$. In particular, if $m \geq n \log^{(k)} n$, for any constant k, the expected running time of the new algorithm is $O(m)$. Our algorithm, as that of Gabow and Tarjan, works in the *comparison model*, i.e., the only operations it performs on edge weights are pairwise comparisons. We also observe that in the word-RAM model, where comparisons are not the only operations allowed on edge weights, both problems can be solved deterministically in $O(m)$ time.

The BP problem can be easily reduced to the BST problem. We give the first randomized reduction in the other direction, showing that the BP and BST problems are essentially equivalent.

Punnen [23] shows that for a wide class of bottleneck problems, if the problem can be solved in $O(f(m))$ time when the weights are given in sorted order, then the problem can be solved in $O(f(m) \log^* m)$-time, when the weights are not given in sorted order.

The BP, BST and SS-BP can also be solved easily in $O(m)$ time if the input graph is *acyclic*.

All the results stated above are for directed graphs. For *undirected* graphs, both the BP and BST are much easier. Camerini [5] gave a simple $O(m)$-time algorithm for BP and BST problems in undirected graphs. Furthermore, if T is a *Minimum Spanning Tree* (MST) of an undirected graph $G = (V, E)$, i.e., a spanning tree such that the *sum* of its edge weights is minimal, then for any $s, t \in V$, the unique path in T between s and t is a min-max path between s and t. (See, e.g., Hu [19].) An MST of an undirected graph can be found in $O(m)$ expected time (Karger, Klein and Tarjan [21]), or deterministically in $O(m\alpha(m,n))$ time (Chazelle [6]). [2]

The *All-Pairs Bottleneck Paths* (AP-BP) problem, in which we want to find a bottleneck path for every pair of vertices in a weighted directed graph can be easily solved in $O(mn)$ time by first sorting all edge weights and then running a linear time SS-BP algorithm from each vertex. In dense enough graphs, faster algorithms can be obtained using fast matrix multiplication. Vassilevska, Williams and Yuster [25] showed that the AP-BP problem can be reduced to the problem of computing $(\max, \min)$ products and gave an algorithm whose running time is $O(n^{2+\omega/3})$, which is $O(n^{2.80})$, for computing such products. Here $\omega < 2.38$

[2] Chazelle's algorithm improves on $O(m\beta(m,n))$- and $O(m \log \beta(m,n))$-time algorithms of Fredman and Tarjan [13] Gabow et al. [15]. Is this an indication that similar improvements are also possible for the BP and BST problems?

is the matrix multiplication exponent. Duan and Pettie [8] obtained a faster algorithm for computing $(\max, \min)$ products whose running time is $O(n^{(3+\omega)/2})$, which is $O(n^{2.69})$, matching Matoušek's [22] fastest known algorithm for computing *dominance* products. For vertex weighted graphs, a faster running time of $O(n^{2.58})$ was previously obtained by Shapira, Yuster and Zwick [24].

1.2 Deterministic Graphical Games

Many turn-based 2-player zero-sum games can be modeled using finite *game trees*. The game starts at the root. At even levels, the first player chooses an edge to one of the children of the current vertex, at odd levels the second player chooses the edge. Each leaf has a *payoff* associated with it, which is the amount the first player has to pay the second player. The *value* of such a game can be easily determined by starting at the leaves and alternatingly computing the minimum or the maximum of the values of the nodes at the lower level.

It is natural to generalize game trees into *game graphs*, yielding exactly the *Deterministic Graphical Games* (DGGs) defined above. A game can now return to a position visited before, as may happen, for example, in *chess*. The main difference between game trees and game graphs is that *infinite* plays are now possible. An infinite play is considered to be a *draw*, i.e., no payment, or equivalently a payment of 0, is made. Such game graphs are implicit in Zermelo's [27] classical, but slightly incomplete, proof that each position in chess has a definite value. For more technical and historical details, see Washburn [26] and Andersson et al. [1].

A *strategy* for a player in a DGG is a rule for selecting the next edge to play in each situation. A general strategy may depend on the full *history* of the play and may be *randomized*. It can be shown, however, that in DGGs, both players may restrict themselves without loss to *pure positional strategies*, i.e., deterministic strategies that depend only on the current position. Each vertex v in a DGG has a *value* $val(v)$. Player 0 has a (pure positional) strategy that guarantees that the outcome of the game will be at most $val(v)$, no matter what strategy is used by player 1. Similarly, player 1 has a (pure positional) strategy that guarantees that the outcome of the game will be at least $val(v)$, no matter what strategy is used by player 0. Such strategies are said to be *optimal* from v. Both players actually have pure positional strategies that are optimal from all vertices.

Let $G = (V, E)$ be a DGG and let t be the terminal with the largest payoff. The set of vertices from which player 1 can force the game to end in t can be easily found in linear time using an alternating backward search from t, also known as *retrograde analysis*. (See details in [1] or in Section 6.) Thus, if the payoffs are given in sorted order, it is easy to find the values of all vertices, and optimal strategies for both players, by "peeling" the terminals one by one, in decreasing order.

Andersson et al. [1] used the technique of Gabow and Tarjan [16] to obtain an $O(m\beta(m, k))$-time algorithm for finding the value and optimal strategies for a specific start vertex s in a DGG with m edges and k terminals. We obtain a simple deterministic $O(m)$-time algorithm for the same problem.

The best known algorithm for finding the values, and corresponding optimal strategies, of *all* vertices of a DGG in the comparison model runs in $O(m + k \log k)$ time. The algorithm begins by sorting the payoffs in $O(k \log k)$ time, and then finds all values in $O(m)$ additional time.

When the payoffs are moved from terminals to edges or non-terminal vertices, and the sequence of resulting payoffs is accumulated in some way, we obtain *Mean Payoff Games* (MPGs) and *Discounted Payoff Games* (DPGs) [11, 17, 28, 2, 3] or *Parity Games* (PGs)

[12, 20]. These games are much harder than DGGs. No polynomial time algorithms are known for their solution.

1.3 Organization of the Paper

In the next section we review the classical $O(m \log^* n)$-time algorithm of Gabow and Tarjan [16] for the Bottleneck Path (BP) and Bottleneck Spanning Tree (BST) problems. In Section 3 we present our improved algorithm. In Section 4 we prove the equivalence of the BP and BST problems. In Section 5 we observe that both BP and BST can be solved deterministically in $O(m)$ time in the word-RAM model. In Section 6 we present a deterministic $O(m)$-time algorithm, in the comparison model, for solving Deterministic Graphical Games (DGGs). This result is independent of the results of the previous sections. The main results of the paper are in Sections 3 and 6. We conclude in Section 7 with some open problems.

2 The $O(m \log^* n)$-time Algorithm of Gabow and Tarjan

In this section we sketch the $O(m \log^* n)$-time algorithm of Gabow and Tarjan [16] for the BST problem in weighted directed graphs. The algorithm can be easily modified to solve the BP problem. We also note that the algorithm performs only $O(m)$ comparisons.

Gabow and Tarjan [16] first observe that if the edge weights are small integers, i.e., $w : E \to \{0, 1, \ldots, k\}$, then the BST problem can be easily solved in $O(m + k)$ time. We first use *bucket sort* to sort the outgoing edges of each vertex in non-decreasing order and then use an *incremental search*. The search starts at the source vertex s and finds all vertices reachable from s using edges of weight 0. If all vertices are reached, we are of course done. Otherwise, we resume the search from all vertices reached allowing now edges of weights 0 and 1, and so on. It is not difficult to check that this can be implemented in $O(m + k)$ time.

Assume now that the edge weights are real numbers. If the edge weights are given to us in sorted order, we could easily replace them by integer weights from $\{1, 2, \ldots, m\}$, depending on their *rank*, and use the algorithm above to solve the problem in $O(m)$ time. However, sorting the edge weights in the comparison model requires $\Omega(m \log n)$ comparisons and time. The challenge is solving the problem *without* sorting all the edge weights.

Let $G = (V, E)$ be an instance of the BST problem were $E = E_0 \cup F$ such that all edges of $E_0 \subseteq E$ are known to have weights below the bottleneck weight and such that the weight of each edge of E_0 is smaller than the weight of each edge of F. (Possibly $E_0 = \emptyset$.) For simplicity, assume that all edges of F have distinct weights. By repeatedly finding medians [4], using $O(|F| \log k)$ time and comparisons, we can partition F into k subsets $E_1, \ldots, E_k$ of almost equal size such that the weight of all edges in E_j are smaller than the weights of all edges in E_{j+1}, for $j = 0, 1, \ldots, k-1$. We can now replace the edge weights of all edges in E_j by j, for $j = 0, 1, \ldots, k$, and run the linear time algorithm above. If the answer we get is i, then the bottleneck edge belongs to E_i. Thus, all the edges of $E_{i+1} \cup \cdots \cup E_k$ are not needed, and all edges of $E_0 \cup \cdots \cup E_{i-1}$ are now known to have weights which are below the bottleneck weight. We are thus left with a smaller instance $G = (V, E')$ where $E' = E'_0 \cup F'$, $E'_0 = E_0 \cup \cdots \cup E_{i-1}$ and $F' = E_i$. Note that $|F'| \leq |F|/k + 1$.

We can now iterate the above step. Let $F^{(j)}$ be the edge set known to contain the bottleneck edge after j iterations and let $m_j = |F^{(j)}|$. Initially $F^{(0)} = E$ and $m_0 = m$. When $m_j = 1$, we are done. Let k_j be the number of sets to which $F^{(j)}$ is partitioned. Thus $m_{j+1} \leq m_j/k_j$, and consequently $m_j \leq m/(k_0 k_1 \ldots k_{j-1})$. The number of comparisons used in the j-th iteration is therefore $O(m_j \log k_j) = O((m/(k_{j-2}k_{j-1})) \log k_j)$. (We let $k_{-2} = k_{-1} = 1$.) The time used in the j-th iteration is $O(m)$. We choose $k_0 = 2$ and

$k_j = 2^{k_{j-1}}$, for $j > 0$. After at most $\log^* n$ iterations we get $m_j = 1$. The total time spent is $O(m \log^* n)$. The number of comparisons used in the j-th iteration is $O(m_j \log k_j) = O((m/(k_{j-2}k_{j-1})) \log k_j) = O(m/k_{j-2})$ and the total number of comparisons performed is thus $O(m)$.

3 An $O(m\beta(m,n))$-time Algorithm for Bottleneck Paths and Trees

Let $G = (V, E)$ be the input graph, $w : E \to \mathbb{R}$ be a weight function defined on its edges, and let $s \in V$ be a source vertex. Let $m = |E|$ and $n = |V|$. For simplicity, we assume that all edge weights are distinct. In the previous section we saw that in $O(m \log k)$ time we can partition E into $E = E_1 \uplus E_2 \uplus \cdots \uplus E_k$ such that $E_1, E_2, \ldots, E_k$ have roughly the same size and such that all edges of E_j have weight smaller than all edges of E_{j+1}, for $j = 1, 2, \ldots, k-1$. In $O(m)$ time, we can then find the set E_i that contains the bottleneck edge.

To obtain the improved algorithm, we adopt a slightly different approach. Let $\lambda_1 < \lambda_2 < \cdots < \lambda_k$ be k *thresholds* and let $\lambda_0 = -\infty$ and $\lambda_{k+1} = \infty$. The thresholds naturally partition E into $E = E_0 \uplus E_1 \uplus \cdots \uplus E_k$ such that $E_i = \{e \in E \mid \lambda_i \leq w(e) < \lambda_{i+1}\}$. Explicitly computing this partition requires $\Omega(m \log k)$ time. We show, however, that we can compute the index i of the set E_i that contains the bottleneck edge in $O(m + nk)$ time, or even $O(m + n \log k)$ time, using a simple deterministic algorithm that does not explicitly compute the partition.

To obtain our improved algorithm, we set the k thresholds to the weights of k *randomly* chosen edges of the graph. We then compute the set E_i that contains the bottleneck edge and revert to the standard algorithm. The exact details will follow after describing the simple algorithm for locating the part that contains the bottleneck edge.

3.1 Locating the Bottleneck Weight Among k Thresholds

Let $G = (V, E)$ be a weighted directed graph, $w : E \to \mathbb{R}$ a weight function defined on its edges, $s \in V$ a source vertex, and let $-\infty = \lambda_0 < \lambda_1 < \cdots < \lambda_k < \lambda_{k+1} = \infty$ be arbitrary thresholds. Let $w^*(G)$ be the bottleneck edge weight of G. We begin by describing a simple $O(m + nk)$ time algorithm, called Locate, for computing the index i such that $\lambda_i \leq w^*(G) < \lambda_{i+1}$. For concreteness, we consider the BST problem. The details for the BP problem are almost identical.

The algorithm is composed of $k + 1$ phases. In the i-th phase, where $i = 0, 1, \ldots, k$, the algorithm finds all vertices $u \in V$ for which there is a directed path from s to u in G all whose edges have weights that are strictly smaller than λ_{i+1}. For every vertex u we maintain a value $d[u]$ such that a path from s to u all whose edge weights are at most $d[u]$ was already discovered. Initially $d[s] = -\infty$ while $d[u] = \infty$ for every $u \in V \setminus \{s\}$. We maintain the invariant that at the beginning of the i-th phase, for $i = 0, 1, \ldots, k$, we have $d[u] < \lambda_i$ for every $u \in V$ for which there is a path from s to u all whose edge weights are smaller than λ_i. In the i-th phase itself, we identify all vertices u with $d[u] < \lambda_{i+1}$ and examine all their outgoing edges. If $(u, v) \in E$, we let $\bar{w}(u, v) = \max\{d[u], w(u, v)\}$. If $\bar{w}(u, v) < d[v]$, we let $d[v] \leftarrow \bar{w}(u, v)$. If at the end of the i-th phase $d[u] < \lambda_{i+1}$ for all $u \in V$, we know that $\lambda_i \leq w^*(G) < \lambda_{i+1}$. The complexity of the algorithm is $O(m + nk)$ as we examine each edge at most once and each vertex at most k times. A more precise description follows.

In addition to the phase number i and the values $d[u]$, for every $u \in V$, the algorithm maintains two sets of vertices $A, B \subseteq V$. The set A contains all vertices $u \in V$ with $d[u] < \lambda_{i+1}$ whose outgoing edges were not examined yet. The set B contains all vertices

Algorithm Locate$(G = (V, E), w, s, (\lambda_1, \ldots, \lambda_k))$

```
λ_0 ← −∞ ; λ_{k+1} ← ∞
foreach v ∈ V do d[v] ← ∞
d[s] = −∞ ; A ← ∅ ; B ← V
for i ← 0 to k do
    foreach u ∈ B do
        if d[u] < λ_{i+1} then Move(v, B, A)
    while A ≠ ∅ do
        u ← Extract(A)
        foreach (u, v) ∈ E do
            w̄ ← max{d[u], w(u, v)}
            if w̄ < d[v] then
                d[v] ← w̄
                if d[v] < λ_{i+1} and v ∈ B then
                    Move(v, B, A)
    if B = ∅ then return i
```

Figure 1 Locating the bottleneck weight among k thresholds.

$u \in V$ for which $d[u] \geq \lambda_{i+1}$. Initially $i = -1$, $A = \emptyset$ and $B = V$. At the beginning of the i-th phase, for $i = 0, 1, \ldots, k$, the algorithm examines all vertices of B and moves to A each vertex u for which $d[u] < \lambda_{i+1}$. As long as A is not empty, the algorithm removes an arbitrary vertex u from A and scans all its outgoing edges as above. For every outgoing edge $(u, v) \in E$ it lets $\bar{w}(u, v) = \max\{d[u], w(u, v)\}$. If $\bar{w}(u, v) < d[v]$, it lets $d[v] \leftarrow \bar{w}(u, v)$. If $d[v] < \lambda_{i+1}$ and $v \in B$, then v is moved from B to A. The i-phase ends when A is empty. If B is also empty, the algorithm returns i and terminates. Otherwise, it moves on to the $(i + 1)$-st phase.

Pseudo-code of Locate is given in Figure 1 . Function Move(v, B, A) moves v from B to A while Extract(A) removes and returns an arbitrary item of A. With a simple linked-list implementation, both these operations take constant time.

▶ **Theorem 1.** *Algorithm* Locate *returns an index i such that $\lambda_i \leq w^*(G) < \lambda_{i+1}$. Its running time is $O(m + nk)$.*

Proof. The correctness of the algorithm follows from the invariant stated above: At the end of the i-phase, for every $u \in V$, if there is a path in G from s to u all whose edges have weights strictly smaller than λ_{i+1}, then $d[u] < \lambda_{i+1}$. The invariant holds vacuously for $i = -1$.

Suppose that the invariant holds at the end of the $(i - 1)$-st phase. Suppose, for the sake of contradiction, that the invariant does not hold at the end of the i-th phase. Namely, suppose that there is a path $s = u_0 \to u_1 \to \cdots \to u_k$ in G such that $(u_j, u_{j+1}) \in E$ and $w(u_j, u_{j+1}) < \lambda_{i+1}$, for $j = 0, 1, \ldots, k - 1$, but $d[u_k] \geq \lambda_{i+1}$. Let u_ℓ be the first vertex on the path for which $d[u_\ell] \geq \lambda_{i+1}$. As $d[u_0] = -\infty < \lambda_{i+1}$, we have $\ell \geq 1$. By definition $d[u_{\ell-1}] < \lambda_{i+1}$. As $d[u_{\ell-1}] < \lambda_{i+1}$, $u_{\ell-1}$ must have been moved to A either in the i-th phase, or before. After the edge $(u_{\ell-1}, u_\ell)$ is examined, at or before the i-th phase, we have $d[u_\ell] \leq \bar{w}(u_{\ell-1}, u_\ell) = \max\{d[u_{\ell-1}], w(u, v)\} < \lambda_{i+1}$, a contradiction.

Thus, if $\lambda_i \leq w^*(G) < \lambda_{i+1}$, then at the end of the i-th phase B is empty and the

algorithm terminates. The algorithm cannot terminate before the i-th phase as there is at least one vertex u for which there is no path all whose edges have weights strictly smaller than λ_i. For such a vertex we must have $d[u] \geq \lambda_i$. Thus, u must be in B until the beginning of phase i.

The running time of the algorithm is $O(m + nk)$ as each edge is examined at most once, and each vertex is examined at most once at each one of the k iterations. ◀

Although the $O(m + nk)$ running time of LOCATE is sufficient for our purposes, the running time can be reduced to $O(m + n \log k)$ using a *lazy binary search*. Details will appear in the full version of the paper.

3.2 A Randomized $O(m\beta(m, n))$-time Algorithm

Let $r \geq 1$. Choose $k = \log^{(r)} n$ *random* edges $e_1, e_2, \ldots, e_k$ from E and sort their edge weights so that $w(e_1) < w(e_2) < \cdots < w(e_k)$. Let $\lambda_i = w(e_i)$, for $i = 1, 2, \ldots, k$, and $\lambda_0 = -\infty$, $\lambda_{k+1} = \infty$. (Sorting the edge weights takes $O(k \log k)$ time, which will be negligible.) We now use LOCATE to find the index i for which $\lambda_i \leq w^*(G) < \lambda_{i+1}$. This takes only $O(m+n \log^{(r)} n)$ time. In $O(m)$ further time, we can compute the sets $E_0 = \{e \in E \mid w(e) < \lambda_i\}$ and $F = \{e \in E \mid \lambda_i \leq w(e) < \lambda_{i+1}\}$. The next lemma shows that the expected size of $F = E_i$ is $O(m/k)$.

▶ **Lemma 2.** *Let $f \in E$ be a fixed edge, and let $e_1, e_2, \ldots, e_k$ be k random edges from E such that $w(e_1) < w(e_2) < \cdots < w(e_k)$. Let $\lambda_i = w(e_i)$, for $i = 1, 2, \ldots, k$, and let $\lambda_0 = -\infty$ and $\lambda_{k+1} = \infty$. Let $E_i = \{e \in E \mid \lambda_i \leq w(e) < \lambda_{i+1}\}$, for $i = 0, 1, \ldots, k$, and let j be such that $f \in E_j$. Then, $\mathbb{E}[|E_j|] \leq 2m/k$, where $m = |E|$.*

Proof. Let $f_1, f_2, \ldots, f_m$ be the edges of E sorted according to weight, i.e., $w(f_1) < w(f_2) < \cdots < w(f_m)$. Let $f = f_r$, where $1 \leq r \leq m$. The probability of a given edge f_i to be one of the k randomly chosen edges, given that no edge from a set F' is in the sample, is $k/(m - |F'|) \geq k/m$. Examine the edges $f_r, f_{r+1}, \ldots, f_m$ one by one until finding an edge from the sample, or until reaching the last edge. As the probability of each inspected edge to be in the sample is at least k/m, the expected number of edges inspected is at most m/k. Similarly, the expected number of edges among $f_{r-1}, f_{r-2}, \ldots, f_1$ that need to be inspected until finding an edge from the sample, or until reaching the first edge, is also at most m/k. ◀

It is not difficult to extend the proof of Lemma 2 to show that the size of F is $O(m/k)$ with high probability. For our purposes it is enough to rely on Markov's inequality to infer that the probability that $|F| \geq 4m/k$ is at most $1/2$. If $|F| \geq 4m/k$, we simply choose a new sample. The expected number of samples needed is at most 2.

After running LOCATE with $k = \log^{(r)} n$ *random* edges, we get that $|F| \leq 4m/\log^{(r)} n$. We now run one iteration of the algorithm of Gabow and Tarjan from the previous section with $k = \log^{(r-1)} n$. The running time is $O(m + |F| \log k) = O(m)$ and the size of F is reduced to $O(m/\log^{(r-1)} n)$. In at most $r - 1$ additional iterations, we can thus reduce the size of F to $O(m/\log n)$, at which point we can afford to sort F and find the bottleneck edge in $O(m)$ time. The total running time of the algorithm is therefore $O(rm + n \log^{(r)} n)$. If we choose $r = \beta(m, n)$, we get an expected running time of $O(m\beta(m, n))$. As mentioned, it is easy to see that a running time of $O(m\beta(m, n))$ is obtained not only in expectation, but also in very high probability.

▶ **Theorem 3.** *The Bottleneck Path (BP) and Bottleneck Spanning Tree (BST) problems can be solved in the comparison model in $O(m\beta(m, n))$ expected time.*

4 Equivalence of Bottleneck Paths and Bottleneck Spanning Trees

In this section we show that if there is an $O(f(m,n))$ time algorithm for the BST problem, then there is also an $O(f(m,n))$ time algorithm for the BP problem, and vice versa. The reduction from BP to BST is immediate. The reduction from BST to BP is slightly more complicated, requires randomization and needs some mild assumptions on $f(m,n)$, which are satisfied if $f(m,n) = m+n$.

▶ **Lemma 4.** *If there is an $f(m,n)$-time algorithm for the Bottleneck Spanning Tree (BST) problem, then there is an $O(f(m+n,n))$-time algorithm for the Bottleneck Path (BP) problem.*

Proof. Given an instance $G=(V,E)$ of the BP problem, with source s and target t, simply add edges (t,v), for every $v \in V$, of weight $-\infty$. The path from s to t in a bottleneck spanning tree of the resulting graph is a bottleneck path from s to t. ◀

Before describing the reduction in the opposite direction, we need to introduce some more notation. If $G=(V,E)$ is a weighted graph with source s, we let $w^*(v)$ be the weight of the bottleneck edge on a min-max path from s to v in G. We then define $E(v) = \{e \in E \mid w(e) \leq w^*(v)\}$ and let $S(v)$ be the set of vertices reachable from s in $(V,E(v))$. Finally, we let $E_{in}(S(v))$ be the set of edges that enter vertices of $S(v)$. We now have the following simple probabilistic lemma.

▶ **Lemma 5.** *Let $G=(V,E)$ be a weighted graph with source s.*
(i) *If v is a randomly chosen vertex, then $\mathbb{P}[|S(v)| \geq \frac{n}{2}] \geq \frac{1}{2}$.*
(ii) *If (u,v) is a randomly chosen edge, then $\mathbb{P}[|E_{in}(S(v))| \geq \frac{m}{2}] \geq \frac{1}{2}$.*

Proof. (i) Let $v_1, v_2, \ldots, v_n$ be an ordering of the vertices such that $w^*(v_1) \leq w^*(v_2) \leq \cdots \leq w^*(v_n)$. Note that $S(v_i) \supseteq \{v_1, v_2, \ldots, v_i\}$. Thus, if v is among $\{v_{\lceil n/2 \rceil}, \ldots, v_n\}$, then $|S(v)| \geq n/2$, and this happens with a probability of at least $1/2$.

(ii) Let d_i be the in-degree of v_i. Let k be the minimal index for which $\sum_{i=1}^{k} d_i \geq \frac{m}{2}$. (In particular, we have $\sum_{i=1}^{k-1} d_i < \frac{m}{2}$. Let (u,v) be a random edge. If $v = v_i$, and $i \geq k$, then $E_{in}(S(v)) \geq m/2$. This happens with a probability of at least $\frac{1}{m}\sum_{i=k}^{n} d_i = \frac{1}{m}(m - \sum_{i=1}^{k-1} d_i) \geq \frac{1}{2}$. ◀

▶ **Lemma 6.** *If there is an $f(m,n)$-time algorithm for the Bottleneck Path (BP) problem, then there is a randomized algorithm whose expected running time is $O(\sum_{i\geq 0} f(\frac{m}{2^i}, \frac{n}{2^i}))$ for the Bottleneck Spanning Tree (BST) problem.*

Proof. Let $G=(V,E)$ be an instance of the BST problem with source s. Choose a random vertex $v \in V$ and solve the BP problem with $t = v$. The bottleneck edge weight $w^*(v)$ returned is clearly a lower bound on the bottleneck edge weight in a spanning tree. If $S(v) = V$, we are done. Otherwise, all vertices of $S(v)$ may be replaced by a new source $\bar{s}$, and all edges of $E_{in}(S(v))$, which now enter $\bar{s}$, may be removed. By Lemma 5(i), $|S(v)| \geq \frac{n}{2}$ with a probability of at least $\frac{1}{2}$. If this is not the case, we can repeat this step. (This is not really required, but it slightly simplifies the analysis.) The expected number of repetitions is constant. We are now left with an instance with at most $\frac{n}{2}$ vertices. To reduce the number of edges we now sample a random edge $(u,v) \in E$ and solve the BP problem with $t = v$. If $S(v) = V$, we are again done. Otherwise, we can again replace $S(v)$ by a new source $\bar{s}$ and remove all the edges of $E_{in}(S(v))$. By Lemma 5(ii), $|E_{in}(S(v))| \geq \frac{m}{2}$ with a probability of at least $\frac{1}{2}$. If this is not the case, we can repeat this step. We continue in this way, alternatingly sampling vertices and edges. The total expected running time is then $O\big(\sum_{i\geq 0}\big(f(\frac{m}{2^i}, \frac{n}{2^i})) + f(\frac{m}{2^i}, \frac{n}{2^{i+1}})\big)\big) = O\big(\sum_{i\geq 0} f(\frac{m}{2^i}, \frac{n}{2^i})\big)$. ◀

If we are only interested in a time bound in terms of m, we can do only edge sampling steps. The running time is then $O\big(\sum_{i\geq 0} f(\frac{m}{2^i})\big)$. If $\frac{f(m)}{m}$ is monotone non-decreasing, the resulting running time is $O(f(m))$.

5 Bottleneck Paths and Trees in the Word-RAM Model

On the word-RAM with word length $w \geq \log n$, we can use a constant number of levels of *fusion nodes* (Fredman and Willard [14]) to split the m edge weights into $k = \log n$ sets $E_1, E_2, \ldots, E_k$ of size $O(m/\log n)$ such that the weights of all edges in E_i are smaller than the weights of all edges in E_{i+1}, for $i = 1, \ldots, k-1$. This requires only $O(m)$ time. Using $O(m)$ further time we can use the simple algorithm of Section 2 to find the subset containing the bottleneck weight. We can afford to completely sort this subset using a standard comparison-base algorithm, as this takes only $O((m/\log n)\log n) = O(m)$ time. As the relevant edge weights are now sorted, we can use the algorithm of Section 2 again to completely solve the problem, using only $O(m)$ additional time.

Using a word-RAM algorithm of Han and Thorup [18], we can actually split the edge weights into $\sqrt{m}$ sets of size $O(\sqrt{m})$, again in $O(m)$ time.

6 An $O(m)$-time Algorithm for Deterministic Graphical Games

A *Deterministic Graphical Game* (DGG) is composed of directed graph $G = (V, E)$, a partition $V = V_0 \uplus V_1 \uplus T$, an initial vertex $s \in V_0 \cup V_1$ and a payoff function $p : T \to \mathbb{R}$. For the exact definition refer to the Introduction and Section 1.2. We begin with the following folklore lemma which is also used in Andersson et al. [1].

▶ **Lemma 7.** *Let $G = (V, E)$ be a DGG with a unique target t of payoff 1. Let W_1 be the set of vertices of value 1, $E_1 = \{(u, v) \in E \mid v \in W_1\}$ and $m_1 = |E_1|$. Then, there is a deterministic algorithm with running time $O(m_1)$ for computing W_1 and for constructing a strategy for player 1 that ensures value 1 from all the vertices of W_1. The set of vertices $W_0 = V \setminus W_1$ of value 0 and an optimal strategy for player 0 from all vertices can be found, if required, in $O(n)$ additional time.*

Proof. We use a backward search from t to find all the vertices in G whose value is 1. The value of all the remaining vertices is 0. Let $W_1 \leftarrow \{t\}$ and $A \leftarrow \{t\}$. While A is not empty, extract a vertex $v \in A$. For every incoming edge $(u, v) \in E$, do the following. If $u \in V_1$, or (u, v) is the last remaining outgoing edge of u, then add u to W_1 and A and set $\pi(u) \leftarrow v$. Otherwise, simply remove (u, v) from the graph. We refer to handling such an incoming edge (u, v) as a *basic step*. When the algorithm terminates, W_1 is the set of all vertices of value 1. The running time of the algorithm is $O(m_1)$ as each incoming edge of a vertex of W_1 is examined exactly once. The strategy that from each vertex $u \in V_1 \cap W_1$ chooses the edge $(u, \pi(u))$ is an optimal strategy for player 1. (The choice at vertices of $V_1 \setminus W_1$ may be arbitrary.) The set $W_0 = V \setminus W_1$ can be computed in $O(n)$ time. An optimal strategy for player 0 is obtained by choosing for each vertex $u \in V_0 \cap W_0$ the first remaining outgoing edge of u. (There must be at least one such edge and it must lead to a vertex of W_0.) Constructing such a strategy also requires only $O(n)$ additional time. ◀

If the terminals $t_1, t_2, \ldots, t_k$ are given in sorted order, i.e., $p(t_1) < p(t_2) < \ldots p(t_k)$, we can apply the algorithm above repeatedly to find the values of all vertices. To find the set of vertices W_k of value $p(t_k)$ we add self-loops to terminals $t_1, t_2, \ldots, t_{k-1}$ and move them from T to either V_0 or V_1, so that t_k is the only remaining terminal, and run the algorithm of

Lemma 7. We then remove the vertices of W_k and all their incoming edges, find all vertices whose value is $p(t_{k-1})$, and so on. The total running time is $O(m)$, as we do not examine again edges that were removed from the graph. If some of the payoffs are negative, we stop when we reach the last positive payoff and then start in a symmetric manner from the smallest negative payoff. The remaining vertices are the vertices of value 0.

If the payoffs of $t_1, t_2, \ldots, t_k$ are not given to us in sorted order, we can sort them in $O(k \log k)$ time and then run the linear time algorithm above. The running time is then $O(m + k \log k)$. This is the fastest known algorithm for finding the values of *all* vertices.

Andersson et al. [1] gave an $O(m\beta(m,k))$-time algorithm for finding the value of a specific start vertex s. Their algorithm is similar to the algorithm of Gabow and Tarjan [16] for the BP and BST problems sketched in Section 2. We obtain an improved deterministic $O(m)$-time algorithm. The key ingredient in our $O(m)$-time algorithm is the following simple lemma.

▶ **Lemma 8.** *Let $G = (V, E)$ be a DGG such that $V = V_0 \cup V_1 \cup T$ where $T = \{t_1, t_2\}$ and $0 < p(t_1) < p(t_2)$. Let W_i be the vertices of G whose value is $p(t_i)$, let $E_i = \{(u, v) \in E \mid v \in W_i\}$, and $m_i = |E_i|$, for $i = 1, 2$. Assume that $W_1 \cup W_2 = V$, i.e., no vertex has value 0. Then, there is a deterministic algorithm for computing either W_1 or W_2 in $O(\min\{m_1, m_2\})$ time.*

Proof. We run in *parallel* two instances of the algorithm of Lemma 7, one on a game obtained by adding a self-loop to t_1, which is no longer a terminal, and one on a game obtained by adding a self-loop to t_2 and replacing the roles of the two players. The first instance is trying to construct W_2 while the second is trying to construct W_1. We alternatingly perform basic steps in these two instances. When one of these instances finishes, we stop the other. The running time of the resulting algorithm is clearly $O(\min\{m_1, m_2\})$. ◀

Using Lemma 8 we obtain the main result of this section.

▶ **Theorem 9.** *There is a deterministic $O(m)$-time algorithm for finding the value and optimal strategies for both players in a Deterministic Graphical Game (DGG) with a given start vertex.*

Proof. Let $G = (V, E)$ be a DGG, where $V = V_0 \cup V_1 \cup T$, $s \in V_0 \cup V_1$ is the start vertex, and $p : T \to \mathbb{R}$ is the payoff function. We begin by describing an algorithm for finding the value of s.

We first perform a preprocessing step that determines for each vertex $u \in V$ whether its value $val(u)$ is positive, zero, or negative. To find all vertices of positive value, we merge all terminals of positive payoff into a single terminal, give this terminal a payoff of 1, and run the algorithm of Lemma 7. Similarly, we can find all vertices with negative values. The remaining vertices have value 0. If $val(s) = 0$, we are done. If $val(s) > 0$, we can remove from the game all vertices with non-positive value and all edges entering them. Similarly, if $val(s) < 0$, we can remove from the game all vertices with non-negative value and all edges entering them. For concreteness, we assume that $val(s) > 0$. The case $val(s) < 0$ is analogous.

Assume therefore that $G = (V, E)$ is a DGG for which $val(u) > 0$, for every $u \in V$, with $|T| = k$. We assume, for simplicity, that all payoffs are distinct. This assumption can be easily removed. Find the *median* of the payoffs and split the terminal set T into two subsets T_1 and T_2 of sizes $\lfloor k/2 \rfloor$ and $\lceil k/2 \rceil$ such that for every $t_1 \in T_1$ and $t_2 \in T_2$ we have $p(t_1) < p(t_2)$. Merge all the terminals in T_i into a new terminal t_i with payoff i, for $i = 1, 2$.

Let W_i be the set of vertices in the new game whose values are i, $E_i = \{(u, v) \in E \mid v \in W_i\}$, and $m_i = |E_i|$, for $i = 1, 2$.

We now run the algorithm of Lemma 8 and in $O(\min\{m_1, m_2\})$ time construct either W_1 or W_2. If the construction of W_1 is complete and $s \in W_1$, or the construction of W_2 is complete but $s \notin W_2$, we know that $val'(s) = 1$, otherwise $val'(s) = 2$, where $val'(s)$ is the value of s in the new game. If $val'(s) = 1$, we construct W_2 and E_2 in $O(m_2)$ time. (If the construction of W_2 was not complete, we let $W_2 \leftarrow V \setminus W_1$ and then compute E_2.) We can now remove all edges of E_2 and all terminals of T_2 from the original game G without changing $val(s)$. Similarly, if $val'(s) = 2$, we construct W_1 and E_1 in $O(m_1)$ time and remove all edges of E_1 and all terminals of T_1 from G. In both cases, in $O(m' + k')$ time we removed m' edges and k' terminals from the game.

We repeat the process until we are left with only one terminal whose payoff is then the value of s. As the running time of each iteration is proportional to the number of edges and terminals removed from the graph, the total running time of the algorithm is $O(m + k) = O(m)$.

Once $val(s)$ is known, it is easy to find optimal strategies for both players from s. To find an optimal strategy for player 1, we merge all terminals with payoffs at least $val(s)$ into a new terminal. To all terminals with payoffs less than $val(s)$ we add a self-loop, so that they are not terminals any longer. An optimal strategy for player 1 from s in this new game, which can be found in $O(m)$ time using the algorithm of Lemma 7, is also an optimal strategy for player 1 in the original game. An optimal strategy for player 0 from s can be found in a similar manner. ◀

7 Concluding Remarks and Open Problems

We presented an improved randomized algorithm for the Bottleneck Path (BP) and Bottleneck Spanning Tree (BST) problems with an expected running time of $O(m\beta(m, n))$ and a deterministic $O(m)$-time algorithm for solving a Deterministic Graphical Game (DGG) with a given start vertex. Many open questions remain. Is there an $O(m)$-time algorithm for the BP and BST problems? Is there a deterministic $O(m\beta(m, n))$-time algorithm for the BP and BST problems? Can the $O(m + n \log n)$-time algorithm for Single-Source Bottleneck Paths (SS-BP) problem be improved? Can the $O(m + k \log k)$-time algorithm for finding the values of *all* vertices of a DGG be improved?

References

1 D. Andersson, K.A. Hansen, P.B. Miltersen, and T.B. Sørensen. Deterministic graphical games revisited. *Journal of Logic and Computation*, 22(2):165–178, 2010.

2 H. Björklund, S. Sandberg, and S. Vorobyov. Memoryless determinacy of parity and mean payoff games: a simple proof. *Theoretical Computer Science*, 310(1-3):365–378, 2004.

3 H. Björklund and S. Vorobyov. Combinatorial structure and randomized subexponential algorithms for infinite games. *Theoretical Computer Science*, 349(3):347–360, 2005.

4 M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.

5 P.M. Camerini. The min-max spanning tree problem and some extensions. *Information Processing Letters*, 7(1):10–14, 1978. doi:10.1016/0020-0190(78)90030-3.

6 B. Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *Journal of the ACM*, 47(6):1028–1047, 2000.

7 E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

8 R. Duan and S. Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *Proc. of 20th SODA*, pages 384–391, 2009.

9 J. Edmonds and D.R. Fulkerson. Bottleneck extrema. *Journal of Combinatorial Theory*, 8(3):299–306, 1970.

10 J. Edmonds and R.M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

11 A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8:109–113, 1979.

12 E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. of 32nd FOCS*, pages 368–377, 1991.

13 M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

14 M.L. Fredman and D.E. Willard. Surpassing the information-theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, 1993.

15 H.N. Gabow, Z. Galil, T.H. Spencer, and R.E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6:109–122, 1986.

16 H.N. Gabow and R.E. Tarjan. Algorithms for two bottleneck optimization problems. *Journal of Algorithms*, 9(3):411–417, 1988.

17 V.A. Gurvich, A.V. Karzanov, and L.G. Khachiyan. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *USSR Computational Mathematics and Mathematical Physics*, 28:85–91, 1988.

18 Y. Han and M. Thorup. Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In *Proc. of 43rd FOCS*, pages 135–144, 2002.

19 T.C. Hu. The maximum capacity route problem. *Operations Research*, 9(6):898–900, 1961.

20 M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM Journal on Computing*, 38(4):1519–1532, 2008.

21 D.R. Karger, P.N. Klein, and R.E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42:321–328, 1995.

22 J. Matoušek. Computing dominances in E^n. *Information Processing Letters*, 38(5):277–278, 1991. doi:10.1016/0020-0190(91)90071-O.

23 A.P. Punnen. A fast algorithm for a class of bottleneck problems. *Computing*, 56(4):397–401, 1996. doi:10.1007/BF02253463.

24 A. Shapira, R. Yuster, and U. Zwick. All-pairs bottleneck paths in vertex weighted graphs. *Algorithmica*, 59:621–633, 2011.

25 V. Vassilevska, R. Williams, and R. Yuster. All pairs bottleneck paths and max-min matrix products in truly subcubic time. *Theory of Computing*, 5(1):173–189, 2009.

26 A. Washburn. Deterministic graphical games. *Journal of Mathematical Analysis and Applications*, 153(1):84–96, 1990.

27 E. Zermelo. Über eine anwendung der mengenlehre auf die theorie des schachspiels. In *Proceedings of the Fifth International Congress of Mathematicians*, pages 501—504, 1913.

28 U. Zwick and M.S. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1–2):343–359, 1996.

Packing Groups of Items into Multiple Knapsacks

Lin Chen[1] **and Guochuan Zhang**[*2]

1 **Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary**
chenlin198662@gmail.com

2 **Zhejiang University, College of Computer Science, Hangzhou, China**
zgc@zju.edu.cn

Abstract

We consider a natural generalization of the classical multiple knapsack problem in which instead of packing single items we are packing groups of items. In this problem, we have multiple knapsacks and a set of items which are partitioned into groups. Each item has an individual weight, while the profit is associated with groups rather than items. The profit of a group can be attained if and only if every item of this group is packed. Such a general model finds applications in various practical problems, e.g., delivering bundles of goods. The tractability of this problem relies heavily on how large a group could be. Deciding if a group of items of total weight 2 could be packed into two knapsacks of unit capacity is already *NP*-hard and it thus rules out a constant-approximation algorithm for this problem in general. We then focus on the parameterized version where the total weight of items in each group is bounded by a factor δ of the total capacity of all knapsacks. Both approximation and inapproximability results with respect to δ are derived. We also show that, depending on whether the number of knapsacks is a constant or part of the input, the approximation ratio for the problem, as a function on δ, changes substantially, which has a clear difference from the classical multiple knapsack problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics

Keywords and phrases approximation algorithms, lower bound, multiple knapsack, bin packing

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.28

1 Introduction

The classical multiple knapsack problem aims at a most profitable subset of given items which admits a feasible packing on a given set of knapsacks. In this setting, if an item is packed, its profit is counted into the objective value. In this paper, we investigate a scenario in which items appear in groups, and the items in a group share a single profit. In other words, one can get the profit if and only if all items in the group are packed (can be placed into different knapsacks). It is obviously a natural generalization of the classical model where each group consists of exactly one item. More precisely, the problem of packing groups of items into multiple knapsacks (*GMKP*) is defined as follows. There are N disjoint sets (groups) of items $S_i = \{J_j^i | 1 \leq j \leq n_i\}$ where J_j^i is the j-th item of the i-th set. Each item has a weight $w(J_j^i) = w_j^i$. There are m identical knapsacks (bins), each having a capacity of B. There is a profit p_i for each set S_i, which could be achieved only if every item of the set is packed. The goal is to pack items into knapsacks such that the total profit is maximized.

* Research supported in part by NSFC (11271325).

© Lin Chen and Guochuan Zhang;
licensed under Creative Commons License CC-BY

By scaling, we assume that the capacity of each knapsack is 1 and $w_j^i \in [0,1]$. We define the weight of S_i as $w_i = w(S_i) = \sum_{j \in S_i} w_j^i$, and its density ratio as $p_i/w(S_i)$. Throughout the paper, "bins" and "knapsacks" are used interchangeably.

Although most of the times it is reasonable to assume that every single item has an individual profit, as we do in the classical (multiple) knapsack problem, it does happen in many cases that the profit can only be defined for a group of items, instead of each of them. Consider several people going for hiking together. All the necessities for building a tent, like the poles, ropes, sticks and the tent itself, have one uniform value which could be achieved only when each of them is carried. Another example would be the delivery of huge equipments, which could be split into smaller parts and carried by multiple trucks. However, no part of one equipment has an individual value and it only makes sense to carry all the parts. All of these natural applications motivate us to study the *GMKP* problem.

In general, *GMKP* does not admit any constant ratio approximation algorithm as it is easy to see that deciding whether a single group of items (with the profit of 1) could be packed into $m = 2$ bins is exactly the Partition problem and is NP-complete. However, the intractability of the problem follows from the fact that a single group may have a weight as large as the total capacity of all the knapsacks (bins), which is often not the case in practice. For example, all parts of one huge equipment may exceed the capacity of one truck, however, compared with the total capacity of all the trucks owned by the delivery company, it is usually small. Hence, we put additionally the constraint that $w(S_i) \leq \delta m$ for all i and discuss the approximability of the problem with respect to the parameter $\delta \in (0,1]$.

Throughout this paper, we say that an algorithm has an approximation factor c if it always produces a feasible packing with total profit at least c times the optimal value. Clearly $c < 1$. For the sake of conventional convenience, if the factor c is arbitrarily small, we say such an algorithm does not have a constant ratio.

Related Work

We first provide a brief overview on the classical multiple knapsack problem (*MKP*). In *MKP*, every item j has a weight w_j and profit p_j, and every knapsack (bin) i has an individual capacity of B_i. The goal is to pack items into knapsacks such that the total profit is maximized. In 1999, Kellerer [11] provided a PTAS (Polynomial Time Approximation Scheme) for the special case of the multiple knapsack problem where all knapsacks have the uniform capacity, i.e., $B_i = B$. Later on, Chekuri and Khanna [2] gave a PTAS for the general multiple knapsack problem where each B_i can be different. This PTAS was later improved by Jansen [6] [7] to an EPTAS (Efficient Polynomial Time Approximation Scheme) of a running time $2^{O(log^4(1/\epsilon)/\epsilon)} + n^{O(1)}$. On the other hand, Jansen et al. [9] also showed that unless the Exponential Time Hypothesis fails, there is no approximation scheme which has a running time of $2^{o(1/\epsilon)} + n^{O(1)}$ for the multiple knapsack problem even if there are only two knapsacks (of the unit capacity). Thus, allowing the number of knapsacks m to be part of the input as well as allowing each knapsack to have a distinct capacity does not essentially make the problem harder in the sense that the $2^{O(log^4(1/\epsilon)/\epsilon)} + n^{O(1)}$ time EPTAS for the general *MKP* is almost the best possible even for the special case that $m = 2$. However, things are substantially different for *GMKP* where the profit is associated with groups instead of items. We show in this paper that if m is a constant, *GMKP* admits a constant-factor approximation algorithm as long as $\delta < 1$. If m is part of the input, *GMKP* admits a constant-factor approximation algorithm only if $\delta \leq 2/3$. Furthermore, if we allow knapsacks to have distinct capacities, then even if there are only two kinds of knapsacks, say, m_1 knapsacks of capacity c_1 and m_2 knapsacks of capacity c_2, then for any $\delta \in (0,1)$ the

Table 1 Overview of the results.

m	δ	Upper Bound	Lower Bound
constant	$(0,1)$	$1-f(\delta)+\epsilon$	$1-f(\delta)-\epsilon$
input	$(2/3,1)$	ϵ	0
	$(1/3,2/3]$	$1/2+\epsilon$	$1/2-\epsilon$
	$(1/4,1/3]$	$1-3\delta/(2+3\delta)+\epsilon$	$1/2-\epsilon$
	$(0,1/4]$	$1-3\delta/(2+3\delta)+\epsilon$	$1-2\delta-\epsilon$

constraint $w(S_i) \leq \delta(m_1c_1 + m_2c_2)$ is no longer capable of guaranteeing a constant-factor approximation (See the full version of the paper). Hence, unlike *MKP*, the parameter m, as well as the capacities of knapsacks, influences *GMKP* substantially. We hope the study in this line will help reveal the impact of these parameters.

Our problem is closely related to the all-or-nothing generalized assignment problem (*AGAP*) [1]. The *AGAP* problem also asks for a most profitable packing of n groups of items into m identical knapsacks, where the profit of a group is defined to be the total profit of items in the group, and is achieved only if every item of this group is packed. The major difference between *AGAP* and our *GMKP* problem is that *AGAP* further requires that every knapsack could accommodate at most one item from each group. This additional constraint allows *AGAP* to admit an $O(1)$-approximation algorithm, while *GMKP* does not admit any constant approximation algorithm in general.

Our problem is also closely related to the bin packing problem (*BPP*) in which every item has a weight and the goal is to pack all the items into the smallest number of bins. In *GMKP*, if we know which groups are selected by the optimum solution, we get a bin packing problem as we need to pack the items of the selected groups into a fixed number of bins. It is proved in [8] that the problem is $W[1]$-hard parameterized by the number of bins m even with unary encoding. This result directly implies the $W[1]$-hardness of our problem.

Our Contribution

We give a thorough study on the approximability of *GMKP* with respect to the parameter δ. From now on we will use *GMKP*(δ) to specify the parameter. The reader may refer to Table 1 for an overview, where each lower bound means there exists an algorithm achieving a profit at least a certain fraction of the optimum, and each upper bound means that there does not exist a polynomial time algorithm achieving a profit of such a fraction of the optimum under $P \neq NP$. Here $f(\delta) = 1/(1/\delta + 1)$ if $1/\delta$ is an integer and could be divided by m, and $f(\delta) = 1/\lceil 1/\delta \rceil$ otherwise, and $\epsilon > 0$ is an arbitrarily small constant.

The main contribution of this paper is to give a full characterization of the approximability of *GMKP*(δ), and distinguish *GMKP*(δ) with m being a constant from *GMKP*(δ) with m being part of the input based on such a characterization. Our results imply that, if m is a constant, then *GMKP*(δ) could be approximated to a factor of roughly $1-\delta$, hence it admits a constant-ratio approximation algorithm as long as $\delta < 1$. However, if m is part of the input, *GMKP*(δ) does not admit any constant-ratio approximation algorithm when $\delta > 2/3$ (assuming $P \neq NP$), and admits a $(1/2-\epsilon)$-approximation algorithm as long as $\delta \leq 2/3$. Furthermore, when δ is sufficiently small (e.g., $\delta \in (0, 1/4]$), the approximation ratio lies within $[1-2\delta-\epsilon, 1-3\delta/(2+3\delta)+\epsilon]$, which has a clear difference from the ratio of $1-\delta$ for the case that m is a constant.

To achieve our results, we study $OPT(m)$ as a function of m, where $OPT(m)$ is the optimum profit by using m bins. By modifying the classical dynamic programming algorithm

for *MKP* [2], we show that a profit of $(1-\epsilon)OPT(m)$ could be achieved by using $(1+\epsilon)m$ bins even for the $GMKP(\delta)$ problem. Hence, we could derive in polynomial time a feasible solution of profit $(1-\epsilon)OPT((1-\epsilon)m)$. A crucial observation leading to a PTAS for *MKP* is that $OPT(m)$ is somehow "continuous" in the sense that $OPT((1-\epsilon)m) \geq (1-\epsilon)OPT(m)$. However, it is no longer true for $GMKP(\delta)$. Indeed, if m is part of the input, we prove that by assuming $P \neq NP$, for any $\epsilon > 0$ and $c < 3/2$, the inequality $OPT((1-\epsilon)m) \geq (1-c\delta)OPT(m)$ does not hold, which implies a jump on the optimum. We also show that $OPT((1-\epsilon)m) \geq (1-2\delta-O(\epsilon))OPT(m)$, which implies a $(1-2\delta-\epsilon)$-approximation algorithm. To prove such a bound, we will use the configuration LP for bin packing problem introduced in [4] and apply discrepancy analysis to estimate how the deletion of certain items influences the whole packing.

2 Packing into a Constant Number of Bins

We give almost tight approximation algorithms for $GMKP(\delta)$ when m is a constant. We start with the upper bound, as is shown by the following theorem.

▶ **Theorem 1.** *Assuming $P \neq NP$, there is no $(1-f(\delta)+\epsilon)$-approximation algorithm for the group packing problem $GMKP(\delta)$ for any constant m, where $f(\delta) = 1/(1/\delta+1)$ if $1/\delta$ is an integer and could be divided by m, and $f(\delta) = 1/\lceil 1/\delta \rceil$ otherwise.*

The approximability of $GMKP(\delta)$ relies on the function f which has a jump when $1/\delta$ is an integer and could be divided by m. This is due to the hardness result of the following *Repartition* problem.

***Repartition*-(x, m).** Given x sets of integers $S_1, S_2, \cdots, S_x$ where $S_i = \{b_j^i \in Z^+ | 1 \leq j \leq n_i, n_i \in Z^+\}$, $\sum_{j \in S_i} b_j^i = B$ for every i and $m|Bx$, the problem asks whether there exists a repartition of all the integers b_j^i into m disjoint sets such that the integers in each set sum up to exactly Bx/m.

▶ **Lemma 2.** *Repartition-(x, m) is NP-complete for any x and m such that x could not be divided by m, and is polynomially solvable otherwise.*

It is easy to see that Partition is actually a special case of the Repartition problem by taking $x = 1$ and $m = 2$.

We complement Theorem 1 by giving an algorithm with the approximation ratio that almost matches the bound.

▶ **Theorem 3.** *There is a $(1-f(\delta)-\epsilon)$-approximation algorithm for the group packing problem $GMKP(\delta)$ if m is a constant, where $f(\delta) = 1/(1/\delta+1)$ if $1/\delta$ is an integer and could be divided by m, and $f(\delta) = 1/\lceil 1/\delta \rceil$ otherwise.*

To prove Theorem 3, we need the following lemma.

▶ **Lemma 4.** *If there exists a feasible solution Sol of profit ϕ for the group packing problem $GMKP(\delta)$ when m is a constant, and the total weight of all the items in the solution is at most $(1-\epsilon)m$, then there exists a polynomial time algorithm which returns a feasible solution of profit at least ϕ.*

The proof of Lemma 4 is a combination of guessing out big items (with weight larger than ϵ^2), and greedily selecting and packing small items (with weight no more than ϵ^2). The fact that the total weight of items in Sol is no more than $(1-\epsilon)m$ ensures that there is

enough room (the amount of ϵm) to offset the errors caused by the possible wrongly selection and packing of small items.

With Lemma 4, Theorem 3 is proved via selecting out appropriate sets whose total weight is at least ϵm and the total profit is at most $f(\delta)OPT$.

Proof of Theorem 3. According to Lemma 4 (for ease of calculation we substitute ϵ by ϵ^2 in the lemma), if the optimum solution of $GMKP(\delta)$ has a total weight at most $m(1-\epsilon^2)$ then the theorem is already proved. Otherwise it suffices to prove that given the optimum solution, say, Sol, we can always delete some sets such that the total weight of these sets is at least $\epsilon^2 m$, and the total profit is at most $f(\delta)OPT$. In Sol if there exists a single set of weight at least $\epsilon^2 m$ and total profit at most $f(\delta)OPT$ we are done. Otherwise every set of weight at least $\epsilon^2 m$ has a profit strictly larger than $f(\delta)OPT$. We call such sets as critical sets and there are at most $1/f(\delta)-1$ critical sets (recall that $1/f(\delta)$ is an integer).

Suppose $1/\delta$ is not an integer dividable by m, then there are at most $1/f(\delta)-1 = \lceil 1/\delta \rceil - 1$ critical sets, with total weight at most $[1-\delta(\lceil 1/\delta \rceil - 1)]m$. Since δ is a constant, it is always possible to choose sufficiently small ϵ such that $1-\delta(\lceil 1/\delta \rceil - 1) \geq 2\epsilon$. As the total weight of all the sets is larger than $(1-\epsilon^2)m$, we know that in addition to critical sets there are also other sets in Sol, and the total weight of these non-critical sets is at least $(1-\epsilon^2)m - \delta(\lceil 1/\delta \rceil - 1)m \geq \epsilon m$. Notice that the total profit of non-critical sets is at most OPT, hence the average ratio of these sets is upper bounded by $OPT/(\epsilon m)$. Hence, by selecting least profitable (in terms of ratios, i.e., $p_i/w(S_i)$) non-critical sets such that their total weight is in $(\epsilon^2 m, 2\epsilon^2 m]$, we know their total profit is at most $2\epsilon^2 m \cdot OPT/(\epsilon m) \leq 2\epsilon OPT$. Overall, we find sets with total weight at least $\epsilon^2 m$ and total profit at most $2\epsilon OPT \leq f(\delta)OPT$, which proves the theorem.

Suppose $1/\delta = \lambda$ is an integer dividable by m, then $f(\delta) = 1/(1+\lambda)$. Recall that every set of weight at least $\epsilon^2 m$ has a profit strictly larger than $OPT/(1+\lambda)$. Consider the optimum solution. If there exist λ sets of items such that their total profit is at least $\lambda/(1+\lambda) \cdot OPT$, then we can guess out these λ sets. As λ could be divided by m, we put items of λ/m sets into one bin, and their total weight is at most $\delta m \cdot \lambda/m = 1$. Hence we derive a feasible packing with profit at least $\lambda/(1+\lambda) \cdot OPT = (1-f(\delta))OPT$. Otherwise, any λ sets in the optimum solution have a total profit less than $\lambda/(1+\lambda) \cdot OPT$, specifically, the λ sets of the largest weight also have a total profit less than $\lambda/(1+\lambda) \cdot OPT$. Hence, among the λ sets of the largest weight, the one of the smallest profit has a profit at most $OPT/(1+\lambda)$, implying that it is not critical, hence has a weight at most $\epsilon^2 m$. Thus, there are at most $\lambda - 1$ critical sets in the optimum, and their total weight is at most $\delta m \cdot (\lambda - 1) = (1-\delta)m$. Given that the total weight of all the sets is at least $m(1-\epsilon^2)$, we know that the non-critical sets have a total weight at least $(\delta - \epsilon^2)m$, and total profit at most OPT. Hence, by selecting out least profitable (in terms of ratios, i.e., $p_i/w(S_i)$) non-critical sets such that their total weight is in $(\epsilon^2 m, 2\epsilon^2 m]$, we know their total profit is at most $2\epsilon^2 m \cdot OPT/(\delta m - \epsilon^2 m) \leq 2\epsilon OPT$ (by taking ϵ sufficiently small such that $\delta > 2\epsilon$). According to Lemma 4 the theorem is proved. ◀

3 Packing into an Arbitrary Number of Bins

Extending the PTAS [2] for the multiple knapsack problem, we have the following.

▶ **Theorem 5.** *There exists a dynamic programming algorithm for $GMKP(\delta)$ which returns a solution of profit $OPT(m)/(1+\epsilon)$ by using $m(1+\epsilon)$ bins.*

Notice that a feasible solution uses m bins, thus the above theorem actually ensures a feasible solution with profit at least $OPT(m(1-\epsilon))/(1+\epsilon)$. In the classical multiple knapsack problem, the profit is associated with items, hence for each bin in the solution of $OPT(m)$ we can calculate the total profit of items packed into this bin. If we delete ϵm bins with the least total profit of items, we obtain a solution of profit at least $(1-\epsilon)OPT(m)$ with $m(1-\epsilon)$ bins, implying that $OPT(m(1-\epsilon)) \geq (1-\epsilon)OPT(m)$, and a PTAS follows directly from Theorem 5. However, this inequality is no longer true when the profit is associated with groups instead of items. We will discuss in the following the approximability of $GMKP(\delta)$ with respect to the value of δ.

Throughout this section we let $OPT = OPT(m)$ for simplicity. We assume ϵ to be an arbitrary small fractional value such that $1/\epsilon$ is an integer, and m to be sufficiently large such that $m\epsilon$ is always an integer.

3.1 $\delta > 2/3$

▶ **Theorem 6.** *Assuming $P \neq NP$, there is no constant ratio approximation algorithm for the group packing problem $GMKP(\delta)$ when $\delta > 2/3$.*

Consider the Bin Packing Problem which asks whether a set of items of weights a_1, a_2, $\cdots$, a_n could be packed into m bins of capacity 1. We denote by $BPP(\delta)$ if $\sum a_j \leq \delta m$. Theorem 6 follows directly from the following lemma.

▶ **Lemma 7.** *$BPP(\delta)$ is strongly NP-complete for $\delta > 2/3$.*

Proof. We reduce from 3-Partition. In the 3-Partition problem, we are given a set of $3k$ positive integers $\{b_1, b_2, \cdots, b_{3k}\}$ such that $\sum b_j = kB$. The problem asks whether there exists a partition of the integers into k disjoint subsets $U_1, U_2, \cdots, U_k$ such that for every i, $|U_i| = 3$ and $\sum_{b_j \in U_i} b_j = B$.

Let $\epsilon > 0$ be an arbitrarily small positive number with $1/\epsilon$ being an integer. Given a 3-Partition instance, we let $b'_i = b_i + B/\epsilon$ and $B' = (1 + 3/\epsilon)B$. We construct an instance of $BPP(\delta)$ with $\delta = 2/3 + O(\epsilon)$ in the following way.

There are $3k$ key items of weights $a_i = b'_i/B'$ for $1 \leq i \leq 3k$. There are $2k/\epsilon$ dummy items, each of weight $(B + B/\epsilon)/B'$. There are $m = k + k/\epsilon$ bins, each of capacity 1. Hence the total weight of items is $(k\epsilon + 5k + 2k/\epsilon)/(\epsilon + 3) \leq (2/3 + O(\epsilon))(k + k/\epsilon)$, i.e., it is a feasible instance of $BPP(\delta)$ for $\delta = 2/3 + O(\epsilon)$.

Suppose the 3-Partition problem admits a feasible solution. Then the bin packing problem also admits a feasible solution by packing all the key items into k bins, and all the dummy items into k/ϵ bins.

Suppose the bin packing problem admits a feasible solution. It is easy to verify that there are three possibilities with respect to the items packed into a single bin. A bin contains only key items, and there are at most three of them, or it contains only dummy items, and there are at most two of them, or it contains one key item and one dummy item. Let x, y, z denote the number of bins with the above-mentioned three kinds of "configuration", respectively. We have the following constraints,

$$3x + z \geq 3k,\ 2y + z \geq 2k/\epsilon,\ x + y + z = m = k + k/\epsilon.$$

Let $z = k + k/\epsilon - x - y$ and plug it back into the first two inequalities, simple calculations show that $x \geq k$ and $y \geq k/\epsilon$. Given that $x, y, z \geq 0$, it follows directly that $x = k$ and $y = k/\epsilon$. Hence, all the key items are packed into k bins, implying a solution to the 3-Partition problem. ◀

3.2 $1/3 < \delta \leq 2/3$

▶ **Theorem 8.** *Assuming $P \neq NP$, for any $\epsilon > 0$ there is no $(1/2+\epsilon)$-approximation algorithm for $GMKP(\delta)$ when $\delta > 1/3$.*

Proof. Recall that the proof of Lemma 7 shows that it is strongly NP-hard to decide whether a group of items of total weight $(2/3+O(\epsilon))m$ could be packed into m bins. To modify it into a feasible instance of $GMKP(\delta)$ for $1/3 < \delta \leq 2/3$, we divide these items into two groups with roughly the same total weight via a simple greedy algorithm, i.e., we open two groups A and B which are initially empty, and each time we add one item into the group with a smaller total weight of items. By doing so items could be divided such that the difference of the total weight between two groups is at most the weight of the largest item, which is $O(\epsilon m)$. Hence, $w(A), w(B) \leq (1/3+O(\epsilon))m$. Let the profit of either group be 1. If there exists a $(1/2+\epsilon)$-approximation algorithm, then it returns a solution with profit strictly larger than 1 if the two groups of items can both be packed into m bins, and returns a solution with profit at most 1 otherwise. Hence, we can use the approximation algorithm to decide whether all the items could be packed into m bins, which is a contradiction to Lemma 7. ◀

We complement Theorem 8 by providing a $(1/2-\epsilon)$-approximation algorithm for $GMKP(\delta)$ when $\delta \leq 2/3$. To achieve this, we first consider $BPP(\delta)$.

▶ **Lemma 9.** *$BPP(\delta)$ is polynomial-time solvable when $\delta \leq 2/3$.*

The Lemma actually falls as corollary of the following observation for the Longest Processing Time (LPT) algorithm for the *Machine Scheduling* problem. In the machine scheduling problem, given is a set of jobs, each of processing time p_j, and the goal is to assign these jobs onto parallel machines such that the completion time of the job that completes last is minimized. LPT is the algorithm that orders jobs in non-increasing order of their processing times, and always assigns a job to the machine with the least load.

▶ **Lemma 10** ([5])**.** *If every job has a processing time larger than $OPT/3$ where OPT is the optimum makespan, then LPT produces an optimal schedule.*

Proof of Lemma 9. Suppose the optimum uses m bins. We show that FFD (First Fit Decreasing) [10] uses no more than m bins. FFD is the algorithm that assigns items into bins in the following way; it sorts items by weight from the largest to the smallest and sorts bins in an arbitrary way. Then it packs each item into the first bin that still has the enough remaining capacity to accommodate it. Consider all the items larger than $1/3$. FFD packs them into no more than m bins via Lemma 10. For the remaining items, if FFD opens an $(m+1)$-st bin for some item j, then at this time all the m bins are filled up to at least $2/3$, hence the total weight of the items, except item j, is at least $2/3m$, which is a contradiction. ◀

Notice that the proof of the above lemma also shows that items of total weight W can always be packed into $\lceil 3/2 \cdot W \rceil$ bins, if every item has a weight no more than $1/2$. To see why, consider items of weight larger than $1/3$. FFD can always pack two of them into one bin. Thus in the solution returned by FFD, except for one bin, every bin is filled up to at least $2/3$. This observation leads to the following lemma.

▶ **Lemma 11.** *A set of items can always be packed into $|S_{>1/2}| + \lceil 3/2 \cdot W_{\leq 1/2} \rceil$ bins, where $S_{>1/2}$ is the set of items whose weight is strictly larger than $1/2$, and $W_{\leq 1/2}$ is the total weight of the remaining items.*

Now we are ready to prove the following theorem.

▶ **Theorem 12.** *There exists a* $(1/2-\epsilon)$*-approximation algorithm for* $GMKP(\delta)$ *when* $\delta \leq 2/3$.

The proof idea is to show that, all the sets selected by the optimum solution of $GMKP(\delta)$ could be divided into two groups such that either group could be packed into αm bins with some constant $\alpha < 1$. If the above claim is true, then $OPT/2$ could be achieved by using at most αm bins, and we could apply Theorem 5 to derive a feasible solution of profit at least $(1/2-\epsilon)OPT$.

Proof. Consider an optimum solution. A set is called huge if its weight is at least ϵm. There are at most $1/\epsilon$ huge sets in the optimum solution and we can guess them (by enumeration). Suppose we guess the correct sets and let them be S_1 to S_h. We partition them into two groups such that either group has a total weight at most $2/3 \cdot m$. This could be achieved via a simple greedy strategy, i.e., we treat each set S_i as a job of processing time $w(S_i)$ and apply LPT (longest processing time first) to schedule them on two identical machines. The makespan of the solution returned is either $\delta m \leq 2/3 \cdot m$ if there are only one or two jobs, or at most $1/2(\sum_{i=1}^{h} w(S_i) - w(S_j)) + w(S_j) \leq 1/2 \cdot m + 1/2 \cdot 1/3m \leq 2/3 \cdot m$ where S_j is the job that finishes last and hence of weight at most $m/3$. Let A and B denotes the two groups returned by the above procedure. Let C be the group of remaining sets in the optimum solution, then each set of C has a weight at most ϵm.

Note that groups A and B are known via guessing (enumeration), while the group C is unknown. Furthermore, the total weight of items in group A (or B) is at most $2/3 \cdot m$. Thus according to Lemma 9, all the items of A (or B) could be packed into m bins. If the total profit of sets in A (or B) is at least $OPT/2$, the theorem is proved.

Otherwise, we prove the theorem using Theorem 5. Consider items of weight larger than $1/2$. Let z_A, z_B and z_C be the number of such items in groups A, B and C respectively. Let W_A, W_B and W_C be the total weight of remaining items in groups A, B and C. We have the following inequalities.

$$z_A + z_B + z_C \leq m$$
$$1/2 \cdot (z_A + z_B + z_C) + W_A + W_B + W_C \leq m$$

According to the above two inequalities, we have

$$z_A + z_B + z_C + 3/2(W_A + W_B + W_C) \leq 7/4 \cdot m.$$

According to Lemma 11, to pack items of group A or group B we need at most $z_A + \lceil 3/2 \cdot W_A \rceil \leq z_A + 3/2 \cdot W_A + 1$ or $z_B + 3/2 \cdot W_B + 1$ bins, respectively. There are two possibilities.

Case 1. Either $z_A + 3/2 \cdot W_A + 1$ or $z_B + 3/2 \cdot W_B + 1$ is very large, i.e., at least $7/8 \cdot m$. Assume w.l.o.g that $z_A + 3/2 \cdot W_A + 1 \geq 7/8 \cdot m$. Recall that the profit of A is less than $OPT/2$, hence the profit of $B \cup C$ is at least $OPT/2$. Notice that $z_A + 3/2 \cdot W_A + 1 \geq 7/8 \cdot m$ implies that $z_B + z_C + 3/2 \cdot (W_B + W_C) + 1 \leq 7/8 \cdot m + 2 \leq (7/8 + \epsilon)m$, hence the sets in $B \cup C$ can be packed into $(7/8 + \epsilon)m$ bins via Lemma 11, which implies that $OPT(m(1-\epsilon)) \geq OPT((7/8+\epsilon)m) \geq 1/2OPT$. Using Theorem 5 we know that the dynamic programming algorithm will return a feasible solution with profit at least $(1/2-\epsilon)OPT$.

Case 2. $z_A+3/2\cdot W_A+1 \leq 7/8\cdot m$ and $z_B+3/2\cdot W_B+1 \leq 7/8\cdot m$. We claim that C could be partitioned into C_1 and C_2 such that $A' = A\cup C_1$, $B' = B\cup C_2$, $z_{A'}+3/2\cdot W_{A'}+1 \leq (7/8+2\epsilon)m$ and $z_{B'} + 3/2 \cdot W_{B'} + 1 \leq (7/8 + 2\epsilon)m$ (Here $z_{A'}, z_{B'}, W_{A'}$ and $W_{B'}$ are defined analogously as before). If the claim is true, then either A' or B' has a profit at least $OPT/2$, implying that $OPT(m(1-\epsilon)) \geq OPT((7/8+2\epsilon)m) \geq 1/2OPT$, and Theorem 12 is proved. To see why the claim holds, we consider the sets in C and let them be S_1 to S_h. We let $z_C(S_i)$ be the number of items with weight larger than $1/2$ in S_i, and $W_C(S_i)$ be the total weight of remaining items in S_i. As group C consists of sets whose weight is at most ϵm, we have $1/2 \cdot z_C(S_i) + W_C(S_i) \leq \epsilon m$ for $1 \leq i \leq h$. To show the partition of C we again view each set S_i as a job of processing time $z_C(S_i) + 3/2 \cdot W_C(S_i) \leq 2\epsilon m$. We shall schedule these jobs onto two identical machines with the initial load of $z_A + 3/2 \cdot W_A + 1 \leq 7/8 \cdot m$ and $z_B + 3/2 \cdot W_B + 1 \leq 7/8 \cdot m$, respectively. Applying List-Scheduling, we claim that after all the jobs are scheduled, the makespan is at most $(7/8 + 2\epsilon)m$ since otherwise, the job that finishes last must be some job S_i, and thus the load of either machine is strictly larger than $7/8 \cdot m$, which contradicts the fact that $z_A + z_B + z_C + 3/2(W_A + W_B + W_C) = z_A + z_B + 3/2(W_A + W_B) + \sum_i (z_C(S_i) + 3/2W_C(S_i)) \leq 7/4 \cdot m$. Taking the sets scheduled on two machines as A' and B', we obtain the desired partition. ◀

3.3 $\delta \leq 1/3$

With a similar proof as for Theorem 8, we have the following lower bound.

▶ **Theorem 13.** *Assuming $P \neq NP$, there is no $(1 - 3\delta/(2 + 3\delta) + O(\epsilon))$-approximation algorithm for GMKP(δ) for any $\epsilon > 0$ when $\delta \leq 1/3$.*

We complement Theorem 13 with the following theorem.

▶ **Theorem 14.** *Given an arbitrary $\epsilon > 0$, there exists a $(1 - 2\delta - O(\epsilon))$-approximation algorithm for GMKP(δ) when $\delta \leq 1/3$.*

By Theorem 5, it suffices to prove $OPT((1-\epsilon)m) \geq (1-2\delta-O(\epsilon))OPT(m)$, as is shown by the following Lemma 15.

We remark that, although intuitively one might expect to show that $OPT(m(1-\epsilon)) \geq (1-O(\epsilon))OPT(m)$, or at least $OPT((1-\epsilon)m) \geq (1-\delta-O(\epsilon))OPT(m)$ for sufficiently small δ, Theorem 13 already implies that $OPT((1-\epsilon)m) \geq (1-c\delta)OPT(m)$ does not hold in general for $c < 3/2$.

▶ **Lemma 15.** *$OPT((1-\Theta(\epsilon^2))m) \geq (1-2\delta-O(\epsilon))OPT(m)$ for $m \geq 20/\epsilon^3$.*

We remark that the above lemma is actually true for any $\delta \in (0, 1]$: for $\delta > 1/2$ it is trivially true, while for $\delta \in (1/4, 1/2]$ although a $(1-2\delta-O(\epsilon))$-approximation algorithm follows, yet the $(1/2-\epsilon)$-approximation algorithm presented in the previous subsection performs better.

We give a brief introduction to the proof. Consider the solution with the profit of $OPT(m)$. In order to prove the inequality, among the sets of items packed in this solution, we need to select some sets such that their total profit is small (at most $(2\delta + O(\epsilon))OPT(m)$), and the deletion of them saves many bins (at least $\Omega(\epsilon^2 m)$ bins). Obviously these sets could not be the sets that consist of items that are very small. To see why, imagine that in $OPT(m)$ each bin is filled up by a huge item of size larger than $1/2$ and a bunch of small items, then even if we delete all the small items the number of bins required for the remaining huge items is still m. Hence, we should better delete sets that contain many big items. To show that such a deletion, combined with the repacking of remaining items could eventually save a significant

number of bins, we will iteratively modify the instance and then apply the discrepancy theory to the Gilmore Gomory LP relaxation [4] for the modified instance. The idea of applying discrepancy theory to Bin Packing is also used in [3] to derive the relationship between Bin Packing and the three-permutation-problem.

Proof. We assume that sets packed in $OPT(m)$ are S_1 to S_h. We further assume that $\sum_i w(S_i) > (1/2-\epsilon)m$ since otherwise $OPT((1-\epsilon)m) = OPT(m)$. To see why, suppose $\sum_i w(S_i) \le (1/2-\epsilon)m$. We let $S_{1/2}$ be the set of items in S_1 to S_h whose weight is larger than $1/2$, and $W_{\le 1/2}$ be the total weight of remaining items. Then $1/2|S_{>1/2}| + W_{\le 1/2} \le \sum_i w(S_i) \le (1/2-\epsilon)m$, whereas $|S_{>1/2}| + \lceil 3/2 W_{\le 1/2} \le \sum_i w(S_i) \rceil \le 2(1/2-\epsilon)m + 1 \le (1-\epsilon)m$. According to Lemma 11 all the sets could be packed into $(1-\epsilon)m$ bins, hence $OPT((1-\epsilon)m) = OPT(m)$.

From now on we will abuse the notation w_i a bit to also denote item i, and we may also abuse the notation $OPT(m)$ to denote the solution that achieves the profit. Let w_1 to w_n be all the items of S_1 to S_h such that $w_1 \ge w_2 \ge \cdots \ge w_n$. Let γ be the least index such that $w_1 + w_2 + \cdots + w_\gamma > (1/2-\epsilon)m$. Obviously w_1 to w_γ should belong to at least $\lceil (1/2-\epsilon)/\delta \rceil$ different sets among S_1 to S_h. For simplicity let these sets be S_1 to S_ℓ with $\ell \ge \lceil (1/2-\epsilon)/\delta \rceil$.

Let $S^\gamma = \{w_1, w_2, \cdots, w_\gamma\}$, $S_i^\gamma = S_i \cap S^\gamma$, $w(S_i^\gamma) = \sum_{j \in S_i^\gamma} w_j$, $p(S_i^\gamma) = p_i$, $\rho(S_i^\gamma) = p_i / w(S_i^\gamma)$. We assume w.l.o.g that $\rho(S_1^\gamma) \ge \rho(S_2^\gamma) \ge \cdots \ge \rho(S_\ell^\gamma)$.

Consider the following knapsack problem. We take each S_i^γ as a single item. Then these ℓ items can be packed into a knapsack of capacity $\sum_{i=1}^{\ell} w(S_i^\gamma) \ge (1/2-\epsilon)m$ with the total profit of $\sum_{i=1}^{\ell} p_i \le OPT(m)$. Recall that $w(S_i^\gamma) \le \delta m$. We let $\ell' \le \ell$ be the least index such that $w(S_{\ell'}^\gamma) + w(S_{\ell'+1}^\gamma) + \cdots + w(S_\ell^\gamma) \in (8\epsilon m, (8\epsilon+\delta)m]$. Furthermore, since $S_{\ell'}^\gamma$ to S_ℓ^γ are the least profitable items (in terms of ratios), we know that

$$\frac{\sum_{i=\ell'}^{\ell} p_i}{\sum_{i=\ell'}^{\ell} w(S_i^\gamma)} \le \frac{\sum_{i=1}^{\ell} p_i}{\sum_{i=1}^{\ell} w(S_i^\gamma)} \le \frac{\sum_{i=1}^{\ell} p_i}{(1/2-\epsilon)m},$$

$$\sum_{i=\ell'}^{\ell} p_i \le (8\epsilon+\delta)m \cdot \frac{\sum_{i=1}^{\ell} p_i}{(1/2-\epsilon)m} \le (2\delta + O(\epsilon)) \sum_{i=1}^{\ell} p_i \le (2\delta + O(\epsilon)) OPT(m).$$

Suppose we delete sets $S_{\ell'}$ to S_ℓ from the optimum solution $OPT(m)$. The total profit of the remaining sets is at least $(1 - 2\delta - O(\epsilon))OPT(m)$, and in the following we show that to pack all the items of the remaining sets, $(1-\Theta(\epsilon^2))m$ bins suffice, which proves the lemma.

Notice that directly deleting items of sets $S_{\ell'}$ to S_ℓ from the solution of $OPT(m)$ leaves some empty space in the m bins, and we aim to somehow merge these spaces to create $\Theta(\epsilon^2 m)$ empty bins. Instead of iteratively moving items, we will use a "global approach" by applying the discrepancy theory to the configuration LP for the bin packing problem.

Consider the instance of packing items $w_1, w_2, \cdots, w_n$. For any set of items X, we denote by $\sigma(X)$ the minimum number of bins needed to pack them. Let $S = \{w_1, w_2, \cdots, w_n\}$, $S' = \cup_{i=\ell'}^{\ell} S_i^\gamma \subseteq S^\gamma$. It is easy to see that $\sigma(S) \le m$, $w(S') \in (8\epsilon m, (8\epsilon+\delta)m]$. To prove the lemma, it suffices to prove Claim 1.

▶ **Claim 1.** $\sigma(S \setminus S') \le (1 - \Theta(\epsilon^2))m$.

Consider w_γ. We claim that, if $w_\gamma \le 2\epsilon$, then $\sigma(S \setminus S') \le (1-\epsilon)m \le (1 - O(\epsilon^2))m$. To see why, recall the definition of γ, we have $w_1 + w_2 + \cdots + w_{\gamma-1} \le (1/2-\epsilon)m$, implying that these items could be packed into $(1-\epsilon)m$ bins. We now delete items of $S' \subseteq \{w_1, w_2, \cdots, w_\gamma\}$ from this solution, and then pack items w_γ to w_n via First-Fit. We claim that, we do not

need to open new bins. Suppose the claim is not true, then among these $(1-\epsilon)m$ bins at least $(1-\epsilon)m-1$ bins are filled up to at least $1-2\epsilon$. Hence, $((1-\epsilon)m-1)(1-2\epsilon) \leq w(S \setminus S') \leq (1-8\epsilon)m$, which is a contradiction.

From now on we assume $w_\gamma > 2\epsilon$. In this case we do not prove Claim 1 directly. In the following, we will iteratively give Claim 2 to Claim 5 and show that, for $1 \leq i \leq 4$, Claim $i+1$ implies Claim i. We then prove Claim 5 at the end, which suffices to show the truth of Claim 1, and consequently the lemma.

Consider small items whose weight is at most ϵ. We modify small items in the following way. We iteratively agglomerate small items into a big item of weight $[\epsilon, 2\epsilon)$. At last there may still be some small items left with total weight less than ϵ, and we simply agglomerate them into a single item. Let $S^\sharp$ be the sets of modified items, then it is easy to see that except at most one item, each item in $S^\sharp$ has a weight at least ϵ, and $w(S) = w(S^\sharp)$. Furthermore, if we order items of $S^\sharp$ in non-increasing order of their weight, the first γ items would still be w_1 to w_γ. Hence, $S' \subseteq S^\gamma \subseteq S^\sharp$. As the modification procedure only agglomerate items, to prove Claim 1, it suffices to prove the following Claim 2.

▶ **Claim 2.** $\sigma(S^\sharp \setminus S') \leq (1-\Theta(\epsilon^2))m$.

Notice that $w(S') \in (8\epsilon m, (8\epsilon+\delta)m]$ and the weight of each item is at most $1 \leq \epsilon m$. We can easily split S' into S'_1 and S'_2 such that $w(S'_1) \in [4\epsilon m, 5\epsilon m)$ and $w(S'_2) \geq 3\epsilon m$. As items are agglomerated, it is no longer true that $\sigma(S^\sharp) \leq m$. However, we claim that, $\sigma(S^\sharp \setminus S'_1) \leq m$. To see why, consider the solution of $\sigma(S) \leq m$. We take out all the small items together with items of S'_1. Now we add back the agglomerated items via First-Fit. We claim that, we do not need to open new bins since otherwise, at least m bins are filled up to at least $1-2\epsilon$, implying that $w(S^\sharp \setminus S'_1) \geq (1-2\epsilon)m$, which is a contradiction as $w(S^\sharp \setminus S'_1) = w(S) - w(S'_1) \leq (1-4\epsilon)m$.

Let $S^\sharp = S^\gamma \cup S_\alpha$. Claim 2 is equivalent to $\sigma(S^\sharp \setminus S') = \sigma((S^\gamma \setminus (S'_1 \cup S'_2)) \cup S_\alpha) \leq (1-\Theta(\epsilon^2))m$. By re-indexing items we assume that $S^\gamma \setminus S'_1 = \{w_1, w_2, \cdots, w_{\gamma'}\}$ for some $\gamma' < \gamma$, and $S_\alpha = \{w_{\gamma'+1}, w_{\gamma'+2}, \cdots, w_{n'}\}$ where $w_{\gamma'+1} \geq w_{\gamma'+2} \geq \cdots \geq w_{n'}$. As $w(S'_2) \geq 3\epsilon m$, S'_2 consists at least $3\epsilon m$ items of $S^\gamma \setminus S'_1$. Instead of deleting items of S'_2, we consider the instance of deleting $3\epsilon m$ largest items from S_α, i.e., deleting $\hat{S}_\alpha = \{w_{\gamma'+1}, \cdots, w_{\gamma'+3\epsilon m}\}$ (if $n' \leq \gamma' + 3\epsilon m$ then $\hat{S}_\alpha = S_\alpha$). Compare $(S^\gamma \setminus (S'_1 \cup S'_2)) \cup S_\alpha$ with $(S^\gamma \setminus S'_1) \cup (S_\alpha \setminus \hat{S}_\alpha)$. Since there are at least $3\epsilon m$ items in S'_2, each being larger than (or equal to) any item in $\hat{S}_\alpha$, we know there exists an injection such that each item in $(S^\gamma \setminus (S'_1 \cup S'_2)) \cup S_\alpha$ could be mapped to a larger or equal item in $(S^\gamma \setminus S'_1) \cup (S_\alpha \setminus \hat{S}_\alpha)$. Hence, to prove Claim 2, it suffices to prove the following Claim 3.

▶ **Claim 3.** $\sigma((S^\gamma \setminus S'_1) \cup (S_\alpha \setminus \hat{S}_\alpha)) \leq (1-\Theta(\epsilon^2))m$.

Recall that $\sigma((S^\gamma \setminus S'_1) \cup S_\alpha) \leq m$. Consider a feasible solution of packing items of $(S^\gamma \setminus S'_1) \cup S_\alpha$ into m bins (empty bins are allowed). We say a bin is critical if items from $S^\gamma \setminus S'_1$ occupy the space of at most $1/2$, and non-critical otherwise. Hence, there are at most $(1-\epsilon)m$ non-critical bins since otherwise the total weight of items from $S^\gamma \setminus S'_1$ is larger than $(1-\epsilon)m/2 \geq (1/2-\epsilon)m \geq w(S^\gamma)$, which is a contradiction.

Let $\beta \geq \epsilon m$ be the number of critical bins. Let S^c be the set of items packed in critical bins, $S^c_\alpha = S^c \cap S_\alpha$ and $\tau = |S^c_\alpha|$. For simplicity let $w'_1 \geq w'_2 \geq \cdots \geq w'_\tau$ be all the items of S^c_α. Let $\hat{S}^c_\alpha = \{w'_1, \cdots, w'_{3\epsilon m}\}$ be the largest $3\epsilon m$ items in $S^c_\alpha \subseteq S_\alpha$. Compare $\hat{S}_\alpha$ and $\hat{S}^c_\alpha$, i.e., the largest $3\epsilon m$ items in S_α and the largest $3\epsilon m$ items in $S^c_\alpha \subseteq S_\alpha$. Obviously there is an injection which maps each item in $\hat{S}^c_\alpha$ to a larger or equal item in $\hat{S}_\alpha$. Hence to prove Claim 3 it suffices to prove the following Claim 4.

▶ **Claim 4.** $\sigma((S^\gamma \setminus S'_1) \cup (S_\alpha \setminus \hat{S}^c_\alpha)) \leq (1 - \Theta(\epsilon^2))m$.

A critical configuration is a configuration for items of S^c_α, represented by a column vector $\nu = (b_1, b_2, \cdots, b_\tau)^T$, where $b_i \in \{0, 1\}$ denoting whether $w'_i \in S^c_\alpha$ is packed. Obviously the packing of each critical bin could be represented by some items of $S^\gamma \setminus S'_1$ together with a critical configuration, and there are β critical configurations corresponding to the β critical bins. Let $B = (\nu_1, \nu_2, \cdots, \nu_\beta) \in \{0, 1\}^{\tau \times \beta}$ be the matrix of these β configurations. Then obviously $Be_\beta = e_\tau$ where e_k is a column vector with k components, each being 1.

We now use the idea of [3] to re-write the equation $Be_\beta = e_\tau$. Let matrix A be defined as $A_i = \sum_{j=1}^{i} B_j$ where A_i (B_j, resp.) denotes the i-th (j-th, resp.) row of the matrix A (B, resp.), i.e., A_{ij} denotes the total number of items w'_1 to w'_i in the j-th configuration ν_j. Then from $Be_\beta = e_\tau$ we derive $Ae_\beta = (1, 2, \cdots, \tau)^T$. Furthermore, since each items, except the smallest one, is of weight at least ϵ, each configuration consists at most $1/\epsilon$ items. As each column of A is monotone, A is a monotone matrix with each entry $A_{ij} \in \{0, 1, 2, \cdots, 1/\epsilon\}$. Hence, A is a $1/\epsilon$-monotone matrix. Let A' be the matrix of attaching $A_{\tau+1} = (1/\epsilon, 1/\epsilon, \cdots, 1/\epsilon)^T$ as the new last row of A, then A' is also monotone with

$$A'e_\beta = (1, 2, \cdots, \tau, \beta/\epsilon)^T.$$

▶ **Claim 5.** There exists a 0-1 vector $x = (x_1, x_2, \cdots, x_\beta)^T$ such that $A'x = (\psi, (\beta - \Delta)/\epsilon)^T$, where $\psi = Ax$ is a vector with i-th component $\psi_i \geq \max\{0, i - 3\epsilon m\}$ for $1 \leq i \leq \tau$, and $\Delta = \Omega(\epsilon^2 m)$.

We prove Claim 5 implies Claim 4 by showing that items of $(S^\gamma \setminus S'_1) \cup (S_\alpha \setminus \hat{S}^c_\alpha)$ could be packed into $\beta - \Delta/2$ bins. Indeed, $Ax = \psi$ means by using $x_i \in \{0, 1\}$ copies of the configuration ν_i (i.e., the i-th column of B), we can pack a subset $S^* \subseteq S^c_\alpha = \{w'_1, \cdots, w'_\tau\}$ of items such that $|S^* \cap \{w'_1, \cdots, w'_i\}| = \psi_i$. As $\psi_i \geq \max\{0, i - 3\epsilon m\}$, using these configurations we are able to pack items of $S^c_\alpha \setminus \hat{S}^c_\alpha$. Furthermore, $\sum x_i = \beta - \Delta$ means that in total we save Δ critical configurations, by removing which we get Δ bins which are at most half full since in a critical bin, items that are not in the critical configuration have a total weight at most $1/2$. Hence, we can merge items of two such bins into one bin, i.e., we can save $\Delta/2$ bins and Claim 4 follows if $\Delta = \Omega(\epsilon^2 m)$.

We have shown so far that Claim $i + 1$ implies Claim i for $1 \leq i \leq 4$, hence Claim 1, and consequently the lemma, will follow from the truth of Claim 5. We now prove Claim 5. We show there exists such an integer solution x. Consider the fractional solution $y = \theta e_\beta$ where $\theta = 1 - \epsilon m/\tau$. Obviously $A'y = (\theta, 2\theta, \cdots, \tau\theta, \beta\theta/\epsilon)^T$. According to the discrepancy theory [12] there exists an integer solution $x \in \{0, 1\}^\beta$ such that

$$||A'x - A'y||_\infty \leq lindisc(A').$$

It is shown in [3] that for k-monotone $m \times n$ matrices the linear discrepancy is bounded by $5k \log_2(2 \min\{m, n\})$, hence we have

$$lindisc(A') \leq 5/\epsilon \cdot \log_2(2 \min\{\tau + 1, \beta\}) \leq 5/\epsilon \cdot \log_2(2m/\epsilon),$$

i.e., $||A'x - A'y||_\infty \leq 5/\epsilon \cdot \log_2(2m/\epsilon) = d$. Let $A'x = (\psi, \omega)$, then $\psi_i \geq \max\{0, i\theta - d\}$. For $i \geq 3\epsilon m$, $m \geq 20/\epsilon^3$, we have

$$i\theta - d \geq i(1 - \epsilon m/\tau) - 5/\epsilon \cdot \log_2(2m/\epsilon) \geq i - \epsilon m - 5/\epsilon \cdot \log_2(2m/\epsilon) \geq i - 3\epsilon m.$$

For ω, we have $\omega \leq \beta\theta/\epsilon + d$. Since each item, except for the smallest one, has a weight at least ϵ, we have $\tau \leq \beta/\epsilon + 1$, and thus $\theta \leq 1 - \epsilon m/(\beta/\epsilon + 1) \leq 1 - \epsilon^2 m/(2\beta)$. Recall that

$\beta \geq \epsilon m$, for $m \geq 20/\epsilon^3$ we have

$$\omega \leq \beta\theta/\epsilon + d \leq \beta/\epsilon - \epsilon m/2 + 5/\epsilon \cdot \log_2(2m/\epsilon) \leq \beta/\epsilon - \epsilon m/4.$$

Thus, $\Delta \geq \epsilon^2 m/4 = \Omega(\epsilon^2 m)$ as we desired. ◀

References

1. R. Adany, M. Feldman, E. Haramaty, R. Khandekar, B. Schieber, R. Schwartz, H. Shachnai, and T. Tamir. All-or-nothing generalized assignment with application to scheduling advertising campaigns. In *Proc. of IPCO 2013*, pages 13–24, 2013.
2. C. Chekuri and S. Khanna. A polynomial time approximaion scheme for the multiple knapsack problem. *SIAM J. Comput.*, 35(3):713–728, 2006.
3. F. Eisenbrand, D. Pálvölgyi, and T. Rothvoss. Bin packing via discrepancy of permutations. *ACM Trans. Algorithms*, 9(3):39–49, 2013.
4. P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:39–49, 1961.
5. R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17(2):416–429, 1969.
6. K. Jansen. Parameterized approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, 39(4):1392–1412, 2009.
7. K. Jansen. A fast approximation scheme for the multiple knapsack problem. In *Proc. of SOFSEM'12*, pages 313–324, 2012.
8. K. Jansen, S. Kratsch, D. Marx, and l. Schlotter. Bin packing with fixed number of bins revisited. *J. Comput. Syst. Sci.*, 79(1):39–49, 2013.
9. K. Jansen, F. Land, and K. Land. Bounding the running time of algorithms for scheduling and packing problems. In *Proc. of WADS'13*, pages 313–324, 2013.
10. D. S. Johnson. *Near-optimal bin-packing algorithms.* Doctoral Thesis. MIT Press, 1973.
11. H. Kellerer. A polynomial time approximation scheme for the multiple knapsack problem. In *Proc. of APPROX'99*, pages 51–62, 1999.
12. J. Matousek. *Geometric discrepancy.* Springer-Verlag, 1999.

Cost Functions Definable by Min/Max Automata*

Thomas Colcombet[1], Denis Kuperberg[2], Amaldev Manuel[3], and Szymon Toruńczyk[†4]

1 **CNRS & LIAFA, Université Paris Diderot, Paris 7, France**
2 **IRIT/ONERA, Toulouse, France**
3 **MIMUW, University of Warsaw, Poland**
4 **MIMUW, University of Warsaw, Poland**

Abstract

Regular cost functions form a quantitative extension of regular languages that share the array of characterisations the latter possess. In this theory, functions are treated only up to preservation of boundedness on all subsets of the domain. In this work, we subject the well known *distance automata* (also called *min-automata*), and their dual *max-automata* to this framework, and obtain a number of effective characterisations in terms of logic, expressions and algebra.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases distance automata, B-automata, regular cost functions, stabilisation monoids, decidability, min-automata, max-automata

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.29

1 Introduction

Regular languages enjoy multiple equivalent characterisations, in terms of regular expressions, automata, monoids, monadic second order (MSO) logic, etc. One of them is purely algebraic: a language $L \subseteq A^*$ is regular if and only if its two-sided Myhill-Nerode congruence on A^* has finite index. These characterisations have been refined further for many subclasses of regular languages. The archetypical example is the Schützenberger-McNaughton-Papert theorem. which equates the class of star-free languages (i.e. expressible by star-free regular expressions with complementation), the class of first-order definable languages, the class of languages accepted by *counter-free* automata, and the class of languages definable by aperiodic monoids (i.e. those that satisfy the equation $x^{n+1} = x^n$ for sufficiently large n). This gives an algebraic and effective characterisation of the class of star-free languages.

The theory of regular languages has been extended in many directions – to infinite words, trees, infinite trees, graphs, linear orders, traces, pictures, data words, nested words, timed words etc. With some effort, some of the above characterisations can be transferred, by finding the right notion of a "regular" language, and by finding algebraic and logical characterisations of certain subclasses of languages among the class of all the "regular" languages.

Whereas languages are qualitative objects, in this paper, we study characterisations of classes of quantitative objects. One of the classes that we study are cost functions defined by *distance automata*. A distance automaton $\mathcal{A}$ is like a nondeterministic finite automaton, where each transition additionally carries a *weight*, i.e., a natural number. The weight of

* The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 259454.
† Author supported by the National Science Center (decision DEC-2012/07/D/ST6/02443).

© Thomas Colcombet, Denis Kuperberg, Amaldev Manuel, and Szymon Toruńczyk;
licensed under Creative Commons License CC-BY

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 29; pp. 29:1–29:13

Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a run is the sum of the weights of the transitions in the run. The distance automaton $\mathcal{A}$ then associates to each input word w a value $[\![\mathcal{A}]\!](w) \in \mathbb{N} \cup \{\infty\}$, defined as the minimal weight of an accepting run over w, and ∞ if there is no accepting run. The central decision problem is the *limitedness problem* – does the function $[\![A]\!]$ have a finite range?

Distance automata were introduced by Hashiguchi in his solution of the star height problem, which he reduced to the limitedness problem for distance automata via a strenuous reduction. As distance automata try to minimize the value of a run, we call them *min-automata* in this paper. *Max-automata* are the dual model, for which the value of the word is the maximal weight of an accepted run. Min-automata and max-automata have appeared in various contexts (see related work below for more on this) under various names.

Distance automata were extended in subtly distinct ways to nested distance-desert automata [12], R-automata [1], B-automata [4, 6] by allowing several counters instead of just one, and allowing these counters to be reset. The limitedness problem is decidable for all these classes. In terms of cost functions all these extended models are equivalent.

Colcombet [6, 8] discovered that functions defined by B-automata enjoy a very rich theory of *regular cost functions*, extending the theory of regular languages. A *cost function* is an equivalence class of functions, where two functions $f, g : A^* \to \mathbb{N} \cup \{\infty\}$ are equivalent if they are bounded over precisely the same subsets of A^*. A *regular cost function* is the equivalence class of a function computed by a B-automaton. Colcombet gave equivalent characterisations of regular cost functions in terms of a quantitative extension of MSO, an extension of monoids called *stabilisation monoids*, Later, analogous characterisations were described, in terms of a quantitative extension of regular expressions, in terms of logics and regular expressions manipulating profinite words (a completion of the set of finite words in a certain metric) [17], and in terms of a finite index property [17, 14]. In [15] a characterisation in terms of a quantitative extension of FO on the Σ-tree is given.

Contributions

In this paper, we propose a Schützenberger-McNaughton-Papert style characterisation of the subclass of the class of regular cost functions, defined by distance automata – in terms of logic, regular expressions and algebra, i.e., by conditions satisfied by the syntactic stabilisation monoid. The last characterisation provides a machine-independent, purely algebraic description of the cost functions defined by distance automata – or min-automata. We also provide similar characterisations for the dual class of max-automata. Although their definition is simply obtained by replacing min by max, the statements and their proofs are quite different for both classes. Our characterisations are effective, i.e., given a B-automaton, it is decidable whether the cost function it defines is recognisable by a min- or max-automaton. The detailed proofs can be found in the long versions, on the webpage of authors.

Related Work

Characterising special classes of regular cost functions in various formalisms has been done in [11, 14]. In [11], the class of temporal cost functions was defined and studied. These cost functions are only allowed to measure *consecutive* events, for instance the function counting the number of occurrences of a letter in an input word is not temporal. Equivalent characterisations of this class were given in terms of cost automata, regular languages, stabilisation monoids. Additionally, in [7], an equivalent fragment of cost MSO was given. In [14], the class of aperiodic cost functions was considered, as a generalisation of star-free languages. It was shown that this class of function can be equivalently characterised by

definability via cost linear temporal logic, cost first order logic, or group-trivial stabilisation monoids, generalising the Schützenberger-McNaughton-Papert theorem to cost functions.

In the papers [5] and [3], min- and max-automata were defined in a different way, as *deterministic* automata with many counters, over which any sequence of instructions could be performed in a single transition, where each instruction is either of the form $c := c + 1$ or $c := \max(d, e)$ (in the case of max-automata) or $c := \min(d, e)$ (in the case of min-automata), where c, d, e are counters. As these models were studied in relationship with logics over infinite words, rather than evaluating a finite word to a number, they were used as acceptors of infinite words. However, their finitary counterparts are equivalent to the models studied in this paper, up to cost function equivalence (see Proposition 2.4). Note that nondeterminism is exchanged for multiple counters with aggregation (min or max).

In the paper [2], Cost Register Automata are studied, and are parametrised by a set of operations. Those too are deterministic automata with many registers (i.e. counters). For the set of operations denoted $(\min, +c)$, one obtains a model equivalent to the min-automata of [5], and for the set of operations denoted $(\max, +c)$, one obtains a model equivalent to the max-automata [3].

Min-automata can be equivalently described as nondeterministic weighted automata over the semiring $(\mathbb{N} \cup \{\infty\}, \min, +)$, where min plays the role of addition and $+$ of multiplication. Similarly, max automata can be equivalently described as nondeterministic weighted automata over the semiring $(\mathbb{N} \cup \{\infty, \bot\}, \max, +)$, where max plays the role of addition and $+$ of multiplication (and $\bot$ is neutral with respect to max and absorbing with respect to $+$). Using this formalism, decidability results about precision of approximation of functions computed by min-automata are shown in [9]. Similar results on max-automata are presented in [10], together with an application to evaluation of time complexity of programs.

2 Preliminaries

In this section, we recall various models of cost automata, and the theory of regular cost functions. For more details, the reader should confer [6, 8, 11]. We write $\mathbb{N}_\infty$ for the set $\mathbb{N} \cup \{\infty\}$. We follow the convention that $\inf \emptyset = \min \emptyset = \infty$ and $\sup \emptyset = \max \emptyset = 0$.

2.1 Automata

We recall the notions of B- and S-automata, and relate them to min- and max-automata.

B-automata and S-automata. *B- and S-automata* are nondeterministic automata over finite words, which are moreover equipped with a finite set of *counters*. Each counter admits three basic operations: incrementation by one, denoted `i`, reset to zero, denoted `r`, and the idle operation, denoted ε. Initially, all counters are set to 0, and during the run each transition of the automaton performs one operation on each counter separately (formally, a transition is a tuple (p, a, o, q), where p is its source state, q is its target state, a is the label and $o = (o_c)_c$ is a vector of operations, one per each counter c). Additionally, we allow counters to be reset after the run terminates in an accepting state, depending on the state. If in a run ρ some transition resets a counter currently storing a value n, then we say that n is a *reset value* for the considered run ρ.

We now define the value of a word under a B-automaton; the defintion for S-automata is dual and will be given later. Let $\mathcal{B}$ be a B-automaton and w be an input word. For a run ρ

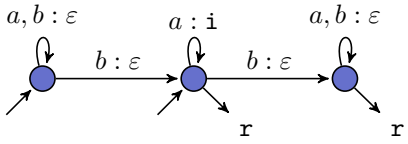


Figure 1 A B-automaton, which is also a min-automaton. Edges with no source state mark initial states, and edges without a target state mark accepting states and may reset the counter.

over the word w, define the *cost* of ρ as the maximum of the set of its reset values:

$$\text{cost}(\rho) = \max\{n \in \mathbb{N} : n \text{ is a reset value for } \rho\}.$$

Recall that the maximum of the empty set is equal to 0. Finally, define $[\![\mathcal{B}]\!](w)$ as the minimal value of an accepting (initial-to-accepting state) run over w:

$$[\![\mathcal{B}]\!](w) \;=\; \min\{\text{cost}(\rho) \mid \rho \text{ is an accepting run of } \mathcal{B} \text{ over } w\}.$$

Note that this value can be infinite, if there is no accepting run of $\mathcal{B}$ over w. In particular, if the set of counters is empty, then $[\![\mathcal{B}]\!](w) = 0$ if $\mathcal{B}$ has an accepting run over w, otherwise $[\![\mathcal{B}]\!](w) = \infty$. In this way, B-automata generalize finite automata.

If $\mathcal{A}$ is an S-automaton, the definitions are obtained by swapping min with max. In particular, the cost of the run is the minimum of the set of its reset values (recall that $\min \emptyset = \infty$) and $[\![\mathcal{A}]\!](w)$ is the maximal value of an accepting run over w. If $\mathcal{A}$ has no counters, then $[\![\mathcal{A}]\!](w) = \infty$ if $\mathcal{A}$ has an accepting run over w, otherwise $[\![\mathcal{A}]\!](w) = 0$.

For a B- or S-automaton $\mathcal{A}$, we call $[\![\mathcal{A}]\!]$ the function *computed* by $\mathcal{A}$.

▶ **Example 2.1.** We construct a B-automaton that computes the smallest length of a block of consecutive a's in an input word consisting of a's and b's. In other words, for an input word w of the form $a^{n_1}ba^{n_2}\dots ba^{n_k}$, the computed value is $f_{\min}(w) = \min_{j=1}^{k} n_j$. The automaton has one counter, and is depicted in Figure 1.

▶ **Example 2.2.** Another example of a function computed by a B-automaton is the following. For a given word w over the alphabet $\{a, b, c\}$ of the form $w_1cw_2\dots cw_k$, where the words $w_1, \dots, w_k$ are over the alphabet $\{a, b\}$, define $f(w) = \max_{j=1}^{k} f_{\min}(w_j)$. The function f is computed by a B-automaton obtained from the automaton in Figure 1 by adding a c-labeled, resetting transition from every accepting state to every initial state.

▶ **Example 2.3.** The automaton in Figure 1 can be interpreted as an S-automaton which, for an input word w of the form $a^{n_1}ba^{n_2}\dots ba^{n_k}$, computes the value $f_{\max}(w) = \max_{j=1}^{k} n_j$.

Min-automata and max-automata. A *min-automaton* is a one-counter B-automaton $\mathcal{B}$, with only two operations allowed: `i` (increment) and ε (do nothing). In particular, resets are not allowed during the run. However, every counter is reset at the end of the run. Therefore the last counter value is a reset value. In other words, a min-automaton is a nondeterministic finite automaton in which every edge carries one of the two operations `i` or ε which manipulate the only counter. The cost of a run is the last value attained by the counter, and $[\![\mathcal{B}]\!](w)$ is the minimum of the costs of all accepting runs. This corresponds exactly to the definition of a *distance automaton* given in the introduction, with `i` corresponding to 1 and ε corresponding to 0 in the distance automaton. The automaton from Example 2.1 is a min-automaton.

Dually, a *max-automaton* $\mathcal{A}$ is a one-counter S-automaton, with only the two operations $\mathtt{i}$ and ε allowed, and where the automaton may, depending of the last state assumed, reset or not the counter at the end of the run. Therefore, the cost of a run is again the last value attained by the counter if it is reset, and $+\infty$ if it is not reset. The value of a word $[\![\mathcal{A}]\!](w)$ is the maximum of the costs of all accepting runs. Example 2.3 gives an example of a max-automaton.

As mentioned in the related work in the introduction, min/max-automata are related to other notions from the literature. We establish this connection in the proposition below, and later on in this paper, we will only talk about min- and max-automata.

A *weighted automaton* over a semiring $\mathcal{S}$ specifies a $n \times n$ matrix $h(a)$ over $\mathcal{S}$ for each letter a in the input alphabet, and two vectors I, F of length n over $\mathcal{S}$. The *value* associated to a word $a_1 \dots a_l$ over the input alphabet is the product of the matrices $I^T \cdot h(a_1) \cdots h(a_l) \cdot F$, which is a 1×1 matrix, identified with an element of $\mathcal{S}$. In the proposition below, a value $\bot$ returned by a weighted automaton is interpreted as 0.

▶ **Proposition 2.4** ([2, 5]). *Min-automata are equivalent to distance automata, to nondeterministic weighted automata over the semiring* $(\mathbb{N}_\infty, \min, +)$*, and to deterministic automata with many registers storing elements of* $\mathbb{N}_\infty$*, allowing the binary* min *operation and unary incrementation operation.*

Dually, max-automata are equivalent to nondeterministic weighted automata over the semiring $(\mathbb{N}_\infty \cup \{\bot\}, \max, +)$*, and to deterministic automata with many registers storing elements of* $\mathbb{N}_\infty$*, allowing the binary* max *operation and unary incrementation operation.*

The goal of this paper is to find effective algebraic characterisations of functions computable by min- and max-automata amongst all functions computable by B- and S-automata. These characterisations are up to equivalence of cost functions.

2.2 Theory of Regular Cost Functions

Throughout this paper, fix a finite input alphabet Σ. Given two functions $f, g : \Sigma^* \to \mathbb{N}_\infty$, we write $f \approx g$ if for all $X \subseteq \Sigma^*$, if g is bounded over X (meaning $\sup g|_X < \infty$), then f is bounded over X, and vice-versa. A *cost function* over the alphabet Σ is an equivalence class of $\approx$. Let $[f]$ denote the equivalence class of $f : \Sigma^* \to \mathbb{N}_\infty$. We will often identify a cost function with any of its representatives and say that f is a cost function, implicitly talking about $[f]$.

Regular cost functions. A cost function is *regular* if it is the cost function of the function computed by some B-automaton. For example, the (equivalence classes of the) functions described in Examples 2.1 and 2.2 are regular cost functions. It turns out [6] that B- and S-automata define equal classes of cost functions (see Theorem 2.15 below). Not every cost function is regular – indeed, there are uncountably many cost functions.

The reason we prefer to study cost functions computed by B-automata rather than the functions themselves is due to the following results, and to the fact that information about boundedness properties suffice in the contexts we will be interested in.

▶ **Theorem 2.5** (Krob [13]). *Given two min-automata* $\mathcal{A}$ *and* $\mathcal{B}$*, it is undecidable whether the functions* $[\![\mathcal{A}]\!]$ *and* $[\![\mathcal{B}]\!]$ *are equal.*

▶ **Theorem 2.6** (Colcombet [6]). *Given two B- or S-automata* $\mathcal{A}$ *and* $\mathcal{B}$*, it is decidable whether the functions* $[\![\mathcal{A}]\!]$ *and* $[\![\mathcal{B}]\!]$ *define the same cost function.*

If we take $\mathcal{B}$ in the theorem above to be such that $[\![\mathcal{B}]\!](w) = 0$ for all words w, we see that in particular, it is decidable whether the function $[\![\mathcal{A}]\!]$ is bounded over all words. The limitedness problem for B-automata easily reduces to this problem.

Cost regular expressions. Cost regular expressions are weighted extensions of classical regular expressions. They come in two forms, B-expressions and their dual S-expressions. A *B-expression* is given by the grammar,

$$E ::= a \in \Sigma \mid \emptyset \mid E \cdot E \mid E + E \mid E^{\leq n} \mid E^*,$$

where n is a variable (there is only one variable available). Note that by substituting $k \in \mathbb{N}$ for n in a cost regular expression E, denoted by $E[k \to n]$, one obtains a regular expression of finite words. Given a B-expression E the cost function computed by E is defined as:

$$[\![E]\!](u) = \inf\{k \mid u \in E[k \to n]\}.$$

Similarly one defines S-expressions, with the difference that we are allowed to use $> n$ instead of $\leq n$, and at the end we take sup instead of inf.

Both kinds of expressions define exactly all regular cost functions. Indeed, it is not difficult to convert between B-expressions and B-automata (and between S-expressions and S-automata), similarly to the conversions between regular expressions and finite automata.

▶ **Example 2.7.** The cost function $f_{\max}$ is defined by the B-expression $(a^{\leq n}b)^*$ and the S-expression $(a^*b)^*a^{>n}(ba^*)^*$. Dually, the cost function $f_{\min}$ is defined by the B-expression $(a^*b)^*a^{\leq n}(ba^*)^*$ and the S-expression $(a^{>n}b)^*a^{>n}$.

Cost monadic second order logic. We recall the basics of monadic second order logic (abbreviated as MSO) over words. The formulas use first order variables $x, y, z \ldots$ that range over positions of the word and second order variables $X, Y, Z \ldots$ that range over sets of positions of the word. MSO formulas are build using the atomic predicates $x \leq y$, $x \in X$, $a(x)$ (denoting that the label at position x is a, where $a \in \Sigma$), the connectives $\neg\varphi$, $\varphi \vee \psi$, $\varphi \to \psi$ and quantifiers $\exists x.\varphi$, $\forall x.\varphi$ (ranging over single elements) and $\exists X.\varphi$,$\forall X.\varphi$ (ranging over sets of elements). *Cost monadic second order logic* (cost-MSO) extends the logic by allowing formulas of the form $|X| \leq n$, where $|X|$ denotes the size of the set X and n is a fixed variable ranging over the natural numbers, with the restriction that they occur only positively (under even number of negations). Given a cost-MSO formula $\varphi(n)$ the cost function defined by $\varphi(n)$ is

$$[\![\varphi]\!](u) = \inf\{n \mid u, n \models \varphi\}.$$

▶ **Example 2.8.** The cost function $f_{\min}$ is expressed by $\exists X.block_a(X) \wedge (|X| \leq n)$, where $ablock_a(X)$ is the first-order formula expressing that X is a maximal block of consecutive a's. Dually, $f_{\max}$ is expressed by the formula $\forall X.block_a(X) \to (|X| \leq n)$.

Stabilisation monoids. Recall that if $\mathbf{M}$ is a finite monoid, $h : \Sigma \to \mathbf{M}$ is a mapping, and F is a subset of $\mathbf{M}$, then the triple $(\mathbf{M}, h, F)$ defines a regular language $L = \hat{h}^{-1}(F)$, where $\hat{h} : \Sigma^* \to \mathbf{M}$ is the unique homomorphism extending h. Conversely, any regular language L is induced by some triple $(\mathbf{M}, h, F)$ of this form. In this section we recall how this correspondence lifts to regular cost functions, by replacing finite monoids to stabilisation monoids. Let $E(\mathbf{M}) = \{e \in M | ee = e\}$ denote the set of idempotents of the monoid $\mathbf{M}$.

A *stabilisation monoid* $\mathbf{M} = \langle M, \cdot, \leq, \sharp \rangle$ is a finite monoid equipped with a partial order $\leq$ and an operation $\sharp : E(\mathbf{M}) \to E(\mathbf{M})$ (called stabilisation), satisfying the following axioms:

$$a \cdot x \leq b \cdot y \quad \text{for} \quad a \leq b, x \leq y \tag{1}$$
$$e^\sharp \leq f^\sharp \quad \text{for} \quad e \leq f, e, f \in E(\mathbf{M}) \tag{2}$$
$$(a \cdot b)^\sharp = a \cdot (b \cdot a)^\sharp \cdot b \quad \text{for} \quad a, b \in \mathbf{M} \text{ such that } a \cdot b, b \cdot a \in E(\mathbf{M}) \tag{3}$$
$$e^\sharp \leq e \quad \text{for} \quad e \in E(\mathbf{M}) \tag{4}$$
$$(e^\sharp)^\sharp = e^\sharp \quad \text{for} \quad e \in E(\mathbf{M}) \tag{5}$$

▶ **Example 2.9.** Any finite monoid can be seen as a stabilisation monoid, in which $e^\sharp = e$ for every idempotent, and where the order is trivial, i.e., $x \leq y$ iff $x = y$.

▶ **Example 2.10.** Every min-automaton $\mathcal{A}$ defines a finite transition stabilisation monoid, defined as follows. Let $\mathcal{T}$ denote the *tropical semiring* or the $(min, +)$-semiring, with domain $\mathbb{N}_\infty$ and min playing the role of addition, and $+$ of multiplication. We equip $\mathbb{N}_\infty$ with the usual topology, in which ∞ is the limit of every strictly increasing sequence.

First we define an infinite monoid, parametrised by a finite set of states Q. Let $\mathbf{M}_Q$ denote the set of $Q \times Q$ matrices with entries from $\mathcal{T}$. Matrices can be multiplied using the semiring operations of $\mathcal{T}$, and the set of matrices $\mathbf{M}_Q$ inherits the product topology from $\mathcal{T}$, i.e., a sequence $(M_n)_{n=1}^\infty$ of matrices is convergent if $M_n[p, q]$ is convergent in $\mathbb{N}_\infty$ for each $p, q \in Q$. Matrix multiplication is continuous, and moreover, one can show that for every matrix $M \in \mathbf{M}_Q$, when $n \to \infty$, the sequence $M^{n!}$ is convergent to a matrix denoted $M^\sharp$; moreover, the mapping $M \mapsto M^\sharp$ is continuous.

An automaton $\mathcal{A}$ with state space Q defines a mapping $h : \Sigma^* \to M$ which assigns to a word $w \in \Sigma$ the matrix $h(w)$ such that for two states p, q of $\mathcal{A}$, the value $h(w)[p, q]$ is the minimal cost of a run of $\mathcal{A}$ over w which starts in state p, ending in state q. The mapping h is a monoid homomorphism.

Define an equivalence relation $\sim_1$ on $\mathbf{M}_Q$ so that two matrices are equivalent iff they yield the same result when each finite, positive entry is replaced by 1. Define $\mathbf{M}_Q^1$ to be the set of $\sim_1$-equivalence classes. We identify an element of $\mathbf{M}_Q^1$ with its unique representative which is a matrix with entries in $\{0, 1, \infty\}$. It turns out [17] that the equivalence $\sim_1$ preserves multiplication and the operation $\sharp$. It follows that $\mathbf{M}_Q^1$ inherits the structure of a monoid, and also an operation $\sharp$; we restrict this operation only to idempotents (for a non-idempotent M, we can still recover $M^\sharp$ as $E^\sharp$, where E is the idempotent power of M).

One way to compute a product of two matrices $M, N \in \mathbf{M}_Q^1$ is to take a min-automaton $\mathcal{A}$ with states Q and with $h(a) = M$ and $h(b) = N$; then $M \cdot N$ is obtained by substituting 1 for every finite positive number in $h(ab)$. Similarly, if $M \in \mathbf{M}_Q^1$ is idempotent, then $M^\sharp$ is obtained by taking an automaton $\mathcal{A}$ with $h(a) = M$ and writing $M^\sharp[p, q] = 0$ if for arbitrarily large n, $h(a^n)[p, q] = 0$, otherwise $M^\sharp[p, q] = 1$ if for arbitrarily large n $h(a^n)[p, q]$ remains bounded, and finally, $M^\sharp[p, q] = \infty$ if $h(a^n)[p, q]$ converges to ∞ when $n \to \infty$. The computations can be performed purely mechanically (see Appendix). For example, for the automaton from Example 2.1, we compute $h(a) \cdot h(b)$ and $h(a)^\sharp$ in $\mathbf{M}_Q^1$:

$$\begin{bmatrix} 0 & \infty & \infty \\ \infty & 1 & \infty \\ \infty & \infty & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & \infty & \infty \\ 0 & \infty & \infty \\ \infty & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & \infty & \infty \\ 1 & \infty & \infty \\ \infty & 0 & 0 \end{bmatrix}, \quad \begin{bmatrix} 0 & \infty & \infty \\ \infty & 1 & \infty \\ \infty & \infty & 0 \end{bmatrix}^\sharp = \begin{bmatrix} 0 & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & 0 \end{bmatrix}$$

For $M, N \in \mathbf{M}_Q^1$, write $M \preceq N$ if there is a sequence of representatives of N which converges to a representative of M. In other words, M can be obtained from N by replacing some 1's

by ∞'s. (The order $\preceq$ is the specialisation order associated to the quotient topology on $\mathbf{M}_Q^1$ inherited from $\mathbf{M}_Q$.)

It can be shown [17] that $\langle \mathbf{M}_Q^1, \cdot, \preceq, \sharp \rangle$ is a stabilisation monoid. The axiom (1) is a remnant of continuity of multiplication in $\mathbf{M}_Q$, (2) – of continuity of the operation $M \mapsto M^\sharp$, (3) – of associativity, (4) – of the fact that $M^\sharp$ is the limit of $M^{n!}$, and (5) – of an analogous property which holds in $\mathbf{M}_Q$.

Let $h^1 : \Sigma \to \mathbf{M}_Q^1$ be defined so that for a letter $a \in \Sigma$, the matrix $h^1(a)$ is equal to $h(a)$ (this is a matrix with entries in $\{0, 1, \infty\}$). The mapping h^1 encodes the transition structure of the automaton $\mathcal{A}$. Let I be the set of matrices $M \in \mathbf{M}_Q^1$ such that $M[p, q] = \infty$ for all initial states p and accepting states q of $\mathcal{A}$. The set I encodes the acceptance condition of $\mathcal{A}$. The cost function $[\![\mathcal{A}]\!]$ defined by $\mathcal{A}$ can be recovered from the triple $(\mathbf{M}_Q^1, h^1, I)$, as we will now describe.

▶ **Definition 2.11** (Computation tree)**.** An *n-computation tree* t is a finite rooted ordered unranked tree in which each node x has an associated *output* in $\mathbf{M}$ and is of one of four types:

Leaf x has no children and has an associated *label* $a \in \Sigma$, and the output of x is $h(a)$;

Binary node x has exactly two children and and the output of x is the product of the output of the first child and the output of the second child;

Idempotent node x has k children with $k \leq n$ and for some idempotent $e \in \mathbf{M}$, the output of each child is equal to e and the output of x is equal to e.

Stabilisation node x has k children with $k > n$ and for some idempotent $e \in \mathbf{M}$, the output of each child is equal to e and the output of x is equal to $e^\sharp$.

The *input* of the tree t is the word formed by the labels of the leaves of the tree, read left to right. The *output* of the tree t is the output of the root, and the neutral element of $\mathbf{M}$ if t is the empty tree.

An *ideal* in a stabilisation monoid $\mathbf{M}$ is a subset I which is downward-closed, i.e., $x \leq y$ and $y \in I$ imply $x \in I$. Let $h : \Sigma \to M$ be a mapping from a finite alphabet to a stabilisation monoid $\mathbf{M}$, and let I be an ideal in $\mathbf{M}$. The triple $(\mathbf{M}, h, I)$ induces a cost function, denoted $[\![\mathbf{M}, h, I]\!]$, and defined as follows. For a fixed height $k \in \mathbb{N}$, let

$$[\![\mathbf{M}, h, I]\!]_k (w) = \inf\{n \mid \text{there is a } n\text{-computation on } w \text{ with output in } \mathbf{M} \setminus I, \text{ height} \leq k\}.$$

It turns out [6] that the cost function $[\![\mathbf{M}, h, I]\!]_k$ does not depend on the choice of $k \geq 3|M|$. We define $[\![\mathbf{M}, h, I]\!]$ as $[\![\mathbf{M}, h, I]\!]_k$ for $k = 3|M|$.

▶ **Example 2.12.** Let $\mathbf{M}_{\max} = \langle \{1, a, b, a^\sharp\}, \cdot, \sharp, \leq \rangle$ be the stabilisation monoid with identity 1, zero $a^\sharp$, such that $ab = ba = b = bb = b^\sharp$, $aa = a$ and $a^\sharp \leq a$. The monoid $\mathbf{M}_{\max}$ defines the function $f_{\max}$ with ideal $\{a^\sharp\}$ and mapping $h(a) = a$ and $h(b) = b$.

▶ **Example 2.13.** Let $\mathbf{M}_{\min} = \langle \{1, a, a^\sharp, b, ba^\sharp, a^\sharp b, 0\}, \cdot, \sharp, \leq \rangle$ be the stabilisation monoid with identity 1, zero 0, product $a^\sharp ba^\sharp = a^\sharp$, $ba^\sharp b = b = ab = ba$, $bb = 0$, stabilisation $(ba^\sharp)^\sharp = ba^\sharp, (a^\sharp b)^\sharp = a^\sharp b$, and order $a^\sharp \leq a, a^\sharp b \leq b, ba^\sharp \leq b \leq 0$. The monoid $\mathbf{M}_{\min}$ defines the function $f_{\min}$ with ideal $\{a^\sharp\}$ and mapping $h(a) = a$ and $h(b) = b$.

▶ **Proposition 2.14.** *For a min-automaton $\mathcal{A}$ with states Q, define $(\mathbf{M}_Q^1, h^1, I)$ as in Example 2.10. Then I is an ideal and the cost functions $[\![\mathbf{M}_Q^1, h^1, I]\!]$ and $[\![\mathcal{A}]\!]$ are equal.*

Example 2.10 and Proposition 2.14 are a special case of a more general construction for B- and S-automata [17].

Closure properties. Regular cost functions have several closure properties, described below. The fundamental cost function is the function count : $\{a,b\} \to \mathbb{N}_\infty$, defined by $\text{count}(u) = |u|_a$, the number of occurrences of letter a. A regular language $L \subseteq \Sigma^*$ is viewed as a (regular) cost function mapping a word w to 0 if $w \in L$ and to ∞ if $w \notin L$. Note that since regular languages are closed under complements, exchanging 0 with ∞ in this definition would give the same class of cost functions.

Let $\mathcal{C}$ be a class of cost functions, possibly over different input alphabets. We define several closure properties for the class $\mathcal{C}$:

composition with morphisms if the cost function $f : \Sigma^* \to \mathbb{N}_\infty$ is in $\mathcal{C}$ and $\alpha : \Gamma^* \to \Sigma^*$ is a morphism, then the cost function $\alpha \circ f : \Gamma^* \to \mathbb{N}_\infty$ is in $\mathcal{C}$;

min if for any two cost functions $f, g : \Sigma^* \to \mathbb{N}_\infty$ in $\mathcal{C}$, the function $w \mapsto \min(f(w), g(w))$ also belongs to $\mathcal{C}$;

max defined dually, with max instead of min;

min with regular languages if for any cost function $f : \Sigma^* \to \mathbb{N}_\infty$ and regular language g viewed as a cost function (see above) the function $w \mapsto \min(f(w), g(w))$ also belongs to $\mathcal{C}$;

sup-projections if the cost function $f : \Sigma^* \to \mathbb{N}_\infty$ is in $\mathcal{C}$ and $\alpha : \Sigma^* \to \Gamma^*$ is a morphism, then the cost function $v \mapsto \sup\{f(w) : w \in \alpha^{-1}(v)\}$ is in $\mathcal{C}$;

inf-projections defined dually, with inf instead of sup.

The main theorem of regular cost functions. The theorem below shows that all the introduced notions give rise to the same class of cost functions, namely regular cost functions.

▶ **Theorem 2.15** (Colcombet [7]). *The following formalisms are effectively equivalent as recognisers of regular cost functions: B-automata, S-automata, B-expressions, S-expressions, cost MSO formulas, and stabilisation monoids. Moreover, the class of regular cost functions is the smallest class of cost functions which is closed under min, max, inf-projections, sup-projections, contains the function count, and all regular languages.*

We may now specify the goal of this paper more precisely. Among all regular cost functions, we characterise those which are of the form $[\![\mathcal{A}]\!]$ for a min- or max-automaton $[\![\mathcal{A}]\!]$. Our characterisations (Theorem 3.2 and Theorem 4.2) are in terms of cost regular expressions, fragments of cost MSO, and stabilisation monoids. Moreover, the characterisations are effective. In algebraic language theory, the usual way of providing effective characterisations is by means of certain algebraic conditions satisfied by the syntactic monoid. For this reason, we need to recall the notion of a syntactic stabilisation monoid.

Syntactic stabilisation monoid. A homomorphism of stabilisation monoids is a mapping $h : \mathbf{M} \to \mathbf{N}$ of stabilisation monoids which is monotone (i.e., $u \leq v \implies h(u) \leq h(v)$), preserves multiplication (i.e., $h(u \cdot v) = h(u) \cdot h(v)$) and stabilisation (i.e., $h(e^\sharp) = h(e)^\sharp$ for every idempotent $e \in \mathbf{M}$).

▶ **Theorem 2.16** ([11]). *Let $f : \Sigma^* \to \mathbb{N}_\infty$ be a regular cost function. There is a unique (up to isomorphism) triple $(\mathbf{M}_f, h_f, I_f)$ recognising f with the following property. For any triple $(\mathbf{M}, h, I)$ recognising f, there is a unique surjective homomorphism $\phi : \mathbf{M} \to \mathbf{M}_f$ such that $h_f = \phi \circ h$ and $I = \phi^{-1}(I_f)$. $\mathbf{M}_f$ is called the syntactic stabilisation monoid of f.*

Moreover, for any triple $(\mathbf{M}, h, I)$ recognising f, the triple $(\mathbf{M}_f, h_f, I_f)$ can be computed in polynomial time from $(\mathbf{M}, h, I)$.

Idempotent power. It is a standard fact that every element a in a finite monoid has a unique idempotent power, denoted a^ω. It can be shown that $a^\omega = a^{n!}$ where n is the size of the monoid. We use the notation $a^{\omega\sharp}$ to denote the element $(a^\omega)^\sharp$.

3 Max-automata

In this section we characterise cost functions computed by max-automata. First we introduce the algebraic condition that characterises this class.

▶ **Definition 3.1** (Max-property). Let $\mathbf{M} = \langle M, \cdot, \sharp, \leq \rangle$ be a stabilisation monoid and $I \subseteq M$ be an ideal. The pair M, I has *Max-property* if for every $u, v, x, y, z \in M$,

1. if $xu^{\omega\sharp}yv^{\omega\sharp}z \in I$, then either $xu^{\omega\sharp}yv^{\omega}z \in I$, or $xu^{\omega}yv^{\omega\sharp}z \in I$, and
2. if $x(uy^{\omega\sharp}v)^{\omega\sharp}z \in I$, then either $x(uy^{\omega\sharp}v)^{\omega}z \in I$, or $x(uy^{\omega}v)^{\omega\sharp}z \in I$.

Now we present the main theorem of Section 3.

▶ **Theorem 3.2.** *The following are effectively equivalent for a regular cost function f:*

1. *f is accepted by a max-automaton,*
2. *f is definable by a formula of the form $\psi \wedge \forall X\ (\varphi(X) \to |X| \leq n)$ where ψ, φ are MSO formulas, i.e. they do not contain cost predicates,*
3. *f is in the smallest class (call it MAX) of cost functions that contains the function count and regular languages, and closed under min with regular languages, max, sup-projections, and composition with morphisms,*
4. *f is equivalent to $[\![\mathbf{M}, h, I]\!]$ for some mapping h from Σ to a stabilisation monoid $\mathbf{M}$ with ideal I having Max-property,*
5. *The syntactic stabilisation monoid and ideal of f has Max-property,*
6. *f is definable by an S-regular expression of the form $h + \sum_i e_i$ where h is a regular expression and each e_i is of the form $ef^{>n}g$ where e, f and g are regular expressions.*

▶ **Example 3.3.** The stabilisation monoid $\mathbf{M}_{\max}$ with ideal $\{a^\sharp\}$ recognising $f_{\max}$ has Max-property. But $\mathbf{M}_{\min}$ with ideal $I = \{a^\sharp\}$ recognising $f_{\min}$ violates the Max-property since $a^\sharp ba^\sharp = a^\sharp$ is in I, but neither of $a^\sharp ba = a^\sharp b, aba^\sharp = ba^\sharp$ belongs to I.

Since $\mathbf{M}_{\min}$ is the syntactic monoid of the function $f_{\min}$, the example above implies that $f_{\min}$ is not accepted a max-automata; in particular, max-automata are not closed under inf-projection. Similarly one verifies that cost functions computed by max automata are not closed under min (for instance the functions $u \in \{a, b\}^* \to |u|_a$ and $u \in \{a, b\}^* \to |u|_b$, as well as their max, is in MAX, but not their min).

Proof sketch. We sketch the implications: $1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 1$.

$1 \to 2$ is by observing that there is a cost MSO formula encoding the runs of a 1-counter S-automata, of the form described in item 2.

To prove $2 \to 3$, it is enough to observe that all the subformulas (including the formula itself) of $\psi \wedge \forall X\ (\varphi \to |X| \leq n)$ (where ψ, φ are cost free) define cost functions in the class MAX. For this, we remark that the cost function count is in MAX, so it is also the case of $[\![|X| \leq n]\!]$, using the usual semantic for formulae with free variables, i.e. enriching the alphabet with a $\{0, 1\}$-component. Moreover MAX is closed under the operation min with regular languages, composition with morphisms (together they imply $[\![\neg\varphi \vee |X| \leq n]\!]$ is in MAX) and sup-projections (hence $[\![\forall X\ (\varphi(X) \to |X| \leq n)]\!]$ is in MAX). Finally closure under max implies that the formula defines a function in MAX.

To show $3 \to 4$, we show that the monoid constructions corresponding to the operations of max, min with a regular language, sup-projection and composition with a morphism preserve the Max-property, and also that the syntactic stabilisation monoids computing the function count as well as regular languages have the Max-property.

$4 \to 5$ follows from Theorem 2.16. Implication $5 \to 6$ is the hardest part of the theorem. By definition, the value of a word w given by the cost function $[\![\mathbf{M}, h, I]\!]$ is the maximum $n \in \mathbb{N}_\infty$ such that there is an n-computation tree of height $3|M|$ with input w and output in the ideal I. A way to attain this value using a cost regular expression is to write an expression that encodes all such computation trees by induction on the tree height; binary nodes translate to concatenation, idempotent nodes stand for Kleene star, and stabilisation nodes translate to the operator $>n$. But as such, this idea results in an expression with multiple occurrences of $>n$ (as there could be many stabilisations in the tree). This difficulty is circumvented by showing that it is sufficient to consider trees with only one stabilisation node. This is achieved by repeated use of the Max-property of the monoid $\mathbf{M}$ and ideal I.

For $6 \to 1$, note that the standard construction from cost regular expressions to S-automata (analogous to the translation from regular expressions to finite state automata) applied to the cost regular expressions of the form described in item 6, gives a max-automaton.

We note that all the transformations are effectively computable. ◀

Given a stabilisation monoid $\mathbf{M}$ and ideal I it is computable in polynomial time whether $\mathbf{M}, I$ has Max-property. Hence by Theorem 2.15 and Theorem 2.16 we obtain,

▶ **Theorem 3.4.** *It is decidable if a regular cost function satisfies a condition of Theorem 3.2.*

4 Min-automata

In this section we characterise cost functions definable by min-automata.

▶ **Definition 4.1** (Min-property)**.** We define the relation $\mathcal{R} \subseteq \mathbf{M} \times \mathbf{M}$ as the smallest reflexive relation satisfying the following implications: (1) if $x \mathcal{R} y$ and $a \mathcal{R} b$, then $(x \cdot a) \mathcal{R} (y \cdot b)$, and (2) if $x \mathcal{R} y$ then $x^\omega \mathcal{R} y^{\omega\sharp}$ and $x^{\omega\sharp} \mathcal{R} y^{\omega\sharp}$. A monoid $\mathbf{M}$ satisfies the *Min-property* if for all elements x, y, if $x^{\omega\sharp} \mathcal{R} y$, then $x^{\omega\sharp} = x^{\omega\sharp} y x^{\omega\sharp}$.

▶ **Theorem 4.2.** *The following are effectively equivalent for a regular cost function f:*

1. *f is accepted by a min-automaton,.*
2. *f is definable by a formula of the form $\exists X\ (\varphi(X) \wedge |X| \leq n)$ where φ does not contain any cost predicates,*
3. *f belongs to the smallest class of cost functions containing count and regular languages that is closed under min, max and inf-projections,*
4. *f is recognised by a stabilisation monoid $\mathbf{M}$ with Min-property,*
5. *The syntactic stabilisation monoid of f has Min-property,*
6. *f is accepted by a B-regular expression E that is generated by the grammar*

$$E := F \mid E + E \mid E \cdot E \mid E^{\leq n} \qquad\qquad F := a \mid F + F \mid F \cdot F \mid F^*$$

   i.e. any subexpression of E of the form F^* is a regular expression (without $X^{\leq n}$),
7. *f is accepted by a B-automaton without reset.*

▶ **Example 4.3.** The monoid $\mathbf{M}_{\max}$ of $f_{\max}$ violates the Min-property since $b = b^\sharp \mathcal{R} a^\sharp$ (to see this, observe $a \mathcal{R} a^\sharp, b \mathcal{R} b$ and hence $ab = b \mathcal{R} a^\sharp = a^\sharp b$), but $b = b^\sharp \neq b^\sharp a^\sharp b^\sharp = b a^\sharp b = a^\sharp$. On the contrary, it can be verified that the monoid $\mathbf{M}_{\min}$ has Min-property.

Whereas max automata are not closed under min, min automata are closed under max. The class MIN falls only short of sup-projections, comparing to all regular cost functions.

The Min-property can be expressed in terms of identities, as follows. Consider the set T of terms involving variables from an infinite set of variables, a binary multiplication operation and unary operations ω and $\omega\sharp$. Let $\mathcal{R}$ be the smallest binary relation on T that is reflexive and satisfies the implications 1 and 2 from Definition 4.1. Then, Min-property is expressed as the family of identities $x^{\omega\sharp} = x^{\omega\sharp}yx^{\omega\sharp}$, indexed by pairs of terms x, y such that $x^{\omega\sharp}\mathcal{R}y$.

Proof sketch. We sketch the implications: $1 \to 2 \to 3 \to 1 \to 4 \to 5 \to 1 \to 6 \to 7 \to 1$.

$1 \to 2$, $2 \to 3$, $6 \to 7$ are analogous to the corresponding cases in Theorem 3.2, and $3 \to 1$ is demonstrated by verifying that all functions and operations in item 3 can be carried out in the framework of min-automata.

$1 \to 4$ proceeds by studying the transition monoid of a min-automaton, described in Example 2.10, and checking that the equation of the Min-property is verified for any $x \preceq y$ (where $\preceq$ is the ordering used in the transition monoid). In particular, the equation is true for the relation $\mathcal{R}$, which is a subset of any stabilisation order.

$4 \to 5$ uses the fact that Min-property is equational, so is preserved by quotients.

$5 \to 1$ is obtained by performing substitutions in computation trees: any idempotent node that is the descendant of a stabilisation node can be transformed into a stabilisation node itself. We get a normal form for computation trees over monoids of this fragment, that we call frontier trees. We then design min-automata that witness the existence of frontier trees for any input word.

We show $1 \to 6$ by adapting the classical automaton-to-expression algorithm performing inductive state removal. Here, new transitions are labeled by regular expressions together with an action from $\{\varepsilon, \mathtt{i}\}$. When the classical algorithm produces a Kleene star, we do so if the looping transition is labeled ε; otherwise if it is $\mathtt{i}$, we replace the Kleene star by $\leq n$, and the resulting edge is again labeled $\mathtt{i}$. This way, a Kleene star cannot be produced on top of a subexpression containing a $\leq n$.

We show $6 \to 7$ by induction on the structure of the expression. We build an ad-hoc generalisation of the expression$\to$ automaton algorithm, and show that the output B-automaton contains no reset.

Finally, $7 \to 1$ is obtained by observing that in the absence of resets, increments performed on k distinct counters can be performed on a single counter, by increasing the result at most k times. ◀

By a saturation algorithm, we can verify whether a given stabilisation monoid has Min-property in polynomial time. Hence by Theorem 2.15 and Theorem 2.16 we obtain:

▶ **Theorem 4.4.** *It is decidable if a regular cost function satisfies a condition of Theorem 4.2.*

5 Conclusion

We studied two dual classes of cost functions, defined by min-automata (also called distance automata), and max-automata. Both these classes have been studied in detail in other works. We showed that these classes enjoy many equivalent characterisations – such as restrictions of automata, logics, and expressions, and algebraic conditions. In both cases, the algebraic characterisation leads to decidability of membership in the class, in the spirit

of Schützenberger's seminal work on star-free languages [16]. Combining with the finite-index characterisation of regular cost functions from [17], we obtain a purely algebraic characterisation of cost functions defined by min- or max-automata.

References

1 Parosh Aziz Abdulla, Pavel Krcál, and Wang Yi. R-automata. In *CONCUR 2008*, volume 5201, pages 67–81, 2008.

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 Mikolaj Bojanczyk. Weak MSO with the unbounding quantifier. *Theory Comput. Syst.*, 48(3):554–576, 2011.

4 Mikolaj Bojańczyk and Thomas Colcombet. Bounds in ω-regularity. In *LICS 06*, pages 285–296, 2006.

5 Mikolaj Bojanczyk and Szymon Torunczyk. Deterministic automata and extensions of weak mso. In *FSTTCS*, pages 73–84, 2009.

6 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, Languages and Programming, Internatilonal Collogquium, ICALP 2009, Proceedings, Part II*, pages 139–150, 2009.

7 Thomas Colcombet. *Fonctions régulières de coût*. Habilitation thesis, Université Paris Diderot–Paris, 2013.

8 Thomas Colcombet. Regular cost functions, part I: logic and algebra over words. *Logical Methods in Computer Science*, 9(3), 2013.

9 Thomas Colcombet and Laure Daviaud. Approximate comparison of distance automata. In *STACS 2013*, volume 20 of *LIPIcs*, pages 574–585, 2013.

10 Thomas Colcombet, Laure Daviaud, and Florian Zuleger. Size-change abstraction and max-plus automata. In *MFCS 2014*, volume 8634 of *Lecture Notes in Computer Science*, pages 208–219, 2014.

11 Thomas Colcombet, Denis Kuperberg, and Sylvain Lombardy. Regular temporal cost functions. *Automata, Languages and Programming*, pages 563–574, 2010.

12 Daniel Kirsten. Distance desert automata and the star height problem. *ITA*, 39(3):455–509, 2005.

13 Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *Internat. J. Algebra Comput.*, 4(3):405–425, 1994.

14 Denis Kuperberg. Linear temporal logic for regular cost functions. *Logical Methods in Computer Science*, 10(1), 2014.

15 Martin Lang, Christof Löding, and Amaldev Manuel. Definability and transformations for cost logics and automatic structures. In *MFCS 2014*, volume 8634 of *Lecture Notes in Computer Science*, pages 390–401. Springer, 2014.

16 M.P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190 – 194, 1965.

17 Szymon Torunczyk. *Languages of profinite words and the limitedness problem*. PhD thesis, PhD thesis, University of Warsaw, 2011.

Varieties of Cost Functions

Laure Daviaud[1], Denis Kuperberg[2], and Jean-Éric Pin[3]

1 **Aix Marseille Université, CNRS, LIF UMR 7279 and LIP, ENS de Lyon, France**
laure.daviaud@ens-lyon.fr
2 **ONERA/DTIM, IRIT, University of Toulouse, France**
denis.kuperberg@gmail.com
3 **LIAFA, CNRS and Université Paris Diderot, France**
jean-eric.pin@liafa.univ-paris-diderot.fr

Abstract

Regular cost functions were introduced as a quantitative generalisation of regular languages, retaining many of their equivalent characterisations and decidability properties. For instance, stabilisation monoids play the same role for cost functions as monoids do for regular languages. The purpose of this article is to further extend this algebraic approach by generalising two results on regular languages to cost functions: Eilenberg's varieties theorem and profinite equational characterisations of lattices of regular languages. This opens interesting new perspectives, but the specificities of cost functions introduce difficulties that prevent these generalisations to be straightforward. In contrast, although syntactic algebras can be defined for formal power series over a commutative ring, no such notion is known for series over semirings and in particular over the tropical semiring.

1998 ACM Subject Classification F.4.3 Algebraic language theory

Keywords and phrases Cost functions, regular language, varieties, syntactic algebra

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.30

1 Introduction

Quantitative extensions of regular languages have been studied for over 50 years. Most of them rely on the early work of Schützenberger [25, 26, 27], who extended Kleene's theorem to formal power series over a semiring. A very nice presentation of this theory can be found in the book of Berstel and Reutenauer [5]. In this setting, weighted automata play the role of automata and weighted logic was introduced as an attempt to generalise Büchi's characterisation of regular languages in monadic second order logic. See the handbook [12] for an overview and further references.

However, this theory also suffers some weaknesses. For instance, the equality problem for rational series with multiplicities in the tropical semiring is undecidable [15], a major difference with the equality problem for regular languages, which is decidable. To overcome this problem and other related questions, Colcombet introduced the notion of regular cost functions [9], an other quantitative generalisation of regular languages. Cost functions are formally defined as equivalence classes of power series with coefficients in the semiring $\mathbb{N} \cup \{\infty\}$. This equivalence does not retain the exact values of the coefficients of the series but measures boundedness in some precise way. Thus cost functions are less general than power series, but are still more general than languages, which can be viewed as cost functions associated with their characteristic functions.


© Laure Daviaud, Denis Kuperberg, and Jean-Éric Pin;
licensed under Creative Commons License CC-BY
33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 30; pp. 30:1–30:14

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


This approach proved to be very successful. It leads to simplified proofs of several major results related to boundedness (the limitedness problem of distance automata, Kirsten's proof of the star-height problem, etc.). Moreover, the equivalences on regular languages

regular languages $\Leftrightarrow$ *finite automata* $\Leftrightarrow$ *finite monoids* $\Leftrightarrow$ *monadic second order logic*

admit the following nontrivial extension

regular cost functions $\Leftrightarrow$ *cost automata* $\Leftrightarrow$ *stabilisation monoids* $\Leftrightarrow$ *cost monadic logic*.

Contributions

The aim of this paper is to show that the algebraic approach to regular languages also extends to the setting of cost functions. To this end, we change the recognising object of cost functions from *stabilisation monoid* [9] to a structure with better algebraic properties, called *stabilisation algebra*. This gives us a new way of interpreting cost functions, as particular sets of a free stabilisation algebra $F(A)$ on the alphabet A, generalising the set of words A^*. This allows us to extend the ordered version [19] of Eilenberg's varieties theorem [13], which gives a bijective correspondence between positive varieties of languages and varieties of finite ordered monoids (Theorem 5.5). We show that the profinite algebra $\widehat{F(A)}$ generalising profinite words is the dual of the lattice of regular cost functions. This leads to an extension of the duality results between profinite words and regular languages. In particular, we extend the equational approach to lattices of regular languages given in [14] (Theorem 7.5). Our approach not only subsumes the corresponding results on languages but it also gives a nice algebraic framework for the results of [11, 16]. A series of examples is given in Section 8.

Related work

Toruńczyk [30] also established a link between cost functions and profinite words, using a different approach. More precisely, Toruńczyk identifies a regular cost function with the set of profinite words that are limits of infinite sequences of words over which the function is bounded.

It is also interesting to compare these results to similar results on formal power series. Syntactic algebras of formal power series over a commutative ring were introduced by Reutenauer [22, 23], but no such notion is known for semirings. Reutenauer also extended Eilenberg's varieties theorem to power series over a commutative field. However, as shown in [24], equational theory only works for power series over finite fields.

Finally, let us mention two new promising approaches to recognisability, using respectively categories [1, 2] and monads [7, 8]. For the time being, these two approaches do not seem to apply to cost functions, but we hope our paper will serve as a test bench for future developments of this new point of view.

2 Regular Cost Functions and Stabilisation Monoids

In this section, we introduce the notions of cost functions and of stabilisation monoids. For a more complete and detailed presentation, the reader is referred to [10].

Let A be a finite alphabet and let $\mathcal{F}$ be the set of all functions from A^* to $\mathbb{N} \cup \{\infty\}$. Colcombet [9] introduced the following equivalence relation on $\mathcal{F}$: two elements f and g of $\mathcal{F}$ are *equivalent* (denoted $f \approx g$) if, for each subset S of A^*, f is bounded on S if and only if g is bounded on S. A *cost function* is a $\approx$-class. In practice, cost functions are always represented by one of their representatives in $\mathcal{F}$.

The equivalence relation $\approx$ behaves well with respect to the operations min and max, defined in the usual way. Indeed for all $f, g, h \in \mathcal{F}$, if $f \approx g$, then $\min(f, h) \approx \min(g, h)$ and $\max(f, h) \approx \max(g, h)$ [9]. It follows that the minimum and the maximum of two cost functions are well-defined notions.

▶ **Example 2.1.** Let $A = \{a, b\}$. Given a word u, let $|u|$ denote the length of u and $|u|_a$ the number of occurrences of the letter a in u. Let us define three functions f, g and h from A^* to $\mathbb{N} \cup \{\infty\}$ by setting $f(u) = |u|$, $g(u) = |u|_a$ and $h(u) = 2|u|_a$. Then g is equivalent to h and they represent the same cost function, whereas g is not equivalent to f. Indeed g is bounded on b^* and f is not since for all n, $g(b^n) = 0$ and $f(b^n) = n$.

The *characteristic function* of a language L on A^* is the function $\chi_L : A^* \to \mathbb{N} \cup \{\infty\}$ defined by $\chi_L(u) = 0$ if $u \in L$ and ∞ otherwise. The crucial observation that $\chi_L \approx \chi_{L'}$ if and only if $L = L'$ allows one to identify a language with the cost function defined by its characteristic function.

Stabilisation monoids were introduced in [9] in order to extend the classical notion of monoids recognising a language to the setting of cost functions. Recall that an *ordered monoid* is a set equipped with an associative binary product, a neutral element and an order compatible with the product, i.e., the conditions $x_1 \leqslant x_2$ and $y_1 \leqslant y_2$ imply $x_1 y_1 \leqslant x_2 y_2$. We let $E(M)$ denote the set of idempotents of a monoid M.

Following [9], we define a *stabilisation monoid* as an ordered monoid M together with a *stabilisation operator* $\sharp : E(M) \to E(M)$ satisfying the following properties:

- ($\mathbf{S}_1$) for all $s, t \in M$ such that $st \in E(M)$ and $ts \in E(M)$, one has $(st)^\sharp s = s(ts)^\sharp$,
- ($\mathbf{S}_2$) for all $e \in E(M)$, one has $(e^\sharp)^\sharp = e^\sharp e = e e^\sharp = e^\sharp \leqslant e$,
- ($\mathbf{S}_3$) for all $e, f \in E(M)$, $e \leqslant f$ implies $e^\sharp \leqslant f^\sharp$,
- ($\mathbf{S}_4$) $1^\sharp = 1$.

Given two stabilisation monoids M and N, a *morphism* φ from M to N is a monoid morphism which is order-preserving and $\sharp$-preserving: if $e \in E(M)$, then $\varphi(e)^\sharp = \varphi(e^\sharp)$.

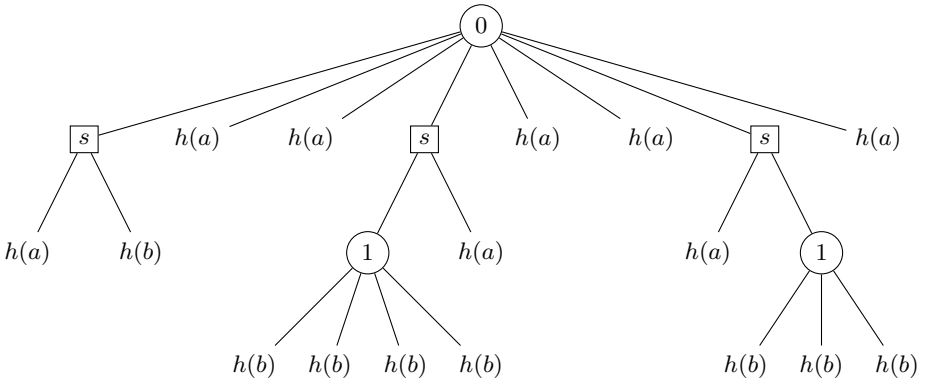
Just like finite (ordered) monoids recognise regular languages, finite stabilisation monoids recognise regular cost functions. However, the formal definition of recognition is more involved for cost functions than for languages and relies on the notion of factorisation trees. Let M be a stabilisation monoid and let $h : A \to M$ be a function, called the *labelling map*.

▶ **Definition 2.2.** Let $w = a_1 a_2 \cdots a_k$ be a word of A^* where each a_i is a letter. An *h-factorisation tree of threshold n* for w is a finite tree labelled by the elements of M and such that:

- ($\mathbf{T}_1$) the tree has exactly k leaves, labelled by $h(a_1), \ldots, h(a_k)$, respectively,
- ($\mathbf{T}_2$) each binary node is labelled by the product of its left child's label by its right child's label,
- ($\mathbf{T}_3$) if a node has arity > 2, then all its children are labelled by the same idempotent e. If the arity of the node is $\leqslant n$, then the node is labelled by e, otherwise it is labelled by $e^\sharp$.

▶ **Example 2.3.** Let S_1 be the stabilisation monoid containing three idempotent elements $\{1, s, 0\}$ with $0 < s < 1$, $0^\sharp = s^\sharp = 0$, 0 is a zero of the monoid and 1 the neutral element. Let $h(a) = s$ and $h(b) = 1$. See on page 4 a factorisation tree of threshold 5 for the word *abaabbbbaaaabbba*. A variant of Simon's factorisation theorem [9, 29] guarantees the existence of trees of bounded height to evaluate input words. More precisely, for each labelling function

$h : A \to M$, there is a positive integer K $(= 3|M|)$ such that for all words w and for all integers $n \geqslant 3$, there is an h-factorisation tree of threshold n for w with height at most K.



We can now give the formal definition of a regular cost function recognised by a finite stabilisation monoid. Recall that a subset D of a partially ordered set is a *downset* if the conditions $t \in D$ and $s \leqslant t$ imply $s \in D$.

▶ **Definition 2.4.** Let M be a finite stabilisation monoid, $h : A \to M$ a labelling map and I a downset of M. The cost function *recognised* by (M, h, I) is the equivalence class of the function that maps a word u to the maximal threshold n such that there exists an h-factorisation tree for u of threshold n, height at most $3|M|$ and root in I. Such an equivalence class of functions is called a *regular cost function*.

▶ **Example 2.5.** Consider the stabilisation monoid S_1 defined in Example 2.3 and let $I = \{0\}$. Let $h : \{a, b\} \to M$ be the labelling map defined by $h(a) = s$ and $h(b) = 1$. Then S_1 is a stabilisation monoid that can make a distinction between products with no s (that are 1), products containing "few" s (that are s) and products containing "a lot of" s (that are $0 = s^\sharp$). The cost function recognised by (S_1, h, I) is the equivalence class of the function $u \mapsto |u|_a$.

For instance the tree from example 2.3 of height 4 and threshold 5 has root labelled by $s^\sharp = 0$ because it is a witness that there are more than 5 occurrences of a in the input word. Conversely, such a factorisation tree of threshold n and height k would have its root labelled by 1 if the input word contains no a, and labelled by s if there are at most n^k occurrences of a. Because k is a fixed constant, these trees can be used to recognise the cost function $u \mapsto |u|_a$, since it is equivalent to $u \mapsto (|u|_a)^k$.

Regular cost functions can also be recognised by generalised forms of nondeterministic finite automata, regular expressions or monadic second-order logical formulas. See [11] for a complete introduction. Moreover, every regular cost function f has a unique *syntactic stabilisation monoid* M, in the sense that:

(1) there is a unique pair (h, I) where $h : A \to M$ is a labelling function and I a downset of M such that f is recognised by (M, h, I),

(2) for any (M', h', I') recognising f, there is a surjective morphism $\varphi : M \to M'$ such that $h = \varphi \circ h'$ and $I' = \varphi^{-1}(I)$.

3 Stabilisation Algebras

The goal of the present work is to study algebraic properties of stabilisation monoids and cost functions. In particular, we would like to define regular cost functions as particular subsets

of a free stabilisation monoid. However, since in a stabilisation monoid, the $\sharp$-operator is only defined on idempotents, the notion of a free stabilisation monoid cannot be defined directly and requires the introduction of a new algebraic structure, in which idempotents are directly defined in the signature of the algebra: stabilisation algebras.

Given a countable set of variables X, let $T(X)$ be the free term algebra of signature $\{\cdot, \omega, \sharp, 1\}$ over X. An *identity over* $T(X)$ is an equation of the form $s \leqslant t$, where s and t are terms of $T(X)$.

A finite stabilisation monoid M *satisfies the identity* $s \leqslant t$ if the equation holds for any instantiation of the variables by elements of M, where 1 is interpreted as the neutral element of M, ω is interpreted as the idempotent power in M, and $\sharp$ is replaced with $\omega\sharp$ (to guarantee that $\sharp$ is only applied to idempotents). A finite stabilisation monoid M *satisfies the identity* $s = t$ if it satisfies the identities $s \leqslant t$ and $t \leqslant s$.

We can now define the structure of a stabilisation algebra in the following way.

▶ **Definition 3.1.** A *stabilisation algebra* is an ordered algebra M with signature $\langle 1, \leqslant, \cdot, \omega, \sharp \rangle$ satisfying the following axioms:

- ($\mathbf{A}_1$) all identities that are satisfied by all finite stabilisation monoids,
- ($\mathbf{A}_2$) a description of the behaviour of ω on idempotent elements: $x^2 = x$ implies $x^\omega = x$,
- ($\mathbf{A}_3$) the three properties expressing that the order $\leqslant$ is compatible with the operations $\cdot, \omega, \sharp$: $x_1 \leqslant x_2$ and $y_1 \leqslant y_2$ imply $x_1 y_1 \leqslant x_2 y_2$, and $x \leqslant y$ implies $x^\omega \leqslant y^\omega$ and $x^\sharp \leqslant y^\sharp$.

In particular, ($\mathbf{A}_1$) implies that a stabilisation algebra is a monoid with neutral element 1. A *morphism* between two stabilisation algebras is a monoid morphism which is order-preserving, ω-preserving and $\sharp$-preserving. Let M and N be two stabilisation algebras. Then N is a *stabilisation subalgebra* of M if $N \subseteq M$, and N is a *quotient* of M if there exists a surjective morphism $M \to N$. The product of two stabilisation algebras M and N is defined on the set product $M \times N$, with operations defined componentwise.

Recall that in a finite monoid, every element x has a unique idempotent power, denoted x^ω. This fact allows one to identify finite stabilisation monoids and finite stabilisation algebras.

▶ **Proposition 3.2.** *Finite stabilisation algebras are in one-to-one correspondence with finite stabilisation monoids, by interpreting ω as the idempotent power.*

We now build a free stabilisation algebra $F(A)$ on each finite alphabet A. Recall that $T(A)$ is the set of terms of the free algebra of signature $\{\cdot, \omega, \sharp, 1\}$ over the alphabet A. Given s and t two elements of $T(A)$, we write $s \equiv t$ if and only if the identity $t = s$ holds in all finite stabilisation monoids. This defines an equivalence relation and we let $F(A)$ denote the set of $\equiv$-classes. Furthermore we let $\bar{t}$ denote the equivalence class of t in $F(A)$.

▶ **Proposition 3.3.** *$F(A)$ can be equipped with a structure of stabilisation algebra.*

This follows from a general result on ordered algebras mentioned without proof in [6].

A recent result [18] states that the equivalence of two $\sharp$-free terms of $T(A)$ is decidable. Actually, the result is more general and also covers the case of $\omega - 1$ powers. However, deciding the equivalence of arbitrary terms in $T(A)$ seems to still be an open problem.

The following theorem shows that $F(A)$ is a free object, by making explicit the corresponding universal property.

▶ **Theorem 3.4** (Universal Property). *For any stabilisation algebra M and any function $h : A \to M$, there exists a unique morphism of stabilisation algebra $\overline{h} : F(A) \to M$ extending h.*

▶ **Corollary 3.5.** *Every A-generated stabilisation algebra is a quotient of $F(A)$.*

4 Recognisability

We now define the notion of recognisable downsets in the free stabilisation algebra. We will later see how a regular cost function can be identified with a recognisable downset. This will allow us to generalise the classical notions of syntactic congruence and syntactic monoid. We identify terms $t \in T(A)$ and their class $\overline{t} \in F(A)$ for more readability.

Let I be a downset of $F(A)$, let M be a stabilisation algebra and let $h : F(A) \to M$ be a surjective morphism. We say that I is *recognised by* h if there exists a downset J of M such that $I = h^{-1}(J)$. A downset I of $F(A)$ is said to be *recognisable* if it is recognised by some morphism onto a finite stabilisation algebra.

Syntactic congruence and syntactic stabilisation algebra

A *context* on A is an element of $T(A \cup \{x\})$, where $x \notin A$. In other words, a context is a term $T(A)$ with possible occurrences of the free variable x. Given a context C on A and an element t of $T(A)$, we let $C[t]$ denote the element of $T(A)$ obtained by replacing all the occurrences of x by t in C, i.e., $C[t] = C[x \leftarrow t]$. Let $\mathrm{Ctx}(A)$ denote the set of contexts on A.

Given a downset I of $F(A)$ and two elements t and s of $F(A)$, we write that $s \sim_I t$ if for any context C, $C[s] \in I$ is equivalent to $C[t] \in I$. This equivalence relation $\sim_I$ is a congruence on $F(A)$, called the *syntactic congruence* of I and the the quotient algebra $F(A)/\sim_I$ is the *syntactic stabilisation algebra* of I. The quotient morphism $h : F(A) \to F(A)/\sim_I$ is the *syntactic morphism* of I and the triple $(F(A)/\sim_I, h, h(I))$ is called the *syntactic triple* of I.

Given a recognisable downset I of M, there is a natural preorder among the morphisms recognising I: given $h_1 : M \to N_1$ and $h_2 : M \to N_2$, we set $h_1 \leqslant h_2$ if there exists a surjective morphism $h : N_1 \to N_2$ such that $h_2 = h \circ h_1$. Then we can state:

▶ **Proposition 4.1.** *The syntactic morphism is a minimal element for this preorder.*

The analog of this property in the framework of stabilisation monoids is given in [16].

Regular Cost Functions Versus Recognisable Downsets

We have seen that regular cost functions are recognised by finite stabilisation monoids and that recognisable downsets are recognised by finite stabilisation algebras. Now, Proposition 3.2 shows that finite stabilisation algebras correspond exactly to finite stabilisation monoids. These results indicate that regular cost functions and recognisable downsets are closely related.

One can make this relation a bijection as follows. Let f be a regular cost function and let M be its syntactic stabilisation monoid. Let also (h, I) be the unique pair (where $h : A \to M$ is a labelling function and I is a downset of M) such that f is recognised by (M, h, I). Then one can view M as a stabilisation algebra and extend h to a morphism $h : F(A) \to M$. Then the recognisable downset associated with f is $h^{-1}(I) \subseteq F(A)$.

Conversely, for every recognisable downset $I \subseteq F(A)$, one can define the regular cost function f associated to I by considering the syntactic triple (M, h, J) of I, and see M as a stabilisation monoid.

It is interesting to consider the link with regular languages. Let L be a regular language and let $\sim_L$ be its syntactic congruence. Let I_L be the downset corresponding to the regular cost function χ_L, i.e., the downset representing the language L. Since downsets represents cost functions via their unbounded elements, I_L is actually the set of elements of $T(A)$ representing words **not** in L. Any element of $T(A)$ can be tested for membership in L by evaluating it in the finite stabilisation algebra $M = A^*/\sim_L$ where $\sharp = id$ and ω is the idempotent power. Remark now that for all $u, v \in A^*$, we have $u \sim_L v$ if and only if $u \sim_{I_L} v$ (where u and v are interpreted in $T(A)$). In this sense, $\sim_I$ is an extension of the classical syntactic congruence on languages.

▶ **Example 4.2.** Let $A = \{a, b\}$. Consider the downset of $T(A)$ which consists of all terms containing an occurrence of a under the scope of $\sharp$. This set describes words containing a large number of a's. It is of finite index, and it represents the cost function g given in Example 2.1 that counts the number of a's in a word. It is also recognised by the stabilisation monoid given in Example 2.3 with elements $1 \geqslant a \geqslant 0$, all idempotent, and with $a^\sharp = 0^\sharp = 0$.

▶ **Proposition 4.3.** *The lattice of regular cost functions under* min *and* max *is isomorphic to the lattice of recognisable downsets under union and intersection.*

5 Varieties

We now generalise the notion of varieties of regular languages and some proofs from [13, 19].

A *lattice of recognisable downsets* is a set of recognisable downsets containing $\emptyset$ and $F(A)$, and closed under finite union and finite intersection. A lattice $\mathcal{L}$ of regular downsets of $F(A)$ is *closed under contexts* if, for every $I \in F(A)$ and each context $C \in \mathrm{Ctx}(A)$, the condition $I \in \mathcal{L}$ implies $C^{-1}[I] \in \mathcal{L}$, where $C^{-1}[I] = \{t \in F(A) \mid C[t] \in I\}$.

A *variety of recognisable downsets* associates with each finite alphabet A a lattice $\mathcal{V}(A)$ of recognisable downsets of $F(A)$ satisfying the following properties:

($\mathbf{V}_1$) For each alphabet A, $\mathcal{V}(A)$ is closed under contexts.

($\mathbf{V}_2$) For each morphism $\varphi : F(B) \to F(A)$, the condition $I \in \mathcal{V}(A)$ implies $\varphi^{-1}(I) \in \mathcal{V}(B)$.

Varieties of downsets generalise positive varieties of languages [19], as there is no complementation for downsets.

▶ **Example 5.1.** A recognisable downset I is *aperiodic* if for all $t \in F(A)$, the relation $t^\omega \sim_I t^\omega t$ holds. It is not too difficult to show that aperiodic downsets form a variety of recognisable downsets.

We now define varieties of stabilisation algebras.

▶ **Definition 5.2.** A *variety of finite stabilisation algebras* is a class of finite stabilisation algebras closed under taking stabilisation subalgebras, quotients and finite products.

Notice that this notion is often called *pseudovarieties* in the literature, as opposed to Birkhoff varieties which are also closed under arbitrary products.

▶ **Example 5.3.** A finite stabilisation algebra M is aperiodic if for all $x \in M$, we have $x^\omega = x^\omega x$. Aperiodic stabilisation algebras form a variety of finite stabilisation algebras.

Let M, N be two stabilisation algebras. We say that M *divides* N if M is a quotient of a stabilisation subalgebra of N. It follows from the definition that varieties of stabilisation algebras are closed under division. If S is a (possibly infinite) set of finite stabilisation algebras, the *variety generated by* S is the smallest variety of finite stabilisation algebras containing the elements of S.

▶ **Lemma 5.4.** *Let* $\mathbf{V}$ *be a variety generated by a set* S *of finite stabilisation algebras, and* M *be a finite stabilisation algebra. Then* $M \in \mathbf{V}$ *if and only if* M *divides a finite product of elements of* S*.*

Given a variety $\mathbf{V}$ of finite stabilisation algebras, let $\mathcal{V}(A)$ denote the set of recognisable downsets over A whose syntactic stabilisation algebra belongs to $\mathbf{V}$. The correspondence $\mathbf{V} \to \mathcal{V}$ associates with each variety of finite stabilisation algebras a class of recognisable downsets.

Thus, each variety of recognisable downsets $\mathcal{V}$ is associated to the variety of finite stabilisation algebras $\mathbf{V}$ generated by the syntactic stabilisation algebras of downsets in $\mathcal{V}$. This defines a correspondence $\mathcal{V} \to \mathbf{V}$. The analog of the ordered version of Eilenberg's theorem can now be stated as follows:

▶ **Theorem 5.5.** *The correspondences* $\mathcal{V} \to \mathbf{V}$ *and* $\mathbf{V} \to \mathcal{V}$ *define mutually inverse bijective correspondences between varieties of finite stabilisation algebras and varieties of recognisable downsets.*

6 Profinite Stabilisation Algebra

The free profinite monoid on A, denoted $\widehat{A^*}$, can be defined as the completion of A^* for the *profinite metric*. See [4, 20, 21] for more information on this space.

We now prove the existence of free profinite stabilisation algebras. Taking the construction of free profinite monoids as a model, we define it as the completion $\widehat{F(A)}$ of $F(A)$ for an appropriate metric.

▶ **Definition 6.1.** A stabilisation algebra M *separates* two elements s and t of $F(A)$ if there is a morphism $\varphi : F(A) \to M$ such that $\varphi(s) \neq \varphi(t)$. For $s, t \in F(A)$, define

$$d(s,t) = \begin{cases} 0 & \text{if } s = t \\ 2^{-n(s,t)} & \text{otherwise} \end{cases}$$

where $n(s,t)$ is the minimum size of a finite stabilisation algebra separating s and t.

Note that d is well defined, since if $s \neq t \in F(A)$, then there is by $(\mathbf{A}_1)$ a finite stabilisation monoid in which the identity $s = t$ fails. Such a monoid can be viewed as a finite stabilisation algebra separating s and t. The following proposition gathers the properties of d and $\widehat{F(A)}$:

▶ **Proposition 6.2.**

(1) *d is an ultrametric distance.*

(2) *The operations on $F(A)$ are uniformly continuous and thus extend by continuity to $\widehat{F(A)}$.*

(3) *The resulting stabilisation algebra $\widehat{F(A)}$ is compact.*

The idempotent power. If M is a finite monoid, then for any $m \in M$ and $n \geqslant |M|$, we have $m^\omega = m^{n!}$ (where ω is the idempotent power). Since finite stabilisation algebras are in particular monoids where ω is the idempotent power, we obtain that for any $u \in F(A)$ and $n > 0$, $d(u^{n!}, u^\omega) \leqslant 2^{-n}$. Therefore, for any element $u \in F(A)$, the sequence $(u^{n!})_{n \in \mathbb{N}}$ converges in $\widehat{F(A)}$, to u^ω.

▶ **Proposition 6.3.** *The morphism $\varphi : F(A) \to \widehat{A^*}$ defined by $\varphi(a) = a$, $\varphi(t^\omega) = t^\omega$ and $\varphi(t^\sharp) = t^\omega$ is uniformly continuous, and therefore can be uniquely extended into a continuous morphism of stabilisation algebras $\widehat{\varphi} : \widehat{F(A)} \to \widehat{A^*}$.*

Notice however that this morphism does not coincide with the interpretation of regular cost functions as subsets of $\widehat{A^*}$ as done in [30].

The profinite metric can be relativised to any variety of stabilisation algebras to obtain the so-called pro-$\mathbf{V}$ metric. For $s, t \in F(A)$ and $\mathbf{V}$ a variety of stabilisation algebras, define $d_{\mathbf{V}}(s,t) = 2^{-|M|}$ where M is one of the smallest stabilisation algebras from $\mathbf{V}$ separating s and t and $d_{\mathbf{V}}(s,t) = 0$ if there is no such M. Remark that the metric d of Definition 6.1 corresponds to $d_{\mathbf{V}}$ where $\mathbf{V}$ is the variety of all finite stabilisation algebras. We also define an equivalence $s \sim_{\mathbf{V}} t$ by $d_{\mathbf{V}}(s,t) = 0$.

▶ **Proposition 6.4.** *For any variety $\mathbf{V}$, $\sim_{\mathbf{V}}$ is a congruence on $F(A)$ and $d_{\mathbf{V}}$ is an ultrametric distance on $F(A)/\sim_{\mathbf{V}}$.*

We now define the pro-$\mathbf{V}$ stabilisation algebra $\widehat{F_{\mathbf{V}}(A)}$ as the completion of $F(A)/\sim_{\mathbf{V}}$ with respect to $d_{\mathbf{V}}$. As before, we can show that $\widehat{F_{\mathbf{V}}(A)}$ is compact and can be equipped with a structure of stabilisation algebra. The following result now follows from general results on profinite algebras.

▶ **Theorem 6.5.** *A finite A-generated stabilisation algebra belongs to $\mathbf{V}$ if and only if it is a continuous quotient of $\widehat{F_{\mathbf{V}}(A)}$.*

7 Duality, Equations and Identities

Stone duality tells us that every bounded distributive lattice $\mathcal{L}$ has an associated compact Hausdorff space, called its *dual space*. The dual space of the Boolean algebra of all *regular* languages of A^* [3] is the free profinite monoid on A.

A similar result holds for the lattice of regular cost functions, which, by Proposition 4.3, is isomorphic to the lattice of recognisable downsets under union and intersection.

▶ **Theorem 7.1.** *The dual space of the lattice of recognisable downsets of $F(A)$ is the space $\widehat{F(A)}$.*

7.1 Equations of Lattices

It is shown in [14] that any lattice of regular languages can be defined by a set of equations of the form $u \to v$, where u and v are profinite words. This result can also be extended to recognisable downsets.

Let $u, v \in \widehat{F(A)}$. We say that a recognisable downset I of $F(A)$ *satisfies the equation* $u \to v$ if $u \in \overline{I}$ implies $v \in \overline{I}$, where $\overline{I}$ denotes the topological closure of I.

A set $\mathcal{L}$ of recognisable downsets is *defined by a set E of equations* if the following property holds: a recognisable downset belongs to $\mathcal{L}$ if and only if it satisfies all the equations of E. We can now state our second main result.

▶ **Theorem 7.2.** *A set of recognisable downsets of $F(A)$ is a lattice of recognisable downsets if and only if it is defined by a set of equations of the form $u \to v$.*

The case of lattices of languages closed under quotients was also considered in [14]. The corresponding notion for lattices of downsets is to be closed under contexts.

A *profinite context* C on the finite alphabet A is an element of $F(\widehat{A \cup \{x\}})$ where $x \notin A$. If u is an element of $\widehat{F(A)}$, then $C[u]$ is also an element of $\widehat{F(A)}$, defined by replacing x by u and evaluating the operations ω and $\sharp$ in the stabilisation algebra $\widehat{F(A)}$.

▶ **Definition 7.3.** A recognisable downset of $F(A)$ *satisfies the equation* $u \leqslant v$ if, for all profinite contexts C, it satisfies the equation $C[u] \to C[v]$.

Equivalently, a stabilisation algebra *satisfies the equation* $u \leqslant v$ if, for all downsets J of M and for all contexts C, $C[v] \in J$ implies $C[u] \in J$. By density of $F(A)$ in $\widehat{F(A)}$, it is enough to consider contexts in Ctx_A for this definition. The notation $u = v$ is used as a shortcut for $u \leqslant v$ and $v \leqslant u$. We can now state:

▶ **Theorem 7.4.** *A set of recognisable downsets of $F(A)$ is a lattice of recognisable downsets closed under contexts if and only if it is defined by a set of equations of the form $u \leqslant v$.*

7.2 Identities of Varieties

Condition ($\mathbf{V}_2$) of the definition of a variety allows one to use identities instead of equations.

Let B be an alphabet and let u and v be two elements of $\widehat{F(B)}$. We say that a recognisable downset I of $F(A)$ *satisfies the profinite identity* $u \leqslant v$ if, for each morphism $\gamma : F(B) \to F(A)$, I satisfies the equation $\widehat{\gamma}(u) \leqslant \widehat{\gamma}(v)$.

We use the term *identity* because, in this case, each letter of B can be replaced (through the morphism γ) by *any* element of $F(A)$.

In practice, it is more convenient to use the following characterisation. Let I be a recognisable downset and let M be its syntactic stabilisation algebra. Then I satisfies the identity $u \leqslant v$ if and only if for every continuous morphism $h : F(B) \to M$, one has $\widehat{h}(u) \leqslant_M \widehat{h}(v)$, where $\leqslant_M$ is the order of M.

▶ **Theorem 7.5.** *A class of finite stabilisation algebras (resp., recognisable downsets) is a variety if and only if it is defined by a set of identities of the form $u \leqslant v$.*

8 Examples of Equational Descriptions of Varieties and Lattices

In this section, we gather examples of varieties of regular cost functions and of sets of equations. First, it is interesting to see how the identification of regular languages with cost functions extends to varieties.

▶ **Proposition 8.1.** *If a positive variety of regular languages is defined by a set of identities E, then it is a variety of regular cost functions, defined by the set of identities $E \cup \{x^\sharp = x^\omega\}$. Conversely, if a variety of regular cost functions is defined by a set of identities E, then the variety of regular cost functions defined by $E \cup \{x^\sharp = x^\omega\}$ can be identified with a positive variety of languages.*

For instance, the variety of regular cost functions defined by $x^\omega = x^{\omega+1}$ and $x^\sharp = x^\omega$ contains only the characteristic functions of star-free languages [28].

Aperiodic Cost Functions

The variety of *aperiodic cost functions* is defined by the identity $x^\omega = x^{\omega+1}$. It contains recognisable downsets that are not languages, like $u \mapsto |u|_a$. This variety has a nice connection with the logics CFO and CLTL, first introduced in [16, 17] as a generalisation to cost functions of the logics FO and LTL on words. Indeed, the results of [16, 17] can be reformulated as follows:

▶ **Theorem 8.2.** *The variety of aperiodic cost functions coincides with the variety of CFO-definable cost functions and with the variety of CLTL-definable cost functions.*

Note that given a finite stabilisation algebra M, one can effectively test whether it verifies equations like $x^\omega = x^{\omega+1}$ or $x^\sharp = x^\omega$: it suffices to check that it stands for each x in M. It follows that one can effectively decide whether a regular cost function is CFO-definable (respectively CLTL-definable).

Temporal Cost Functions

Another interesting example is the class of temporal cost functions, first introduced in [11]. These functions allow one to count the number of occurrences of consecutive events. Many equivalent characterizations of these functions are known. In [11], the algebraic characterization is expressed in terms of the interplay between Green relations and stabilisation in the syntactic monoid, but it can be formulated in terms of equations as follows:

▶ **Theorem 8.3.** *Temporal cost functions over A form a lattice of regular cost functions, defined by the equations $(xy^\sharp z)^\sharp = (xy^\sharp z)^\omega$, for all $x, z \in F(A)$ and all $y \in F(A) - \{1\}$.*

Proof. Let M be the syntactic stabilisation monoid of a regular cost function f. An idempotent e is called *stable* if $e^\sharp = e$. The algebraic characterization from [11] states that f is temporal if and only if an idempotent $\mathcal{J}$-below a stable idempotent different from 1 is itself stable. Recall that the $\mathcal{J}$-order is defined by $e \leqslant_\mathcal{J} s$ if there exist $x, y \in M$ such that $e = xsy$. To show that our set of equations is equivalent to this characterization, it suffices to observe that an element is a stable idempotent if and only if it is of the form $s^\sharp$ for some s. This means that the characterization from [11] specifies that the idempotents of the form $(xs^\sharp z)^\omega$, with $s \neq 1$, are stable. Using Corollary 3.5, one can now lift these properties to $F(A)$, yielding the equations of the statement. ◀

Commutative Cost Functions

The description of the variety of languages corresponding to commutative monoids is one of the first known examples of Eilenberg's correspondence between varieties of languages and varieties of monoids [13]. We prove below a similar result for cost functions.

Let us say that a finite stabilisation algebra M is *commutative* if for all $x, y \in M$, we have $xy = yx$. We will say that M is $\sharp$-*commutative* if it is commutative and for all $x, y \in M$, $x^\sharp y^\sharp = (xy)^\sharp$. A cost function is called *commutative* (resp., $\sharp$-*commutative*) if its syntactic stabilisation algebra is commutative (resp., $\sharp$-commutative).

▶ **Example 8.4.** The cost function $\mathrm{maxblock}_a$ on the alphabet $\{a, b\}$ defined by:

$$\mathrm{maxblock}_a(a^{n_1}ba^{n_2}b\cdots ba^{n_k}) = \max(n_1, n_2, \ldots, n_k).$$

is commutative but not $\sharp$-commutative. The downset representing this regular cost function consists in $\omega\sharp$-expressions on $\{a, b\}$ containing a subexpression of the form $u^\sharp$, where u is an $\omega\sharp$-expression on the alphabet $\{a\}$ with at least one occurrence of a. Its syntactic stabilisation algebra M has four elements: $0 \leqslant a \leqslant 1 \leqslant b$, all elements are idempotent and commute, and we have $a^\sharp = 0$, $x^\sharp = x$ for $x \neq a$, and $ab = b = ba$. It is not $\sharp$-commutative because $a^\sharp b^\sharp = 0b = 0$ and $(ab)^\sharp = b^\sharp = b$.

A stabilisation algebra is said to be *monogenic* if it can be generated by a single element. We will use freely the following useful lemma.

▶ **Lemma 8.5.** *Any finite $\sharp$-commutative stabilisation algebra divides the product of its monogenic stabilisation subalgebras.*

The stabilisation monoid S_1 defined in Example 2.3 has three idempotent elements $0 < a < 1$ such that $0^\sharp = a^\sharp = 0$. It is also the syntactic stabilisation algebra of the function $f = u \mapsto |u|_a$. Let U_1^+ denote the stabilisation monoid with two idempotent elements $0 \leqslant 1$ such that $0^\sharp = 0$.

▶ **Proposition 8.6.** *Let $\mathbf{J_1^+}$ be the variety of finite stabilisation algebras defined by the equations $x \leqslant 1$, $x^2 = x$, $xy = yx$ and $x^\sharp y^\sharp = (xy)^\sharp$. Then the corresponding variety of cost functions is generated by the functions $u \mapsto |u|_a$ for all letters a.*

Proof. Since S_1 is the syntactic stabilisation algebra of the function $u \mapsto |u|_a$, it is equivalent to prove that the variety of finite stabilisation algebras $\mathbf{V}$ generated by S_1 is equal to $\mathbf{J_1^+}$. Since S_1 satisfies all the equations of $\mathbf{J_1^+}$, the relation $\mathbf{V} \subseteq \mathbf{J_1^+}$ holds. To prove the opposite inclusion, consider a finite stabilisation algebra M of $\mathbf{J_1^+}$. By Lemma 8.5, M divides the product of its monogenic stabilisation subalgebras. But if $m \in M$, the stabilisation algebra generated by m is $\{1, m, m^\sharp\}$: indeed, the equations $m^2 = m$, $(m^\sharp)^2 = m^\sharp$ and the properties of a stabilisation monoid imply that $mm^\sharp = m^\sharp m = m^\sharp$. Thus, this stabilisation algebra is either $\{1\}, U_1^+$ or S_1. Since U_1^+ is a quotient of S_1, M actually divides a product of copies of S_1, and therefore $M \in \mathbf{V}$. Thus $\mathbf{V} = \mathbf{J_1^+}$. ◀

As stated earlier, if we add the equation $x^\sharp = x^\omega$, we obtain the positive variety of regular languages corresponding to the variety of ordered monoids generated by the ordered monoid U_1^+ (see [19]).

▶ **Proposition 8.7.** *Let* **Acom** *be the variety of finite stabilisation algebras defined by the equations $x^\omega = x^{\omega+1}$, $xy = yx$ and $x^\sharp y^\sharp = (xy)^\sharp$. Then the corresponding variety of cost functions is generated by the functions $u \mapsto |u|_a$ and $\chi_{L_{a,k}}$ where $L_{a,k} = \{u \mid |u|_a = k\}$ for each $k \geqslant 0$ and each letter a.*

▶ **Proposition 8.8.** *Let* **Com** *be the variety of finite stabilisation algebras defined by the equations $xy = yx$ and $x^\sharp y^\sharp = (xy)^\sharp$. Then the corresponding variety of cost functions is generated by the functions $u \mapsto |u|_a$, $\chi_{L_{a,k}}$ and $\chi_{L_{k,n}}$, where $L_{a,k,n} = \{u \mid |u|_a \equiv k \bmod n\}$.*

9 Conclusion

We provide a new representation of regular cost functions as downsets of a free stabilisation algebra, an ordered algebraic structure. This new representation allows us to extend Eilenberg's variety theory, in its ordered version: varieties of regular cost functions correspond to varieties of finite stabilisation algebras and are characterised by profinite identities. Furthermore, we also extend the duality approach of [14] to this new setting, leading to profinite equational descriptions of lattices of regular cost functions. Finally, we give several examples of equational characterisations of classes of cost functions related to logic. We also investigate the extensions of commutative languages to regular cost functions. We uncover the role of a new identity, $x^\sharp y^\sharp = (xy)^\sharp$, in the study of these extensions.

These results confirm the pertinence and the usefulness of the theory of regular cost functions as a well-behaved quantitative generalisation of regular languages. They also open new perspective for the study of cost functions.

For instance, it would be interesting to extend other known characterisations of varieties of languages to the setting of cost functions. An emblematic example would be Simon's characterisation of piecewise testable languages.

Acknowledgments . The authors would like to thank Thomas Colcombet, Nathanaël Fijalkow and Sam Van Gool for many helpful discussions.

References

1 Jirí Adámek, Stefan Milius, Robert S. R. Myers, and Henning Urbat. Generalized eilenberg theorem I: local varieties of languages. In Anca Muscholl, editor, *FOSSACS 2014*, volume 8412 of *Lecture Notes in Comput. Sci.*, pages 366–380. Springer, 2014.

2 Jirí Adámek, Robert S. R. Myers, Henning Urbat, and Stefan Milius. Varieties of languages in a category. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 414–425. IEEE, 2015.

3 Jorge Almeida. Residually finite congruences and quasi-regular subsets in uniform algebras. *Portugaliæ Mathematica*, 46:313–328, 1989.

4 Jorge Almeida. *Finite semigroups and universal algebra*. World Scientific Publishing Co. Inc., River Edge, NJ, 1994. Translated from the 1992 Portuguese original and revised by the author.

5 Jean Berstel and Christophe Reutenauer. *Noncommutative rational series with applications*, volume 137 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 2011.

6 Stephen L. Bloom. Varieties of ordered algebras. *J. Comput. System Sci.*, 13(2):200–212, 1976.

7 Mikolaj Bojanczyk. Recognisable languages over monads. In Igor Potapov, editor, *Developments in Language Theory – 19th International Conference, DLT 2015*, volume 9168 of *Lecture Notes in Comput. Sci.*, pages 1–13. Springer, 2015.

8 Mikolaj Bojanczyk. Recognisable languages over monads. *CoRR*, abs/1502.04898, 2015.

9 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, languages and programming. Part II*, volume 5556 of *Lecture Notes in Comput. Sci.*, pages 139–150, Berlin, 2009. Springer.

10 Thomas Colcombet. Regular cost functions, Part I: Logic and algebra over words. *Log. Methods Comput. Sci.*, 9(3):3:3, 47, 2013.

11 Thomas Colcombet, Denis Kuperberg, and Sylvain Lombardy. Regular temporal cost functions. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, languages and programming, ICALP 2010, Part II*, volume 6199 of *Lecture Notes in Comput. Sci.*, pages 563–574. Springer, Berlin, 2010.

12 Manfred Droste, Werner Kuich, and Heiko Vogler, editors. *Handbook of weighted automata*. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2009.

13 Samuel Eilenberg. *Automata, languages, and machines. Vol. B*. Academic Press [Harcourt Brace Jovanovich Publishers], New York, 1976.

14 Mai Gehrke, Serge Grigorieff, and Jean-Éric Pin. Duality and equational theory of regular languages. In L. Aceto and al., editors, *ICALP 2008, Part II*, volume 5126 of *Lecture Notes in Comput. Sci.*, pages 246–257, Berlin, 2008. Springer.

15 Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *Internat. J. Algebra Comput.*, 4(3):405–425, 1994.

16 Denis Kuperberg. Linear temporal logic for regular cost functions. In *28th International Symposium on Theoretical Aspects of Computer Science*, volume 9 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages 627–636. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2011.

17 Denis Kuperberg and Michael Vanden Boom. On the expressive power of cost logics over infinite words. In *Automata, languages, and programming. Part II*, volume 7392 of *Lecture Notes in Comput. Sci.*, pages 287–298. Springer, Heidelberg, 2012.

18 I. Mikhailova. A proof of Zhil'tsov's theorem on decidability of equational theory of epigroups, 2013. URL: http://arxiv.org/abs/1310.5023.

19 Jean-Éric Pin. A variety theorem without complementation. *Russian Mathematics (Izvestija vuzov.Matematika)*, 39:80–90, 1995.

20 Jean-Éric Pin. Profinite methods in automata theory. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009)*, pages 31–50. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2009.

21 Jean-Éric Pin. Equational descriptions of languages. *Int. J. Found. Comput. S.*, 23:1227–1240, 2012.

22 Christophe Reutenauer. Variétés d'algèbres et de séries rationnelles. In *1er Congrès Math. Appl. AFCET-SMF*, volume 2, pages 93–102. AFCET, 1978.

23 Christophe Reutenauer. Séries formelles et algèbres syntactiques. *J. Algebra*, 66(2):448–483, 1980.

24 Christophe Reutenauer. Séries rationnelles et algèbres syntactiques. Thèse de Doctorat d'État, Université Paris 6, 1980.

25 Marcel-Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.

26 Marcel-Paul Schützenberger. Finite counting automata. *Information and Control*, 5:91–107, 1962.

27 Marcel-Paul Schützenberger. On a theorem of R. Jungen. *Proc. Amer. Math. Soc.*, 13:885–890, 1962.

28 Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

29 Imre Simon. Factorization forests of finite height. *Theoret. Comput. Sci.*, 72(1):65–94, 1990.

30 Szymon Toruńczyk. *Languages of profinite words and the limitedness problem.* Phd thesis, Computer Science Department, University of Warsaw, 2011.

Kernelization and Sparseness: the Case of Dominating Set*

Pål Grønås Drange[1], Markus Dregi[2], Fedor V. Fomin[3], Stephan Kreutzer[4], Daniel Lokshtanov[5], Marcin Pilipczuk[6], Michał Pilipczuk[7], Felix Reidl[8], Fernando Sánchez Villaamil[9], Saket Saurabh[10], Sebastian Siebertz[11], and Somnath Sikdar[12]

1 **Department of Informatics, University of Bergen, Norway**
pal.drange@ii.uib.no
2 **Department of Informatics, University of Bergen, Norway**
markus.dregi@ii.uib.no
3 **Department of Informatics, University of Bergen, Norway**
fomin@ii.uib.no
4 **Institute of Software Technology and Theoretical Computer Science, Technische Universität Berlin, Germany**
stephan.kreutzer@tu-berlin.de
5 **Department of Informatics, University of Bergen, Norway**
daniello@ii.uib.no
6 **Institute of Informatics, University of Warsaw, Poland**
marcin.pilipczuk@mimuw.edu.pl
7 **Institute of Informatics, University of Warsaw, Poland**
michal.pilipczuk@mimuw.edu.pl
8 **Theoretical Computer Science, Department of Computer Science, RWTH Aachen University, Germany**
reidl@cs.rwth-aachen.de
9 **Theoretical Computer Science, Department of Computer Science, RWTH Aachen University, Germany**
fernando.sanchez@cs.rwth-aachen.de
10 **Department of Informatics, University of Bergen, Norway: and Institute of Mathematical Sciences, India**
saket.saurabh@ii.uib.no, saket@imsc.res.in
11 **Institute of Software Technology and Theoretical Computer Science, Technische Universität Berlin, Germany**
sebastian.siebertz@tu-berlin.de
12 **Theoretical Computer Science, Department of Computer Science, RWTH Aachen University, Germany**
sikdar@cs.rwth-aachen.de

Abstract

We prove that for every positive integer r and for every graph class $\mathcal{G}$ of bounded expansion, the r-Dominating Set problem admits a linear kernel on graphs from $\mathcal{G}$. Moreover, in the

* The research of P. Drange, F. Fomin, M. Pilipczuk, and M. Pilipczuk leading to these results has received funding from the European Research Council (ERC) under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 267959. Mi. Pilipczuk currently holds a post-doc position at Warsaw Centre of Mathematics and Computer Science, and is supported by the Foundation for Polish Science. F. Reidl, F. Villaamil, and S. Sikdar are supported by DFG-Project RO 927/13-1 "Pragmatic Parameterized Algorithms". M. Dregi and D. Lokshtanov are supported by the BeHard grant under the recruitment programme of the of Bergen Research Foundation. S. Saurabh is supported by PARAPPROX, ERC starting grant no. 306992. S. Kreutzer has been supported by the ERC under the EU's Horizon 2020 research and innovation programme (grant agreement No 648527).

© Pål Grønås Drange, Markus Dregi, Fedor V. Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, Felix Reidl, Fernando Sánchez Villaamil, Saket Saurabh, Sebastian Siebertz, and Somnath Sikdar
licensed under Creative Commons License CC-BY

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 31; pp. 31:1–31:14

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

more general case when $\mathcal{G}$ is only assumed to be nowhere dense, we give an almost linear kernel on $\mathcal{G}$ for the classic Dominating Set problem, i.e., for the case $r = 1$. These results generalize a line of previous research on finding linear kernels for Dominating Set and r-Dominating Set (Alber et al., JACM 2004, Bodlaender et al., FOCS 2009, Fomin et al., SODA 2010, Fomin et al., SODA 2012, Fomin et al., STACS 2013). However, the approach taken in this work, which is based on the theory of sparse graphs, is radically different and conceptually much simpler than the previous approaches.

We complement our findings by showing that for the closely related Connected Dominating Set problem, the existence of such kernelization algorithms is unlikely, even though the problem is known to admit a linear kernel on H-topological-minor-free graphs (Fomin et al., STACS 2013). Also, we prove that for any somewhere dense class $\mathcal{G}$, there is some r for which r-Dominating Set is W[2]-hard on $\mathcal{G}$. Thus, our results fall short of proving a sharp dichotomy for the parameterized complexity of r-Dominating Set on subgraph-monotone graph classes: we conjecture that the border of tractability lies exactly between nowhere dense and somewhere dense graph classes.

1998 ACM Subject Classification F.2.2 Nonnumerical algorithms and problems

Keywords and phrases kernelization, dominating set, bounded expansion, nowhere dense

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.31

1 Introduction

In the classic Dominating Set problem, given a graph G and an integer k, we are asked to determine the existence of a subset $D \subseteq V(G)$ of size at most k such that every vertex $u \in V(G)$ is *dominated* by D: either u belongs to D itself, or it has a neighbor that belongs to D. The r-Dominating Set problem, for a positive integer r, is a generalization where each vertex of D dominates all the vertices at distance at most r from it. Dominating Set parameterized by the target size k plays a central role in parameterized complexity as it is a predominant example of a W[2]-complete problem. Recall that the main focus in parameterized complexity is on designing *fixed-parameter* algorithms, or shortly *FPT* algorithms, whose running time on an instance of size n and parameter k has to be bounded by $f(k) \cdot n^c$ for some computable function f and constant c. Downey and Fellows introduced a hierarchy of parameterized complexity classes FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \cdots$ that is believed to be strict, see [8, 13]. As Dominating Set is W[2]-complete in general, we do not expect it to be solvable in FPT time.

However, it turns out that various restrictions on the input graph lead to robust tractability of Dominating Set. Out of these, a particularly fruitful line of research concerned investigation of the complexity of the problem in sparse graph classes, like planar graphs, graphs of bounded genus, or graphs excluding some fixed graph H as a minor. In these classes we can even go one step further than just showing fixed-parameter tractability: It is possible to design a linear kernel for the problem. Formally, a *kernelization algorithm* (or a *kernel*) is a polynomial-time preprocessing procedure that given an instance (I, k) of a parameterized problem outputs another instance (I', k') of the same problem which is equivalent to (I, k), but whose total size $|I'| + k'$ is bounded by $f(k)$ for some computable function f, called the *size* of the kernel. If f is polynomial (resp. linear), then such an algorithm is called a *polynomial* (resp. *linear*) *kernel*.

The quest for small kernels for Dominating Set on sparse graph classes began with the groundbreaking work of Alber et al. [1], who showed the first linear kernel for the problem

on planar graphs. Another important step was the work of Alon and Gutner [2, 21], who gave an $\mathcal{O}(k^c)$ kernel for the problem on H-topological-minor free graphs, where c depends on H only. Moreover, if $H = K_{3,h}$ for some h, then the size of the kernel is actually linear. This led Alon and Gutner to pose the following excellent question: Can one characterize the families of graphs where Dominating Set admits a linear kernel?

The research program sketched by the works of Alber et al. [1] and Alon and Gutner [2, 21] turned out to be one of particularly fruitful directions in parameterized complexity in recent years, and eventually led to the discovery of new and deep techniques. In particular, linear kernels for Dominating Set have been given for bounded genus graphs [3], apex-minor-free graphs [14], H-minor-free graphs [15], and most recently H-topological-minor-free graphs [16]. In all these results, the notion of *bidimensionality* plays the central role. Using variants of the Grid Minor Theorem, it is possible to understand well the connections between the minimum possible size of a dominating set in a graph and its treewidth. The considered graph classes also admit powerful decomposition theorems that follow from the Graph Minors project of Robertson and Seymour [26], or the recent work of Grohe and Marx [20] on excluding H as a topological minor. The combination of these tools provides a robust base for a structural analysis of the input instance.

Beyond the current frontier of H-topological-minor-free graphs [16], kernelization of Dominating Set was studied in graphs of bounded degeneracy. Recall that a graph is called *d-degenerate* if every subgraph contains a vertex of degree at most d. Philip et al. [24] obtained a kernel of size $\mathcal{O}(k^{(d+1)^2})$ on d-degenerate graphs for constant d. However, as proved by Cygan et al. [5], the exponent of the size of the kernel needs to increase with d at least quadratically (unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$).

As far as r-Dominating Set is concerned, the current most general result gives a linear kernel for any apex-minor-free class [14], and follows from a general protrusion machinery [4]. The techniques used by Fomin et al. [15, 16] for H-(topological)-minor-free classes are tailored to the classic Dominating Set problem, and do not carry over to an arbitrary radius r. Therefore, up to this point the existence of linear kernels for r-Dominating Set on H-(topological)-minor-free classes was open.

Sparsity. The general concept of sparsity, beyond graphs with excluded (topological) minors, has been recently the subject of intensive study both from the point of view of pure graph theory and of computer science. In particular, the notions of graph classes of *bounded expansion* and *nowhere dense* graph classes have been introduced by Nešetřil and Ossona de Mendez. The main idea behind these models is to establish an abstract notion of sparsity based on known properties of well-studied sparse graph classes, e.g. H-minor-free graphs, and to develop tools for combinatorial analysis of sparse graphs based only on this abstract notion. We refer to the book of Nešetřil and Ossona de Mendez [23] for an introduction to the topic.

To measure sparsity of a graph more formally, we need a few definitions. We say that a graph H is a *minor* of G if there exists a family $(X_v)_{v \in V(H)}$ of pairwise disjoint subsets of $V(G)$ such that for every $v \in V(H)$ the graph $G[X_v]$ is connected and for every $uv \in E(H)$ there exists an edge in G between X_u and X_v. The sets X_v are called *branch sets*, and the family $(X_v)_{v \in V(H)}$ is called a *minor model* of H in G. We say that H is an *r-shallow minor* of G if there exists a minor model of H in G such that for every branch set X_v the graph $G[X_v]$ is of radius at most r. The set of all r-shallow minors of a graph G is denoted by $G \triangledown r$, and for a graph class $\mathcal{G}$ we define $\mathcal{G} \triangledown r = \bigcup_{G \in \mathcal{G}} (G \triangledown r)$. The essence of sparsity of a graph or a graph class is captured in the following definition.

▶ **Definition 1.1** (Grad and bounded expansion)**.** For a graph G and an integer $r \geq 0$, we define the *greatest reduced average density (grad) at depth r* as

$$\nabla_r(G) = \max_{M \in G \triangledown r} \text{density}(M) = \max_{M \in G \triangledown r} |E(M)|/|V(M)|.$$

For a graph class $\mathcal{G}$, we let $\nabla_r(\mathcal{G}) = \sup_{G \in \mathcal{G}} \nabla_r(G)$. A graph class $\mathcal{G}$ then has *bounded expansion* if there exists a function $f\colon \mathbb{N} \to \mathbb{R}$ such that for all r we have that $\nabla_r(\mathcal{G}) \leq f(r)$.

Graph classes excluding a topological minor, such as planar and bounded-degree graphs, have bounded expansion [23]. Bounded expansion implies bounded degeneracy, since the degeneracy of G lies between $\nabla_0(G)$ and $2\nabla_0(G)$. However, having bounded expansion is much stronger than just bounded degeneracy: in some sense, it requires every shallow minor to be of bounded degeneracy, with the bound on degeneracy increasing with the depth of the minors. The notion of a *nowhere dense* graph class is a further relaxation of this concept.

As far as the theory of sparsity is concerned, from the point of view of theoretical computer science, of particular importance is the program of establishing fixed-parameter tractability of model checking first order logic on sparse graphs. A long line of work resulted in FPT algorithms for model checking first order formulae on more and more general classes of sparse graphs [6, 11, 12, 17, 19, 27], similarly to the story of kernelization of Dominating Set. Finally, FPT algorithms for the problem have been given for graph classes of bounded expansion by Dvořák et al. [11], and very recently for nowhere dense graph classes by Grohe et al. [19]. This is the ultimate limit of this program: as proven in [22, 11], for any class $\mathcal{G}$ that is not nowhere dense (is *somewhere dense*) and is closed under taking subgraphs, model checking first order formulae on $\mathcal{G}$ is not fixed-parameter tractable (unless FPT = W[1]).

Fixed-parameter tractability of r-Dominating Set on nowhere dense graph classes follows immediately from the result of Grohe et al. [19], as the problem is definable in first order logic for a fixed r; an explicit algorithm was given earlier by Dawar and Kreutzer [7].

Our results. In this work we prove that having bounded expansion or being nowhere dense is sufficient for a graph class to admit an (almost) linear kernel for Dominating Set. Henceforth, for a graph G, we let $\text{ds}(G)$ denote the minimum size of a dominating set of G.

▶ **Theorem 1.2.** *Let $\mathcal{G}$ be a graph class of bounded expansion. There exists a polynomial-time algorithm that given a graph $G \in \mathcal{G}$ and an integer k, either correctly concludes that $\text{ds}(G) > k$ or finds a subset of vertices $Y \subseteq V(G)$ of size $\mathcal{O}(k)$ with the property that $\text{ds}(G) \leq k$ if and only if $\text{ds}(G[Y]) \leq k$.*

▶ **Theorem 1.3.** *Let $\mathcal{G}$ be a nowhere dense graph class and let $\varepsilon > 0$ be a real number. There exists a polynomial-time algorithm that given a graph $G \in \mathcal{G}$ and an integer k, either correctly concludes that $\text{ds}(G) > k$ or finds a subset of vertices $Y \subseteq V(G)$ of size $\mathcal{O}(k^{1+\varepsilon})$ with the property that $\text{ds}(G) \leq k$ if and only if $\text{ds}(G[Y]) \leq k$.*

For r-Dominating Set, for $r > 1$, we can give a linear kernel for any graph class of bounded expansion. Unfortunately, there is a technical subtlety that does not allow us to state the kernel in a nice form as in Theorems 1.2 and 1.3. Instead, we kernelize to an annotated version of the problem, where only a given subset of vertices of G needs to be dominated. The annotated version can be reduced to the classic one by simple gadgeteering but that, unfortunately, may lead to a slight increase in the bounded expansion guarantees.

In the following, by $\text{ds}_r(G)$ we denote the minimum size of an r-Dominating Set in a graph G, while for $Z \subseteq V(G)$, $\text{ds}_r(G, Z)$ denotes the minimum size of a (Z, r)-*dominator* in G, that is, a set $D \subseteq V(G)$ that r-dominates (i.e., is at distance at most r of) every vertex of Z.

▶ **Theorem 1.4.** *Let $\mathcal{G}$ be a graph class of bounded expansion, and let r be a positive integer. There exists a polynomial-time algorithm that given a graph $G \in \mathcal{G}$ and an integer k, either correctly concludes that $\mathrm{ds}_r(G) > k$ or finds subsets of vertices $Z \subseteq W \subseteq V(G)$, where $|W| = \mathcal{O}(k)$, with the property that $\mathrm{ds}_r(G) \leq k$ if and only if $\mathrm{ds}_r(G[W], Z) \leq k$.*

The obtained results strongly generalize the previous results on linear kernels for Dominating Set on sparse graph classes [1, 3, 14, 15, 16], since all the graph classes considered in these results have bounded expansion. Moreover, by giving a linear kernel for r-Dominating Set on any class $\mathcal{G}$ of bounded expansion, we obtain the same result for any H-minor-free or H-topological-minor-free class as well. The existence of such kernels was not known before.

We see the main strength of our results in that they constitute an abrupt turn in the current approach to kernelization of Dominating Set and r-Dominating Set on sparse graphs: the tools used to develop the new algorithms are radically different from all the previously applied techniques. Instead of investigating bidimensionality and treewidth, and relying on intricate decomposition theorems originating in the work on graph minors, our algorithms exploit only basic properties of bounded expansion and nowhere dense graph classes. As a result, the full version of this work presents essentially self-contained proofs of all the stated kernelization results. The only external facts that we use are basic properties of weak and centered colorings, and the constant-factor approximation algorithm for r-Dominating Set of Dvořák [10]. All in all, the results show that only the combinatorial sparsity of a graph class is essential for designing (almost) linear kernels for Dominating Set, and further topological constraints like excluding some (topological) minor are unnecessary.

Lower bounds. We complement our study by proving that for the closely related Connected Dominating Set problem, where the sought dominating set D is additionally required to induce a connected subgraph, the existence of even polynomial kernels for bounded expansion and nowhere dense graph classes is unlikely.

▶ **Theorem 1.5.** *There exists a class of graphs $\mathcal{G}$ of bounded expansion such that* Connected Dominating Set *does not admit a polynomial kernel when restricted to $\mathcal{G}$, unless* NP $\subseteq$ coNP/poly. *Furthermore, $\mathcal{G}$ is closed under taking subgraphs.*

Up to this point, linear kernels for Connected Dominating Set were given for the same family of sparse graph classes as for Dominating Set: a linear kernel for the problem on H-topological-minor-free graphs was obtained by Fomin et al. [16]. Hence, classes of bounded expansion constitute the point where the kernelization complexity of both problems diverge; the connectivity constraint seems to have a completely different nature, and topological properties of the graph class become necessary to handle it efficiently.

Next, we show also that nowhere dense classes form the ultimate limit of parameterized tractability of r-Dominating Set, as it was for model checking first order formulae.

▶ **Theorem 1.6.** *For every somewhere dense graph class $\mathcal{G}$ closed under taking subgraphs, there exists an integer r such that r-*Dominating Set *is $W[2]$-hard on graphs from $\mathcal{G}$.*

In this extended abstract we provide an almost complete proof of Theorems 1.2 and 1.4. The full version of the paper contains [9] full proofs of all results, as well as an extended discussion in the introduction and conclusions.

2 Neighborhoods and closure lemma

The cornerstone of our approach stems from an observation of Gajarský et al. [18] that in a graph G from a class of bounded expansion, the number of possible neighborhoods in a

given subset of vertices X is bounded linearly in $|X|$ and, moreover, the number of vertices that have many neighbors in X is also small. Let $X \subseteq V(G)$ and $v \in V(G)$. We denote by $N_X(v) = N(v) \cap X$ the neighborhood of v in X.

▶ **Lemma 2.1** ([18]). *Let G be a graph, $X \subseteq V(G)$ be a vertex subset, and $R = V(G) \setminus X$. Then for every integer $p \geq \nabla_1(G)$ it holds that*
1. $|\{v \in R \colon |N_X(v)| \geq 2p\}| \leq 2p \cdot |X|$, *and*
2. $|\{A \subseteq X \colon |A| < 2p \text{ and } \exists_{v \in R}\ A = N_X(v)\}| \leq (4^p + 2p)|X|$.
Consequently, the following bound holds:

$$|\{A \subseteq X \colon \exists_{v \in R}\ A = N_X(v)\}| \leq \left(4^{\nabla_1(G)} + 4\nabla_1(G)\right) \cdot |X|.$$

This statement is best suited for the standard Dominating Set problem on graphs of bounded expansion, but to extend our result to r-Dominating Set, we need proper generalizations. Suppose G is a graph and X is a subset of its vertices. For $u \in V(G) \setminus X$ and positive integer r, we define the *r-projection* of u onto X as follows: $M_r^G(u, X)$ is the set of all those vertices $w \in X$, for which there exists a path P in G that starts in u, ends in w, has length at most r, and whose all internal vertices do not belong to X. Whenever the graph is clear from the context, we omit the superscript. In the following we will use the following strengthening of Lemma 2.1(1).

Let $\mathcal{G}$ be a class of bounded expansion, $G \in \mathcal{G}$, r a positive integer, and $X \subseteq V(G)$. A set $\mathrm{cl}_r(X)$ is called an *r-closure* of X if *(1)* $X \subseteq \mathrm{cl}_r(X) \subseteq V(G)$; *(2)* $|\mathrm{cl}_r(X)| \leq ((r-1)\xi+2) \cdot |X|$, where $\xi = \lceil 2\nabla_{r-1}(\mathcal{G}) \rceil$; and *(3)* $|M_r^G(u, \mathrm{cl}_r(X))| \leq \xi(1 + (r-1)\xi)$ for each $u \in V(G) \setminus \mathrm{cl}_r(X)$.

▶ **Lemma 2.2** (Closure lemma). *Let $\mathcal{G}$ be a class of bounded expansion. There exists an algorithm that, given a graph $G \in \mathcal{G}$, positive integer r, and $X \subseteq V(G)$, computes the* r-closure *of X.*

Proof. Consider the following iterative procedure: *(1)* Start with $H = G$ and $Y = X$. We will maintain the invariant that $Y \subseteq V(H)$. *(2)* As long as there exists a vertex $u \in V(H) \setminus Y$ with $|M_r^H(u, Y)| \geq \xi$, do the following:

- Select an arbitrary subset $Z_u \subseteq M_r^H(u, Y)$ of size ξ.
- For each $w \in Z_u$, select a path P_w that starts at u, ends at w, has length at most r, and all its internal vertices are in $V(H) \setminus Y$.
- Modify H by contracting $\bigcup_{w \in Z_u}(V(P_w) \setminus \{w\})$ onto u, and add the obtained vertex to Y.

Observe that in a round of the procedure above we always make a contraction of a connected subgraph of $H - Y$ of radius at most $r - 1$. Also, the resulting vertex falls into Y and hence does not participate in future contractions. Thus, at each point H is an $(r-1)$-shallow minor of G. For any moment of the procedure and any $u \in V(H)$, by $\tau(u)$ we denote the subset of original vertices of G that were contracted onto u during earlier rounds. Note that either $\tau(u) = \{u\}$ when u is an original vertex of G, or $|\tau(u)| \leq 1 + (r-1)\xi$.

We claim that the presented procedure stops after at most $|X|$ rounds. Suppose otherwise, that we successfully constructed the graph H and subset Y after $|X| + 1$ rounds. Examine graph $H[Y]$. This graph has $2|X| + 1$ vertices: $|X|$ original vertices of X and $|X| + 1$ vertices that were added during the procedure. Whenever a vertex u is added to Y after contraction, then it introduces at least ξ new edges to $H[Y]$: these are edges that connect the contracted vertex with the vertices of Z_u. Hence, $H[Y]$ has at least $\xi(|X| + 1)$ edges, which means that

$$\mathrm{density}(H[Y]) = \frac{|E(H[Y])|}{|Y|} \geq \frac{\xi(|X| + 1)}{2|X| + 1} > \nabla_{r-1}(\mathcal{G}).$$

This is a contradiction with the fact that H is an $(r-1)$-shallow minor of G.

Therefore, the procedure stops after at most $|X|$ rounds producing pair (H, Y) where $|M_r^H(u, Y)| < \xi$ for each $u \in V(H) \setminus Y$. Define $\mathrm{cl}_r(X) = \tau(Y) = \bigcup_{u \in Y} \tau(u)$. Property *(1)* is obvious. Since $|\tau(u)| = 1$ for each original vertex of X and $|\tau(u)| \leq 1 + (r-1)\xi$ for each u that was added during the procedure, property *(2)* follows. We are left with property *(3)*.

By the construction $V(H) \setminus Y = V(G) \setminus \mathrm{cl}_r(X)$. Take any $u \in V(H) \setminus Y$ and observe that $M_r^G(u, \mathrm{cl}_r(X)) \subseteq \tau(M_r^H(u, Y))$. Since $|M_r^H(u, Y)| < \xi$ for each $u \in V(H) \setminus Y$ and $|\tau(u)| \leq 1 + (r-1)\xi$ for each $u \in V(H)$, property *(3)* follows. ◀

As we can safely assume $\nabla_{r-1}(\mathcal{G}) \geq 1$ (otherwise $\mathcal{G}$ contains only forests), we can use simplified, weaker bounds: $|\mathrm{cl}_r(X)| \leq 3r\nabla_{r-1}(\mathcal{G}) \cdot |X|$ and $|M_r^G(u, \mathrm{cl}_r(X))| \leq 9r\nabla_{r-1}(\mathcal{G})^2$. Observe that Lemma 2.2 is not merely a generalization of Lemma 2.1(1) to r-neighborhoods. It shows that a certain maximality property can be achieved; this property may be not true if, even for $r = 1$, we would construct $\mathrm{cl}_r(X)$ from X by just adding all vertices with many neighbors in X.

The generalization of Lemma 2.1(2) which we will use later is the following.

▶ **Lemma 2.3.** *Let $\mathcal{G}$ be a class of bounded expansion and let r be a positive integer. Let $G \in \mathcal{G}$ be a graph and $X \subseteq V(G)$. Then*

$$|\{Y : Y = M_r(u, X) \text{ for some } u \in V(G) \setminus X\}| \leq c \cdot |X|,$$

for some constant c depending only on r and the grads of $\mathcal{G}$.

The proof of Lemma 2.3 is essentially contained in the PhD thesis of the eight author, Reidl [25]. For the sake of completeness, in the full version we give a self-contained proof based on the presentation of Reidl.

Finally, for the proof of Theorem 1.4 for $r > 1$ we will need the following lemma.

▶ **Lemma 2.4** (Short paths closure lemma)**.** *Let $\mathcal{G}$ be a class of bounded expansion and let r be a positive integer. Let $G \in \mathcal{G}$ be a graph and $X \subseteq V(G)$. Then there is a superset of vertices $X' \supseteq X$ with the following properties:*

1. *Whenever $\mathrm{dist}_G(u, v) \leq r$ for some distinct $u, v \in X$, then $\mathrm{dist}_{G[X']}(u, v) = \mathrm{dist}_G(u, v)$.*
2. *$|X'| \leq Q_r(\nabla_{r-1}(\mathcal{G})) \cdot |X|$ for some polynomial Q_r.*

Moreover, X' can be computed in polynomial time.

Proof sketch. We start with $X' := X_0 := \mathrm{cl}_r(X)$ and then, for every $u, v \in X_0$, we add to X' a path $P_{u,v}$ between u and v that is shortest among paths that do not visit X_0 apart from the endpoints, provided that it is of length at most r. The first required property of X' and polynomial time computability is immediate.

For the size bound, first note that the properties of $X_0 = \mathrm{cl}_r(X)$ ensure that every vertex $x \in X' \setminus X_0$ can lie only on a constant number of paths $P_{u,v}$, as the endpoints of such path needs to lie in $M_r(x, X_0)$. On the other hand, if too many paths $P_{u,v}$ are pairwise disjoint, they constitute a dense r-shallow minor of G. We infer that only $\mathcal{O}(|X_0|) = \mathcal{O}(|X|)$ paths $P_{u,v}$ are added, and hence $|X'| = \mathcal{O}(|X|)$, as required. ◀

3 Linear kernel in graphs of bounded expansion

Let us fix a graph class $\mathcal{G}$ that has bounded expansion, and let (G, k) be the input instance of r-Dominating Set, where $G \in \mathcal{G}$. We assume that $\mathcal{G}$ is fixed, and hence so are also the values of $\nabla_i(\mathcal{G})$ for all nonnegative integers i. We start with the following pivot definition.

▶ **Definition 3.1** (r-domination core)**.** Let G be a graph and Z be a subset of vertices. We say that Z is an *r-domination core* in G if every minimum-size (Z, r)-dominator in G is also an r-dominating set in G.

Clearly, the whole $V(G)$ is an r-domination core, but we will look for an r-domination core that is small in terms of k. Note that if Z is an r-domination core, then $\mathrm{ds}_r(G) = \mathrm{ds}_r(G, Z)$. Let us remark that in this definition we do not require that every (Z, r)-dominator is an r-dominating set in G; there can exist (Z, r)-dominators that are not of minimum size and that do not dominate the whole graph.

The heart of our argument lies in providing a small domination core.

▶ **Theorem 3.2.** *There exists a constant $c_{coresize}$ depending on r and the grads of $\mathcal{G}$ only and a polynomial-time algorithm that, given an instance (G, k) where $G \in \mathcal{G}$, either correctly concludes that $\mathrm{ds}_r(G) > k$, or finds an r-domination core $Z \subseteq V(G)$ with $|Z| \leq c_{coresize} \cdot k$.*

We fix G and k in the following to improve readability. For the proof of Theorem 3.2 we start with $Z = V(G)$ and gradually reduce $|Z|$ by removing one vertex at a time, while maintaining the invariant that Z is an r-domination core. To this end, we need to prove the following lemma, from which Theorem 3.2 follows trivially as explained:

▶ **Lemma 3.3.** *There exists a constant $c_{coresize}$ depending on r and the grads of $\mathcal{G}$ only and a polynomial-time algorithm that, given an r-domination core $Z \subseteq V(G)$ with $|Z| > c_{coresize} \cdot k$, either correctly concludes that $\mathrm{ds}_r(G) > k$, or finds a vertex $z \in Z$ such that $Z \setminus \{z\}$ is still an r-domination core.*

In Sections 3.1 and 3.2 we prove Lemma 3.3. Then, in Section 3.3 we show how Theorem 3.2 implies Theorems 1.2 and 1.4.

3.1 Iterative extraction of Z-dominators

The first phase of the algorithm of Lemma 3.3 is to build a structural decomposition of the graph G. More precisely, we try to "pull out" a small set X of vertices that r-dominates Z, so that after removing them, Z contains a large subset S that is $2r$-scattered in the remaining graph.[1] Given such a structure, intuitively we can argue that in any optimal (Z, r)-dominator, vertices of X serve as "hubs" that route almost all the domination paths leading to vertices of S. This is because any vertex of $V(G) \setminus X$ can r-dominate only at most one vertex from S via a path that avoids X. Since S will be large compared to X, some vertices of S will be indistinguishable from the point of view of r-domination routed through X, and these will be precisely the vertices that can be removed from the domination core. The identification of the irrelevant dominatee will be the goal of the second phase of the algorithm, whereas the goal of this phase is to construct the pair (X, S).

Our main tool in this part is the constant-factor approximation for r-Dominating Set proved by Dvořák [10]. We use the following convenient form; the full version of our work describes in detail how to derive it from the work of Dvořák.

▶ **Lemma 3.4.** *Let r be a positive integer. There is a polynomial P_r and a polynomial-time algorithm that, given a graph G, a vertex subset $Z \subseteq V(G)$ and an integer k, finds either*

- *a (Z, r)-dominator in G of size at most $P_r(\nabla_r(G)) \cdot k$, or*
- *a subset of Z of size at least $k + 1$ that is $2r$-scattered in G.*

[1] Recall that a set $S \subseteq V(G)$ is *ℓ-scattered* if every two distinct vertices of S are within distance greater than ℓ; a $2r$-scattered set of size $k + 1$ is an obstruction for an r-dominating set of size k.

Let $C_{\mathrm{dv}} = P_r(\nabla_r(\mathcal{G}))$ be the approximation ratio of the algorithm of Lemma 3.4. Given Z, we first apply the algorithm of Lemma 3.4 to G, Z, and the parameters r and k. Thus, we either find a (Z, r)-dominator Y_1 such that $|Y_1| \leq C_{\mathrm{dv}} \cdot k$, or we find a subset $S \subseteq Z$ of size at least $k+1$ that is $2r$-scattered in G. In the latter case, since S is an obstruction to an r-dominating set of size at most k, we may terminate the algorithm and provide a negative answer. Hence, from now on we assume that Y_1 has been successfully constructed.

Let C_0 be a constant depending on $\nabla(\mathcal{G})$, to be defined later. Now, in search for the pair (X, S), we inductively construct sets $X_1, Y_2, X_2, Y_3, \ldots$ such that $Y_1 \subseteq X_1 \subseteq Y_2 \subseteq \cdots$ using the following definitions:

- If Y_i is already defined, then set $X_i = \mathrm{cl}_{3r}(Y_i)$.
- If X_i is already defined, then apply the algorithm of Lemma 3.4 to $G - X_i$, $Z \setminus X_i$, and the parameters r and $C_0 \cdot |X_i|$.
  1. Suppose the algorithm finds a set $S \subseteq Z \setminus X_i$ that is $2r$-scattered in $G - X_i$ and has cardinality greater than $C_0 \cdot |X_i|$. Then we let $X = X_i$, terminate the procedure and proceed to the second phase with the pair (X, S).
  2. Otherwise, the algorithm has found a $(Z \setminus X_i, r)$-dominator D_{i+1} in $G - X_i$ of size at most $C_{\mathrm{dv}} \cdot C_0 \cdot |X_i|$. Then set $Y_{i+1} = X_i \cup D_{i+1}$ and proceed.

Let $\Gamma_{\mathrm{cl}} = 9r\nabla_{3r-1}(\mathcal{G})$ be the bound on the size blow-up in Lemma 2.2 applied to radius $3r$, and let $\Delta_{\mathrm{cl}} = 27r\nabla_{3r-1}(\mathcal{G})^2$ be the upper bound on the sizes of $3r$-projections given by Lemma 2.2. From Lemmas 3.4, 2.2, and a trivial induction we infer that the following bounds hold for all i for which (Y_i, X_i) were constructed:

$$|Y_i| \leq C_{\mathrm{dv}}\Gamma_{\mathrm{cl}}^{i-1}(1 + C_{\mathrm{dv}}C_0)^{i-1} \cdot k \qquad \text{and} \qquad |X_i| \leq C_{\mathrm{dv}}\Gamma_{\mathrm{cl}}^{i}(1 + C_{\mathrm{dv}}C_0)^{i-1} \cdot k.$$

For a nonnegative integer i, let $K_i = C_{\mathrm{dv}}\Gamma_{\mathrm{cl}}^{i}(1 + C_{\mathrm{dv}}C_0)^{i-1}$.

In this manner, the algorithm consecutively extracts dominators $D_2, D_3, D_4, \ldots$ and performs $3r$-closure, constructing sets $X_2, X_3, X_4, \ldots$ up to the point when case (1) is encountered. Then the computation is terminated and the sought pair (X, S) is constructed. We now claim that case (1) always happens within a constant number of iterations.

▶ **Lemma 3.5.** *Let $\Lambda = \sum_{i=0}^{r} \Delta_{\mathrm{cl}}^i \leq (r+1)\Delta_{\mathrm{cl}}^r$. Assuming that $|Z| > K_\Lambda \cdot k$, the construction terminates yielding some pair (X, S) before performing Λ iterations.*

Proof. For the sake of contradiction, suppose Y_Λ and X_Λ were successfully constructed. Since $|Z| > K_\Lambda \cdot k$ and $|X_\Lambda| \leq K_\Lambda \cdot k$, there is some vertex $u \in Z \setminus X_\Lambda$. For an index $1 \leq i \leq \Lambda$, we shall say that a vertex $w \in X_i \setminus X_{i-1}$ is *i-good* if there is a path P that starts at u, ends at w, has length at most r, and all its internal vertices do not belong to X_i (we denote $X_0 = \emptyset$). Vertex w is *good* if it is good for some index i.

▶ **Claim 3.6.** *The number of good vertices is at most $\Lambda - 1$.*

Proof of claim. Let w be any good vertex, and let P be a path certifying this. Let $q \leq r$ be the length of P, and denote the vertices of P by u_i for $0 \leq i \leq q$, where $u_0 = u$ and $u_q = w$. Observe that internal vertices of P can belong only to sets $X_j \setminus X_{j-1}$ for $j > i$, or to $V(G) \setminus X_\Lambda$. We say that a vertex u_ℓ of P is *important* if there is an index j, with $i \leq j \leq \Lambda$, such that $u_\ell \in X_j \setminus X_{j-1}$ but $u_{\ell'} \notin X_j$ for all $\ell' < \ell$. Clearly, $w = u_q$ is important. Let $\ell_1 < \ell_2 < \ldots < \ell_p = q$ be the indices of important vertices on P, and let $j_1 > j_2 > \ldots > j_p$ be such that $u_{\ell_i} \in X_{j_i} \setminus X_{j_i - 1}$, for all $1 \leq i \leq p$. We will denote $\ell_0 = 0$, so $u_{\ell_0} = u$, and $j_0 = \Lambda + 1$ (denoting $X_{\Lambda+1} = V(G)$).

Consider any index i with $1 \leq i \leq p$. Observe that on the part between $u_{\ell_{i-1}}$ and u_{ℓ_i}, path P never entered $X_{j_{i-1}-1}$, because first such entrance would constitute an important

vertex that was not recorded. Since $u_{\ell_i} \in X_{j_{i-1}-1}$, we infer that $u_{\ell_i} \in M_r^G(u_{\ell_{i-1}}, X_{j_{i-1}-1})$. Since $X_{j_{i-1}-1} = \mathrm{cl}_{3r}(Y_{j_{i-1}-1})$ by the construction, we infer that $|M_r^G(u_{\ell_{i-1}}, X_{j_{i-1}-1})| \leq \Delta_{\mathrm{cl}}$. Therefore, once vertex $u_{\ell_{i-1}}$ is selected, there are at most Δ_{cl} choices for the next important vertex u_{ℓ_i}. We infer that the choice of the sequence of important vertices on P can be modeled by taking at most r decisions, each from a selection of at most Δ_{cl} options. Since w is the last important vertex, there are at most $\sum_{i=1}^{r} \Delta_{\mathrm{cl}}^i = \Lambda - 1$ ways to select w. ◀

▶ **Claim 3.7.** *For every $1 \leq i \leq \Lambda$, there is an i-good vertex.*

Proof of claim. Recall that $D_i \subseteq X_i \setminus X_{i-1}$ is a $(Z \setminus X_{i-1}, r)$-dominator in the graph $G - X_{i-1}$. Since $u \in Z \setminus X_{i-1}$, in $G - X_{i-1}$ there is a path P of length at most r from u to a vertex of D_i. Take w to be the first vertex of this path that belongs to $X_i \setminus X_{i-1}$. Then the prefix of P from u to w certifies that w is an i-good vertex. ◀

Claims 3.6 and 3.7 contradict each other, which finishes the proof. ◀

In Lemma 3.3 we will set $c_{\mathrm{coresize}} = K_\Lambda$, so that Lemma 3.5 can be applied. Therefore, unless the size of Z is bounded by $K_\Lambda \cdot k$, the construction terminates within $\Lambda = \sum_{i=0}^{r} \Delta_{\mathrm{cl}}^i \leq (r+1)\Delta_{\mathrm{cl}}^r$ iterations with a pair (X, S). By the construction of X and S, we have the following properties:

- $|X| \leq K_\Lambda \cdot k$ and $|S| > C_0 \cdot |X|$;
- X is a (Z, r)-dominator in G (because $Y_1 \subseteq X$);
- for each $u \in V(G) \setminus X$, we have $|M_{3r}^G(u, X)| \leq \Delta_{\mathrm{cl}}$;
- $S \subseteq Z \setminus X$ and S is $2r$-scattered in $G - X$.

3.2 Finding an irrelevant dominatee

Given G, Z, and the constructed sets X and S, we denote by $R = V(G) \setminus X$ the set of vertices outside X. Using this notation, S is $2r$-scattered in the graph $G[R]$. Recall that for any vertex $u \in R$, we have $|M_{3r}(u, X)| \leq \Delta_{\mathrm{cl}}$.

Define the following equivalence relation $\simeq$ on S: for $u, v \in S$, let

$$u \simeq v \quad \Leftrightarrow \quad M_i(u, X) = M_i(v, X) \text{ for each } 1 \leq i \leq 3r.$$

Let us denote by C_{nei} the constant c given by Lemma 2.3 for class $\mathcal{G}$ and radius $3r$. Hence, the number of different $3r$-projections in X of vertices of R is bounded by $C_{\mathrm{nei}} \cdot |X|$.

▶ **Lemma 3.8.** *The equivalence relation $\simeq$ has at most $C_{\mathrm{nei}} \cdot (3r)^{\Delta_{\mathrm{cl}}} \cdot |X|$ classes.*

Proof. Observe that for each $u \in S$,

$$M_1(u, X) \subseteq M_2(u, X) \subseteq \ldots \subseteq M_{3r-1}(u, X) \subseteq M_{3r}(u, X).$$

By Lemma 2.3, the number of choices for $M_{3r}(u, X)$ is at most $C_{\mathrm{nei}} \cdot |X|$. Moreover, since $u \in R$, we have that $|M_{3r}(u, X)| \leq \Delta_{\mathrm{cl}}$. Hence, to define the sets $M_i(u, X)$ for $1 \leq i < 3r$ it suffices, for every $w \in M_{3r}(u, X)$, to choose the smallest index j, $1 \leq j \leq 3r$, such that $w \in M_j(u, X)$. The number of such choices is at most $(3r)^{\Delta_{\mathrm{cl}}}$, and hence the claim follows. ◀

We can finally set the constant C_0 that was introduced in the previous section. We let $C_0 = (\Delta_{\mathrm{cl}} + 1) \cdot C_{\mathrm{nei}} \cdot (3r)^{\Delta_{\mathrm{cl}}}$. Since we have that $|S| > C_0 \cdot |X|$, from Lemma 3.8 and the pigeonhole principle we infer that there is a class κ of relation $\simeq$ with $|\kappa| > \Delta_{\mathrm{cl}} + 1$. Note that we can find such a class κ in polynomial time, by computing the classes of $\simeq$ directly

from the definition and examining their sizes. We are ready to prove the final lemma of this section: any vertex of κ can be removed from the r-domination core Z (recall that $S \subseteq Z$), concluding the proof of Lemma 3.3.

▶ **Lemma 3.9.** *Let z be an arbitrary vertex of κ. Then $Z \setminus \{z\}$ is an r-domination core.*

Proof. Let $Z' = Z \setminus \{z\}$. Take any minimum-size (Z', r)-dominator D in G. If D also dominates z, then D is a minimum-size (Z, r)-dominator as well. Since Z was an r-domination core, we infer that D is an r-dominating set in G, and we are done. Hence, suppose z is not r-dominated by D. We prove that this case leads to a contradiction.

Every vertex $s \in \kappa \setminus \{z\}$ is r-dominated by D. For each such s, let $v(s)$ be an arbitrarily chosen vertex of D that r-dominates s, and let $P(s)$ be an arbitrarily chosen path of length at most r that connects $v(s)$ with s.

▶ **Claim 3.10.** *For each $s \in \kappa \setminus \{z\}$, path $P(s)$ does not pass through any vertex of X (in particular $v(s) \notin X$). Consequently, vertices $v(s)$ for $s \in \kappa \setminus \{z\}$ are pairwise different.*

Proof of claim. Suppose otherwise and let w be the vertex of $V(P(s)) \cap X$ that is closest to s on $P(s)$. Then the suffix of $P(s)$ from w to s certifies that $w \in M_j(s, X)$, for j being the length of this suffix. As $s \simeq z$, we also have that $w \in M_j(z, X)$, so there is a path Q of length at most j from w to z. By concatenating the prefix of $P(s)$ from $v(s)$ to w with Q we obtain a walk of length at most r from $v(s)$ to z, a contradiction with the assumption that z is not r-dominated by D.

For the second part of the claim, suppose $v(s) = v(s')$ for some distinct $s, s' \in \kappa \setminus \{z\}$. Then the concatenation of $P(s)$ and $P(s')$ would be a path of length at most $2r$ connecting s and s' that is entirely contained in $G[R]$. This would be a contradiction with the fact that S is $2r$-scattered in $G[R]$. ◀

Let $W = \{v(s) \colon s \in \kappa \setminus \{z\}\}$. From Claim 3.10 we have that $|W| = |\kappa \setminus \{z\}| \geq \Delta_{\mathrm{cl}} + 1$. Define $D' = (D \setminus W) \cup M_{3r}(z, X)$. Since $|M_{3r}(z, X)| \leq \Delta_{\mathrm{cl}}$, we have that $|D'| < |D|$.

▶ **Claim 3.11.** *D' is a (Z', r)-dominator.*

Proof of claim. For the sake of contradiction, suppose there is some $a \in Z'$ that is not r-dominated by D'. Since a was r-dominated by D and $D \setminus D' = W$, there must be a vertex $s \in \kappa \setminus \{z\}$ such that vertex $v(s)$ r-dominates a. Consequently, in G there is a path Q_0 of length at most r that leads from $v(s)$ to a. Furthermore, since X is a (Z, r)-dominator, there is a path Q_1 of length at most r that leads from a to some $x \in X$. Let Q be the concatenation of $P(s)$, Q_0, and Q_1; Q is a walk of length at most $3r$ that connects s and $x \in X$.

Let x' be the first (closest to s) vertex on Q that belongs to X; such a vertex exists as $x \in X$ is on Q. As the length of Q is at most $3r$, we have $x' \in M_{3r}(s, X)$. Since $s \simeq z$, we have $x' \in M_{3r}(z, X)$, and, consequently, $x' \in D'$. However, by Claim 3.10, x' does not lie on $P(s)$. Hence x' lies on the part of Q between $v(s)$ and x, but each vertex of this part is at distance at most r from a on Q. Thus a is r-dominated by x', a contradiction. ◀

As $|D'| < |D|$, Claim 3.11 is a contradiction with the assumption that D is a minimum-size (Z', r)-dominator. This concludes the proof. ◀

3.3 Reducing dominators

In the rest of this section we work with arbitrary r towards the proof of Theorem 1.4. At some point we will argue that for $r = 1$, the statement of Theorem 1.2 is immediate. Having reduced the number of vertices whose domination is essential, we arrive at the situation where the vast majority of vertices serve only the role of dominators, or, when $r > 1$, they serve as connections between dominators with dominatees. Now, it is relatively easy to reduce the number of candidate dominators in one step. This immediately gives the sought kernel for $r = 1$, i.e., proves Theorem 1.2. For $r > 2$, the treatment of vertices connecting dominators and dominatees without introducing additional gadgets turns out to be problematic. Therefore, we are unable to give a kernel that is an induced subgraph of the original graph, and we resort to the statement of Theorem 1.4.

The algorithm of Theorem 1.4 works as follows. First, we apply the algorithm of Theorem 3.2 to compute a small domination core in the graph. In case the algorithm gives a negative answer, we output that $\mathrm{ds}_r(G) > k$. Hence, from here on, we assume that we have correctly computed an r-domination core $Z_0 \subseteq V(G)$ of size $\mathcal{O}(k)$.

Compute $Z = \mathrm{cl}_r(Z_0)$ using Lemma 2.2; then we have that $|Z| \leq 3r\nabla_{r-1}(\mathcal{G})|Z_0| = \mathcal{O}(k)$. Observe that in any graph, any superset of an r-domination core is also an r-domination core; this follows easily from the definition. Consequently, Z is an r-domination core in G. Partition $V(G) \setminus Z$ into equivalence classes with respect to the following relation $\simeq$, defined similarly as in Section 3.2: For $u, v \in V(G) \setminus Z$, let:

$$u \simeq v \quad \Leftrightarrow \quad M_i(u, Z) = M_i(v, Z) \text{ for each } 1 \leq i \leq r.$$

From Lemma 2.2 we know that for each $u \in V(G) \setminus Z$, it holds that $|M_i(v, Z)| \leq 9r\nabla_{r-1}(\mathcal{G})^2$. Moreover, Lemma 2.3 implies that the number of possible different projections $M_r(u, Z)$ for $u \in V(G) \setminus Z$ is at most $c \cdot |Z|$, for some constant c depending on the grads of $\mathcal{G}$. Hence, using the same reasoning as in the proof of Lemma 3.8 we obtain the following. (Fully formal verification of Claims 3.12–3.15 is contained in the full version.)

▶ **Claim 3.12.** *For $C = c \cdot r^{9r\nabla_{r-1}(\mathcal{G})^2}$, the equivalence relation $\simeq$ has at most $C \cdot |Z|$ classes.*

Construct Y as follows: start with Z and, for each equivalence class κ of relation $\simeq$, add an arbitrarily selected member v_κ of κ. Hence we have that $|Y| \leq (C + 1) \cdot |Z|$, so in particular $|Y| = \mathcal{O}(k)$. The following claim follows from a standard replacement argument.

▶ **Claim 3.13.** *There exists a minimum-size r-dominating set in G that is contained in Y.*

Theorem 1.2 now follows from the following immediate claim.

▶ **Claim 3.14.** *If $r = 1$, then $\mathrm{ds}(G) \leq k$ if and only if $\mathrm{ds}(G[Y]) \leq k$.*

For Theorem 1.4, we need not only to preserve the potential dominatees and dominators (the set Y), but also distances (up to length r) between them. To this end, we run the algorithm of Lemma 2.4 on set Y, and let $W = Y'$ be the obtained superset of Y. By Lemma 2.4 we have that $|W| = \mathcal{O}(k)$. Then Theorem 1.4 follows immediately from the following claim, which in turn follows easily from the properties of the set W promised by Lemma 2.4.

▶ **Claim 3.15.** *$\mathrm{ds}_r(G) \leq k$ if and only if $\mathrm{ds}_r(G[W], Z) \leq k$.*

4 Conclusions

We have shown that, for each $r \geq 1$, r-Dominating Set admits a linear kernel on any graph class of bounded expansion. Before this work, the most general family of graph classes where

such a kernelization result was known were apex-minor-free graphs [14], whereas in the case of the classic Dominating Set, linear kernels were shown also for general H-minor-free [15] and H-topological-minor-free classes [16]. Moreover, for $r = 1$, i.e., the Dominating Set problem, we can also give a kernel on any nowhere dense class of graphs, at the cost of increasing the size bound to almost linear, i.e., $\mathcal{O}(k^{1+\varepsilon})$ for any $\varepsilon > 0$. These results vastly and broadly extend the current frontier of kernelization results for domination problems on sparse graph classes.

The most important question left is understanding the kernelization complexity of r-Dominating Set on nowhere dense graph classes. So far we know that this problem admits a linear kernel on any class of bounded expansion, for each r, whereas on any somewhere dense class closed under taking subgraphs, for some r it is W[2]-hard. Our approach for bounded expansion graph classes fails to generalize to nowhere dense classes mostly because of technical reasons. We believe that, in fact, for any nowhere dense class $\mathcal{G}$ and any positive integer r, r-Dominating Set has an almost linear kernel on $\mathcal{G}$. Together with the lower bound of Theorem 1.6, this would confirm the following dichotomy conjecture that we pose:

▶ **Conjecture 1.** Let $\mathcal{G}$ be a graph class closed under taking subgraphs and $r \in \mathbb{N}$. Then:

- If $\mathcal{G}$ is nowhere dense, then for every $r \geq 1$ and real $\varepsilon > 0$, r-Dominating Set admits an $\mathcal{O}(k^{1+\varepsilon})$ kernel on $\mathcal{G}$.
- If $\mathcal{G}$ is somewhere dense, then r-Dominating Set is W[2]-hard on $\mathcal{G}$ for some $r \geq 1$.

References

1. Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for Dominating Set. *J. ACM*, 51(3):363–384, 2004.
2. Noga Alon and Shai Gutner. Kernels for the Dominating Set problem on graphs with an excluded minor. *Electronic Colloquium on Comp. Complexity (ECCC)*, 15(066), 2008.
3. H. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. M. Thilikos. (Meta) Kernelization. In *Proceedings of the 50th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 629–638. IEEE, 2009.
4. Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (meta) Kernelization. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 629–638. IEEE Computer Society, 2009.
5. Marek Cygan, Fabrizio Grandoni, and Danny Hermelin. Tight kernel bounds for problems on graphs with small degeneracy. In *Proc. of the 21st Annual European Symp. on Algorithms (ESA)*, volume 8125 of *LNCS*, pages 361–372. Springer, 2013.
6. Anuj Dawar, Martin Grohe, and Stephan Kreutzer. Locally excluding a minor. In *Proceedings of the 22nd IEEE Symposium on Logic in Computer Science (LICS)*, pages 270–279. IEEE Computer Society, 2007.
7. Anuj Dawar and Stephan Kreutzer. Domination problems in nowhere-dense classes. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 4 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 157–168. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2009.
8. Rodney G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer-Verlag, New York, 1999.
9. Pål Grønås Drange, Markus Dregi, Fedor V. Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, Felix Reidl, Saket Saurabh, Fernando Sánchez Villaamil, and Somnath Sikdar. Kernelization and Sparseness: the case of Dominating Set. *CoRR*, abs/1309.4022v1, 2014.
10. Z. Dvořák. Constant-factor approximation of the domination number in sparse graphs. *Eur. J. Comb.*, 34(5):833–840, 2013.

11 Zdenek Dvořák, Daniel Král', and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36, 2013.

12 Jörg Flum and Martin Grohe. Fixed-parameter tractability, definability, and model-checking. *SIAM J. Comput.*, 31(1):113–145, 2001.

13 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.

14 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 503–510. SIAM, 2010.

15 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Linear kernels for (Connected) Dominating Set on H-minor-free graphs. In *Proc. of the 22nd Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 82–93. SIAM, 2012.

16 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Linear kernels for (Connected) Dominating Set on graphs with excluded topological subgraphs. In *Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 20 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 92–103. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013.

17 Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001.

18 J. Gajarský, P. Hliněný, J. Obdržálek, S. Ordyniak, F. Reidl, P. Rossmanith, F. Sánchez Villaamil, and S. Sikdar. Kernelization using structural parameters on sparse graph classes. In *Proceedings of the 21st Annual European Symposium on Algorithms (ESA)*, volume 8125 of *Lecture Notes in Comput. Sci.*, pages 529–540. Springer, 2013. Full version available at `http://arxiv.org/abs/1302.6863`.

19 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 (STOC)*, pages 89–98. ACM, 2014.

20 Martin Grohe and Dániel Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 173–192. ACM, 2012.

21 Shai Gutner. Polynomial kernels and faster algorithms for the Dominating Set problem on graphs with an excluded minor. In *Proceedings of the 4th International Workshop on Parameterized and Exact Computation (IWPEC)*, volume 5917 of *Lecture Notes in Comput. Sci.*, pages 246–257. Springer, 2009.

22 Stephan Kreutzer. Algorithmic meta-theorems. In J. Esparza, C. Michaux, and C. Steinhorn, editors, *Finite and Algorithmic Model Theory*, chapter 5, pages 177–270. Cambridge University Press, 2011.

23 J. Nešetřil and P. Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.

24 Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Polynomial kernels for Dominating Set in graphs of bounded degeneracy and beyond. *ACM Transactions on Algorithms*, 9(1):11, 2012. `doi:10.1145/2390176.2390187`.

25 Felix Reidl. *Structural sparseness and complex networks*. Dr., Aachen, Techn. Hochsch., Aachen, 2015. Aachen, Techn. Hochsch., Diss., 2015. URL: `https://publications.rwth-aachen.de/record/565064`.

26 Neil Robertson and Paul D. Seymour. Graph Minors. XVI. Excluding a non-planar graph. *J. Comb. Theory, Ser. B*, 89(1):43–76, 2003.

27 Detlef Seese. Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, 1996.

Canonizing Graphs of Bounded Tree Width in Logspace

Michael Elberfeld[1] and Pascal Schweitzer[2]

1 **RWTH Aachen University, Aachen, Germany**
2 **RWTH Aachen University, Aachen, Germany**

Abstract

Graph canonization is the problem of computing a unique representative, a canon, from the isomorphism class of a given graph. This implies that two graphs are isomorphic exactly if their canons are equal. We show that graphs of bounded tree width can be canonized in deterministic logarithmic space (logspace). This implies that the isomorphism problem for graphs of bounded tree width can be decided in logspace. In the light of isomorphism for trees being hard for the complexity class logspace, this makes the ubiquitous classes of graphs of bounded tree width one of the few classes of graphs for which the complexity of the isomorphism problem has been exactly determined.

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

Keywords and phrases algorithmic graph theory, computational complexity, graph isomorphism, logspace, tree width

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.32

1 Introduction

The *graph isomorphism problem* (ISOMORPHISM) – deciding whether two given graphs are the same up to renaming vertices – is one of the few fundamental problems in NP for which we neither know that it is solvable in polynomial-time nor that it is NP-complete. Since NP-hardness would imply a collapse of the polynomial hierarchy to its second level [4, 24], significant effort has been put into better understanding the graph-theoretic requirements on input graphs that make ISOMORPHISM polynomial-time decidable. A classical result of Bodlaender [3] shows that ISOMORPHISM is in P (deterministic polynomial time) for graphs of *bounded tree width* [3]. It is also in P for other graph classes like planar graphs [15, 25] and more general graphs with a crossing-free embedding into a fixed surface [12, 13, 21]. Since ISOMORPHISM is hard for NL (nondeterministic logarithmic space) [26], a deeper complexity-theoretic insight behind the polynomial-time algorithms for embeddable graphs is given by the fact that ISOMORPHISM for graphs embeddable into the plane [7] or a fixed surface [10] is in L (deterministic logarithmic space, also called *logspace*). So far, it has been an open question whether for graphs of bounded tree width the isomorphism problem can also be solved in logspace.

Guided by the goal to determine the complexity of the isomorphism problem for graphs of bounded tree width, there has been a sequence of partial results. Bodlaender's algorithm [3], which places ISOMORPHISM for graphs of bounded tree width in P, was first refined to an upper bound in terms of logarithmic-depth circuits with threshold gates (that means, circuits defining the complexity class TC^1) [14] and later improved to use semi-unbounded fan-in Boolean gates (that means, circuits defining the complexity class SAC^1) [6]. Since the chain


© Michael Elberfeld and Pascal Schweitzer;
licensed under Creative Commons License CC-BY

$L \subseteq NL \subseteq SAC^1 \subseteq TC^1 \subseteq P$ is all we know about the relations of these classes, these works leave the question for a logspace approach for graphs of bounded tree width open. Logspace approaches are known for small constant bounds on the tree width. Indeed, Lindell's [18] classical approach to testing isomorphism of trees provides us with a logspace algorithm for graphs of tree width at most 1. This was generalized to graphs of tree width at most 2 [2] and results for graphs without K_5 as a minor [8] apply to graphs of tree width at most 3. Moreover, k-trees, the maximal graphs of tree width k, admit logspace isomorphism tests [1] as well as graphs with a bounded tree depth [5]. While providing us with logspace algorithms for ever larger classes of graphs, the general question remained open.

Results. Our first main result answers the above question in its most general way by showing that the isomorphism problem for graphs of bounded tree width can be solved in logspace. Together with a result of Jenner et al. [16], showing that the isomorphism problem for trees is L-hard, this pinpoints the complexity of the isomorphism problem for graphs of bounded tree width to be L-complete.

▶ **Theorem 1.** *For every positive $k \in \mathbb{N}$, the language* ISOMORPHISM-TW-k*, which contains exactly the pairs of isomorphic graphs of tree width at most k, is complete for* L *under first-order reductions.*

For testing whether two graphs are isomorphic, it is in practice often helpful to perform a two-step approach that first computes a canonical representative for each isomorphism class, called the *canon*, and then declares the two graphs to be isomorphic exactly if their canons are equal (rather than isomorphic). To also be able to construct an *isomorphism* between the input graphs (that means, a bijective function between the vertex sets of given graphs that preserves their edge relations), it is helpful to have additionally access to an isomorphism from the input graphs to their canons. Such an isomorphism to the canon is called a *canonical labeling* of a graph. An isomorphism between the input graphs can be constructed by composing canonical labelings.

For most isomorphism algorithms that have been developed so far, it was possible, with varying amounts of extra effort, to turn them into an algorithm that computes canons and canonical labelings. Hence, deciding ISOMORPHISM and computing canons often have the same known complexity. However, the current situation for graphs of bounded tree width is different: While the approach from [6] puts the isomorphism problem for graphs of bounded tree width into SAC^1, this is not done by providing a canonization procedure. In fact, the best known upper bound for canonizing graphs of bounded tree width uses logarithmic-depth circuits with unbounded fan-in Boolean gates (that means, circuits defining the complexity class AC^1) [27]. Between these classes only the relation $SAC^1 \subseteq AC^1$ is known. Our second main result clarifies this situation by canonizing graphs of bounded tree width in logspace.

▶ **Theorem 2.** *For every $k \in \mathbb{N}$, there is a logspace-computable mapping that turns a graph G with tree width at most k into an isomorphism-invariant encoding of G (a canon) and an isomorphism to it (a canonical labeling).*

Techniques

The known logspace approaches for canonizing certain classes of bounded tree width graphs are based on first computing an isomorphism-invariant tree decomposition for the given input graph and then adjusting Lindell's tree canonization approach to canonize the graph with respect to the decomposition. For example, for k-trees [1] an isomorphism-invariant tree

decomposition arises by taking a graph's maximal cliques and their size-k intersections as the bags of the decomposition and connecting two bags based on inclusion. The resulting tree decomposition is both isomorphism-invariant, which is required for a canonization procedure to be correct, and has width k, which enables the application of an extension of Lindell's approach by taking (the constant number of) orderings of the vertices of the bags into account.

Technique 1: Isomorphism-invariant tree decomposition into bags without clique separators. In general, for graphs of tree width at most k, there is no isomorphism-invariant tree decomposition of width k. A simple example of graphs demonstrating this are cycles, which have tree width 2, but no isomorphism-invariant tree decomposition of width 2. We could hope to find an isomorphism-invariant tree decomposition by allowing approximate tree decompositions (that means, allowing an increase of the width to some constant k'). Again, cycles show that such tree decompositions do not always exist. To address this issue, we could consider not just one tree decomposition, but an isomorphism-invariant and polynomial-size collection of tree decompositions. However, for all $k' \in \mathbb{N}$, there are graphs of tree width at most 3 for which the smallest isomorphism-invariant collection of tree decompositions of width k' has exponential size. Simple graphs demonstrating this fact are given by forming the disjoint union of n cycles of length n, and adding a vertex that is adjacent to every other vertex.

We work around this problem by considering isomorphism-invariant tree decompositions that may have bags of unbounded size, but with bags that are easier from a graph-theoretic and algorithmic perspective than the original graph. An algorithm developed recently [19] (which refined the time complexity for ISOMORPHISM on graphs of tree width k from Bodlaender's $n^{O(k)}$ bound to $g(k) \cdot n^{O(1)}$ for a function g) applies a technique from Leimer [17] that turns the input graph into its isomorphism-invariant collection of maximal induced subgraphs without clique separators called *maximal atoms*. (In the example above, the maximal atoms are exactly the enriched cycles.) Conceptually, the first step of the proofs of our main results is similar, but produces isomorphism-invariant tree decompositions into bags that are maximal atoms instead of just the isomorphism-invariant collection whose arrangement as a tree highly depends on the order in which subgraphs are considered. While it is sufficient to have an isomorphism-invariant set of potential bags capturing a tree decomposition in order to perform polynomial-time isomorphism tests (see [22]), in order to apply or work towards logspace techniques it is necessary to have an isomorphism-invariant tree decomposition. Our first main technical contribution consists of the graph-theoretic concepts and algorithmic ideas that are needed to compute isomorphism-invariant tree decompositions into maximal atoms for graphs of bounded tree width.

Technique 2: Nested tree decomposition and a quasi-complete isomorphism-based ordering. Lindell's approach [18] for canonizing trees is based on using a weak order on the class of all trees whose incomparable elements are exactly the isomorphic ones, and showing that the order can be computed in logspace. Das, Torán, and Wagner [6] extended this to also work for graphs with respect to given tree decompositions of bounded width. This is done by adding the idea that, for bounded width, it is possible in logspace to guess partial isomorphisms between bags and recursively check whether they can be extended to isomorphisms between the whole graphs and the tree decompositions. When working with the tree decompositions into maximal atoms described above, it is not possible to just guess and check partial isomorphisms between bags since they have an unbounded width.

In order to handle the width-unbounded bags of the above decomposition, we use the fact that (as shown in [19]), after appropriate preprocessing, the maximal atoms have polynomial-size isomorphism-invariant families of approximate tree decompositions. To compute these families, we combine an approach for constructing separator-based tree decompositions from [9] to work with the isomorphism-invariant separators from [19]. If we choose a bounded width tree decomposition for each atom, and replace each atom by the chosen tree decomposition, we can turn the width-unbounded decomposition into a width-bounded decomposition for the whole graph. However, since each maximal atom may be associated with several decompositions, we need to consider for each atom a family of decompositions. We call the structure that is obtained a *nested tree decomposition*. In order to extend the approach that canonizes with respect to width-bounded decompositions to nested tree decompositions, we incorporate a bag refinement step into the weak ordering. It turns root bags of unbounded width into width-bounded tree decompositions. For each candidate tree decomposition of the root bag this triggers a modification of the original tree decomposition. However, it turns out that determining whether there is an isomorphism between two graphs that respects two given nested tree decompositions is as hard as the general graph isomorphism problem. Having a polynomial-time algorithm for this, let alone a logspace algorithm, would thus put the general graph isomorphism problem into P. Consequently, we do not generalize the idea of using isomorphism-based orderings with respect to decompositions in a direct way to nested tree decompositions. Instead, we define an approximation of the isomorphism-based ordering. This approximation has the property that it is isomorphism-invariant (that means, graphs that are isomorphic with respect to given nested decompositions are incomparable) but is only quasi-complete, by which we mean that graphs that are incomparable must be isomorphic but not necessarily via an isomorphism that respects the nested decompositions. Developing the notion of nested tree decompositions along with just the right notion of a quasi-complete isomorphism-based ordering is our second main technical contribution.

Technique 3: Recursive logspace algorithm implementing the quasi-complete ordering. Trying all choices of a decomposition on all of the atoms yields exponentially many refined decompositions in total. Avoiding this exponential blowup, our third main technical contribution is a dynamic-programming approach along the tree decomposition that shows how to cycle through candidate decompositions of the maximal atoms while, still, canonizing the graph along the coarser tree decomposition in logspace.

Since recursively cycling through tree decompositions of a bag needs space, we cannot just use the polynomial-size family of tree decompositions that we get from applying the results of [9] to those of [19] as described above. In order to implement the recursion in logspace, we compute nested tree decompositions that satisfy a certain additional property, which we call *p-boundedness*. It allows us to maintain a trade-off between the number of candidate tree decompositions chosen for each bag and the size of the subdecomposition sitting below the bag. This makes a recursive algorithm that uses only logarithmic space possible.

Organization. Section 2 provides background on graphs and logspace. The remaining paper is structured along the proofs of the theorems: Section 3 shows how to compute isomorphism-invariant decompositions into clique-separator-free graphs, while Section 4 contains the decomposition approach for graphs without clique separators. Section 5 defines the notion of nested decompositions and a weak ordering defined along them, while Section 6 proves that the ordering is logspace-computable for width-bounded and p-bounded decompositions.

Section 7 proves the main theorems and Section 8 concludes the paper. Due to lack of space, proofs are often sketched or omitted; see the paper's preprint for details [11].

2 Background

The present section sketches the paper's background on graphs, the isomorphism problem, and logspace.

We denote the set of natural numbers, which start at 0, by $\mathbb{N}$, and use shorthands $[n,m] := \{n,\dots,m\}$ and $[m] := [1,m]$ for every $n \in \mathbb{N}$ and $m \in \mathbb{N} \setminus \{0\}$.

For a graph $G = (V,E)$ with *vertices* V and *edges* $E \subseteq V \times V$, we define $V(G) := V$ and $E(G) := E$. All graphs considered in the present paper are finite, undirected and simple (neither parallel edges nor loops are present). We denote the class of all finite graphs by $\mathcal{G}$. To simplify later definitions, we define the *coloring function* $\mathrm{col}_G \colon V(G) \times V(G) \to \mathbb{Z}$ of a graph G as follows. $\mathrm{col}_G(u,v)$ equals -1 if $v = w$, 1 if $v \neq w$ and $\{u,v\} \in E(G)$, and 0 if $v \neq w$ and $\{u,v\} \notin E(G)$. If G's vertices or edges are *colored*, we extend the coloring function to return natural number encodings of colors. We use standard definitions related to connectivity functions and tree decompositions. We write a tree decomposition as a tuple $D = (T,\mathcal{B})$ where T is the tree underlying the decomposition and $\mathcal{B}$ is the family of bags that implicitly defines the decomposition's adhesion sets and torsos.

An *isomorphism* from a (colored) graph G to a (colored) graph H is a bijective mapping $\varphi \colon V(G) \to V(H)$, such that $\mathrm{col}_G(u,v) = \mathrm{col}_H(\varphi(u),\varphi(v))$ holds for every $u,v \in V(G)$. Graphs G and H that admit an isomorphism between them are *isomorphic*. This gives rise to an equivalence relation that partitions $\mathcal{G}$ into *isomorphism classes*. The *graph isomorphism problem* is the language $\textsc{isomorphism} := \{(G,H) \in \mathcal{G} \times \mathcal{G} \mid G \text{ and } H \text{ are isomorphic}\}$. A mapping inv that associates an object $\mathrm{inv}(G)$ with every graph $G \in \mathcal{G}$, for example a tree decomposition or a family of tree decompositions, is *isomorphism-invariant* if for every isomorphism φ between two graphs the result of applying φ and inv is independent of the order in which they are applied. That means, for every isomorphism φ from a graph G to a graph H, replacing all occurrences of vertices $v \in V(G)$ in $\mathrm{inv}(G)$ by their image $\varphi(v)$ yields $\mathrm{inv}(H)$.

Two graphs G and G' are *isomorphic with respect to tree decompositions* $D = (T,\mathcal{B})$ and $D' = (T',\mathcal{B}')$, respectively, if there exists an isomorphism φ from G to G' and an isomorphism ψ from T to T' satisfying $B'_{\psi(n)} = \{\varphi(v) \mid v \in B_n\}$ for every node $n \in V(T)$. Under these conditions we say that φ *respects* D and D'. Based on this definition and the way of how it refines the isomorphism equivalence relation among graphs, we also consider *canons of graphs with respect to tree decompositions*.

A deterministic Turing machine whose working space is logarithmically bounded by the input length is called a *logspace* DTM. The functions $f \colon \{0,1\}^* \to \{0,1\}^*$ computed by such machines are *logspace-computable* (or *in logspace*). The complexity class L, called *(deterministic) logspace*, contains all *languages* $P \subseteq \{0,1\}^*$ whose characteristic functions are in logspace.

3 Decomposing Graphs into Parts Without Clique Separators

A *clique* is a graph with an edge between every two vertices, including the empty graph by definition. A separation (A,B) is a *clique separation* with *clique separator* $A \cap B$ in a graph G if it (1) separates two vertices $x,y \in V(G)$, and (2) $G[A \cap B]$ is a clique.

We construct isomorphism-invariant tree decompositions for graphs of bounded tree width whose bags induce subgraphs without clique separators and whose adhesion sets are

cliques (that means, the torsos are exactly the subgraphs induced by the bags). These tree decompositions serve as an intermediate decomposition step in the proofs of the main theorems.

▶ **Lemma 3.** *For every $k \in \mathbb{N}$, there is a logspace-computable and isomorphism-invariant mapping that turns a graph G with tree width at most k into a tree decomposition D for G in which* (1) *subgraphs induced by the bags do not contain clique separators, and* (2) *adhesion sets are cliques.*

The tree decomposition we construct to prove the lemma is a refined version of a decomposition of Leimer [17] of graphs into their collections of maximal induced subgraphs without clique separators. The crucial point is that we need to adjust his method to not only output the collection of maximal induced subgraphs without clique separators, which suffices for its application in [19], but also an isomorphism-invariant tree decomposition that is based on it. In order to do that, we replace the approach of [17], which is based on finding clique-separator-free parts in a single phase via computing elimination orderings, by an approach that consists of $k+1$ steps, where $k \in \mathbb{N}$ is the (constant) tree width of the input graph: Given a connected graph G, step 1 finds the maximal induced subgraphs of G that do not contain clique separators of size at most 1. Then we build a tree decomposition whose bags are the computed subgraphs without clique separators of size 1 and adhesion sets are the computed size-1 clique separators. Each following step $c \in \{2, \dots, k+1\}$ continues in a similar way: We already know (from the previous step) that the subgraphs induced by the bags do not contain clique separators of size at most $c-1$. For each bag, we find the clique separators of size at most c in its induced graph, compute a tree decomposition whose bags are the induced subgraphs without size-c clique separators and adhesion sets are size-c clique separators. In order to proceed with a single tree decomposition that satisfies the above mentioned precondition, we merge the tree decompositions for the bags into the already computed decomposition. This results in a tree decomposition whose bags induce graphs without clique separators of size at most c and adhesions are clique separators of size at most c. Since graphs of tree width at most k contain only cliques of size at most $k+1$, step $k+1$ finishes with a tree decomposition whose bags induce graphs without clique separators and adhesion sets induce cliques. Implementing this approach both in an isomorphism-invariant and logspace-computable way requires to refine the classical connection between clique separators and candidate tree representations of chordal completions of a graph in terms of size-bounded clique separators and uniquely-defined tree representations of graphs that arise by a refined notion of chordal completions.

4 Decomposing Graphs Without Clique Separators

The decomposition procedure from the previous section provides us with a tree decomposition whose bags are clique-separator-free. In the present section, we decompose clique-separator-free graphs further into isomorphism-invariant tree decompositions of bounded width (formalized by Lemma 4). This needs two additional assumptions that we later meet during the proofs of the main theorems. First, the decomposition is based on two distinguished nonadjacent vertices from the graph. Second, we assume that the given graph is improved as defined next.

Let $\mathrm{impr}\colon \mathcal{G} \to \mathcal{G}$ be the mapping that takes a graph G and adds edges between all vertices $u, v \in V(G)$ with $\kappa(u,v) > \mathrm{tw}(G)$, where $\kappa(u,v)$ is the size of a smallest separator that separates u from v. The impr-operator *improves* the graph by adding edges to G

based on its tree width. To avoid losing information, we introduce a function $\mathrm{col}_{\mathrm{impr}(G)}$ that colors edges that appear originally in the inputs with a different color than those coming from the improvement. The mapping impr is isomorphism-invariant by definition. Besides this, we use three further properties of the mapping impr. First, the graph we get from applying impr is saturated in the sense that a second application of it does not add new edges. Formally, this means $\mathrm{impr}(G) = \mathrm{impr}(\mathrm{impr}(G))$ for every graph G as proved in [20, Lemma 2.5]. Second, the tree decompositions of a graph G are exactly the tree decompositions of $\mathrm{impr}(G)$. This implies $\mathrm{tw}(G) = \mathrm{tw}(\mathrm{impr}(G))$ and is proved in [20, Lemma 2.6]. Third, the mapping impr is logspace-computable for graphs of bounded tree width. This follows from Reingold's algorithm for UNDIRECTED-REACHABILITY, and the fact that the tree width of a graph bounds the size of the separators we need to consider in order to compute impr.

▶ **Lemma 4.** *For every $k \in \mathbb{N}$, there is a $k' \in \mathbb{N}$ and a logspace-computable and isomorphism-invariant mapping that turns every graph G with a distinguished non-edge $\{u, v\} \notin E(G)$, where G (1) has tree width at most k, (2) does not contain clique separators, and (3) is improved (that means, $G = \mathrm{impr}(G)$), into a width-k' tree decomposition $D = (T, \mathcal{B})$ for G.*

The construction of the decomposition is based on recursively splitting the graph into smaller subgraphs using size-bounded and isomorphism-invariant separators. In order to do this, we adapt in a first step the isomorphism-invariant separators from [19] and show their logspace-computability. Then we combine this with a logspace approach for handling the recursion involved in this approach from [9].

5 Isomorphism-Based Ordering of Nested Tree Decompositions

We develop the notion of nested tree decompositions to later combine the decomposition that we get from Lemma 3 with the candidate decompositions we get from Lemma 4. Nested tree decompositions are tree decompositions whose parts are not just bags, but where every bag is associated with a family of tree decompositions for the bag's torso. We use polynomial-size nested tree decompositions to represent exponential-size families of width-bounded tree decompositions that arise by replacing bags with tree decompositions from their families. In order to solve the isomorphism problem with the help of nested tree decompositions, we use a recursively defined weak ordering on pairs of graphs and nested tree decompositions. Incomparable elements in this weak ordering represent isomorphic graphs. In the following we first define nested tree decompositions and, then, the weak ordering for them.

A *nested (tree) decomposition* $\bar{D} = (T, \mathcal{B}, \mathcal{D})$ for a graph G consists of a tree decomposition $(T, \mathcal{B})$ for G, and a family $\mathcal{D} = (\mathcal{D}_n)_{n \in V(T)}$ where every $\mathcal{D}_n$ is a family of tree decompositions $D \in \mathcal{D}_n$ for the torso of n. Normal tree decompositions can be viewed as nested decompositions where $\mathcal{D}_n$ is empty for every $n \in V(T)$. We adjust some terminology that usually applies to tree decompositions for the use with nested decompositions. Let $\bar{D} = (T, \mathcal{B}, \mathcal{D})$ be a nested decomposition. The definition of the *width* of a bag B_n in a nested decomposition depends on whether $\mathcal{D}_n$ is empty or contains a set of tree decompositions. If $|\mathcal{D}_n| = 0$, we set $\mathrm{tw}(B_n) := |B_n| - 1$ and $\mathrm{tw}(B_n) := \max\{\mathrm{tw}(D) \mid D \in \mathcal{D}_n\}$, otherwise. The *width* of $\bar{D}$ is $\mathrm{tw}(\bar{D}) := \max\{\mathrm{tw}(B_n) \mid n \in V(T)\}$. The *size* of $\bar{D}$ is $|\bar{D}| := \sum_{n \in V(T)}(1 + \max\{|D| + 1 \mid D \in \mathcal{D}_n\})$, where $|\mathcal{D}_n| = 0$ implies $\max\{|D| + 1 \mid D \in \mathcal{D}_n\} = 0$. An *(unordered) root set* M of a nested decomposition $\bar{D} = (T, \mathcal{B}, \mathcal{D})$ is a subset $M \subseteq B_r$ of the root bag B_r of $\bar{D}$ with (1) $M = B_r$ in case $|\mathcal{D}_r| = 0$, and (2) every $D \in \mathcal{D}_r$ has a bag B with $M \subseteq B$ in case $|\mathcal{D}_r| > 0$. An *ordered root set* σ is an ordering of an unordered root set. *Refining* a nested decomposition $\bar{D} = (T, \mathcal{B}, \mathcal{D})$ with respect to a tree decomposition $D \in \mathcal{D}_r$ for the

root $r \in V(T)$ and an ordered root set σ is done as follows. First, we decompose $G[B_r]$ using D. Then, for each child bag B_c of B_r in $\bar{D}$, we find the highest bag in D that contains the adhesion set $B_{\{r,c\}} = B_r \cap B_c$ and make B_c adjacent to it. A bag of this kind exists since, by definition, D is a tree decomposition of the torso of B_r. We add a new bag containing the elements of σ. This bag is the new root of the obtained decomposition and adjacent to the highest bag in D that contains all elements of σ (in particular, this operation may change which bag of D is highest). The newly constructed nested decomposition is said to be obtained by *refining* $\bar{D}$ and denoted by $\bar{D}_{D,\sigma}$. The size of a nested decomposition decreases when it is refined. That means $|\bar{D}_{D,\sigma}| < |\bar{D}|$ holds. We use this property for proofs by induction.

To be able to distinguish original bags and bags from refining decompositions, we could mark the bags of D, which arise from the refinement step. We circumvent the need to mark the bags by assuming that the bags B_n with empty $\mathcal{D}_n$ are exactly the marked ones. In turn, we require from all nested decompositions $\bar{D}$ we consider that the set of bags B_n with empty $\mathcal{D}_n$ form a connected subtree in $\bar{D}$ containing the root.

▶ **Proposition 5.** *The mapping that turns a nested decomposition $\bar{D} = (T, \mathcal{B}, \mathcal{D})$ with decomposition $D \in \mathcal{D}_r$ and an ordered root set σ into $\bar{D}_{D,\sigma}$ is logspace-computable and isomorphism-invariant.*

In order to define the isomorphism-based ordering for nested decompositions, we start to review notions related to composed orderings and define an ordering of graphs with given vertex sequences.

Let $\prec$ be a *weak ordering* on a set M, and $a \equiv a'$ denote that two elements $a, a' \in M$ are *incomparable* with respect to $\prec$. That means, neither $a \prec a'$ nor $a' \prec a$ holds. We define the *weak ordering on sequences* from $M^* := \cup_{n\in\mathbb{N}} M^n$ with respect to $\prec$ as follows. We set $a = a_1 \dots a_s \prec a'_1 \dots a'_t = a'$ for $a, a' \in M^*$ if $s < t$, or $s = t$ and there is an $i \in [s]$ with $a_i \prec a'_i$ while $a_j \equiv a_j$ holds for every $j \in [i-1]$. The *weak ordering on tuples* from $M_1 \times \dots \times M_k$ with respect to weak orderings $\prec_i$ for sets M_i, respectively, is defined in the same way except that tuples always have the same length. We denote it by $\prec_{(1,\dots,k)}$. We define a *weak ordering on finite subsets* of M by setting $M_1 \prec M_2$ for two finite $M_1, M_2 \subseteq M$ based on comparing the sequences we get by sorting their elements to be monotonically increasing with respect to $\prec$.

We write the *concatenation* of sequences σ and τ as $\sigma\tau$. Suppose that (G, σ) and (G', σ') are pairs consisting of graphs G and G' with sequences of vertices $\sigma = v_1 \dots v_s$ and $\sigma' = v'_1 \dots v'_t$ from the respective graphs. We set $(G, \sigma) \prec_{\text{seq}} (G', \sigma')$ if sequence $\text{col}_G(v_1, v_1) \dots \text{col}_G(v_1, v_s) \text{col}_G(v_2, v_1) \dots \text{col}_G(v_s, v_1) \dots \text{col}_G(v_s, v_s)$ is smaller than sequence $\text{col}_{G'}(v'_1, v'_1) \dots \text{col}_{G'}(v'_1, v'_t) \text{col}_{G'}(v'_2, v'_1) \dots \text{col}_{G'}(v'_t, v'_1) \dots \text{col}_{G'}(v'_t, v'_t)$ with respect to the (standard) ordering $<$ of $\mathbb{N}$. We write $(G, \sigma) \equiv_{\text{seq}} (G', \sigma')$ if (G, σ) and (G', σ') are *incomparable* with respect to $\prec_{\text{seq}}$. The ordering $\prec_{\text{seq}}$ is logspace-computable by enumerating all pairs of vertices in lexicographic order of the indices.

Graphs G and G' are *isomorphic with respect to sequences of vertices* $\sigma = v_1 \dots v_s$ and $\sigma' = v'_1 \dots v'_t$ from the respective graphs if $s = t$ and there is an isomorphism φ from G to G' with $\varphi(v_i) = v'_i$ for every $i \in [s]$. We say that φ *respects* σ and σ' in this case. Based on this definition, we also consider *canons of graphs with respect to vertex sequences*. Due to the following statement, which we immediately get from the definition, we call $\prec_{\text{seq}}$ an *isomorphism-based ordering of graphs with vertex sequences*.

▶ **Proposition 6.** *Let G and G' be graphs with sequences of vertices $\sigma = v_1 \dots v_s$ and $\sigma' = v'_1 \dots v'_t$ from the respective graphs.*

- ("invariance"-property) *If G and G' are isomorphic with respect to σ and σ', then $(G, \sigma) \equiv_{\text{seq}} (G', \sigma')$.*
- ("quasi-completeness"-property) *If $(G, \sigma) \equiv_{\text{seq}} (G', \sigma')$, then the (induced subgraphs) $G[\{v_1, \dots, v_s\}]$ and $G'[\{v'_1, \dots, v'_t\}]$ are isomorphic with respect to σ and σ'.*

We define an ordering of graphs with nested decompositions by recursively ordering the child decompositions and combining this with the root bags. If a root bag has no refining tree decompositions, this is done by trying all possible orderings of the vertices of the bag. If the root bag has refining tree decompositions, this is done by first refining it before going into recursion.

For each child c of the root node r of a nested decomposition $D = (T, \mathcal{B}, \mathcal{D})$, we define a set $\Pi(c)$ of orderings of a vertex set as follows. If $|\mathcal{D}_c| = 0$, then $\Pi(c)$ contains all orderings of the vertices of B_c. If $|\mathcal{D}_c| > 0$, then $\Pi(c)$ is the set of orderings of the adhesion set $B_{\{r,c\}} = B_r \cap B_c$. We use the sequences from $\Pi(c)$ as ordered root sets for the child decomposition of $\bar{D}$ rooted at c.

For all tuples $(G, \bar{D}, \sigma)$ and $(G', \bar{D}', \sigma')$ of graphs with nested decompositions and ordered root sets, we define whether $(G, \bar{D}, \sigma) \prec_{\text{dec}} (G', \bar{D}', \sigma')$ holds based on a case distinction:

"size"-comparison. If $|\bar{D}| < |\bar{D}'|$, or $|\bar{D}| = |\bar{D}'|$ and $|\mathcal{D}_r| < |\mathcal{D}'_{r'}|$, then set $(G, \bar{D}, \sigma) \prec_{\text{dec}} (G', \bar{D}', \sigma')$.

"bag"-comparison. If $|\bar{D}| = |\bar{D}'| = 1$ (which implies $|\mathcal{D}_r| = |\mathcal{D}'_{r'}| = 0$), then set $(G, \bar{D}, \sigma) \prec_{\text{dec}} (G', \bar{D}', \sigma')$ if $(G, \sigma) \prec_{\text{seq}} (G', \sigma')$.

"recursive"-comparison. If $|\bar{D}| = |\bar{D}'| > 1$, and $|\mathcal{D}_r| = |\mathcal{D}'_{r'}| = 0$, we compare the decompositions recursively. Let $c_1, \dots, c_s$ be the children of r in $\bar{D}$ with respective child decompositions $\bar{D}_1, \dots, \bar{D}_s$ and subgraphs $G_1, \dots, G_s$. Let $c'_1, \dots, c'_t$ be the children of r' in $\bar{D}'$ with respective child decompositions $\bar{D}'_1, \dots, \bar{D}'_t$ and subgraphs $G'_1, \dots, G'_t$. Set $(G, \bar{D}, \sigma) \prec_{\text{dec}} (G', \bar{D}', \sigma')$ if the following relation holds, which compares sets of sets that contain tuples to which $\prec_{(\text{dec,seq})}$ applies directly:

$$\left\{ \{((G_i, \bar{D}_i, \tau), (G, \sigma\tau)) \mid \tau \in \Pi(c_i)\} \;\middle|\; i \in [s] \right\} \prec_{(\text{dec,seq})} \left\{ \{((G'_i, \bar{D}'_i, \tau'), (G', \sigma'\tau')) \mid \tau' \in \Pi(c'_i)\} \;\middle|\; i \in [t] \right\}.$$

"refinement"-comparison. If $|\bar{D}| = |\bar{D}'| > 1$, and $|\mathcal{D}_r| = |\mathcal{D}'_{r'}| > 0$, then set $(G, \bar{D}, \sigma) \prec_{\text{dec}} (G', \bar{D}', \sigma')$ if $\{(G, \bar{D}_{D,\sigma}, \sigma) \mid D \in \mathcal{D}_r)\} \prec_{\text{dec}} \{(G', \bar{D}'_{D',\sigma'}, \sigma') \mid D' \in \mathcal{D}'_{r'})\}$ holds.

Graphs G and G' are *isomorphic with respect to nested decompositions* $\bar{D} = (T, \mathcal{B}, \mathcal{D})$ and $\bar{D}' = (T', \mathcal{B}', \mathcal{D}')$ as well as ordered root sets σ and σ', respectively, if there exists an isomorphism φ from G to G' that (1) respects the (normal) tree decompositions $(T, \mathcal{B})$ and $(T', \mathcal{B}')$, (2) respects the sequences σ and σ', and (3) for every $n \in V(T)$ there is a bijection π_n from $\mathcal{D}_n$ to $\mathcal{D}_{n'}$, such that φ restricted to B_n respects D and $\pi(D)$ for all $D \in \mathcal{D}_n$. Based on how this definition refines the isomorphism equivalence relation among graphs, we consider *canons of graphs with respect to nested decompositions*. We call $\prec_{\text{dec}}$ an *isomorphism-based ordering of graphs with nested decompositions*, which is justified by the following lemma.

▶ **Lemma 7.** *Let $(G, \bar{D}, \sigma)$ and $(G', \bar{D}', \sigma')$ be tuples consisting of graphs with respective nested decompositions and ordered root sets.*

- ("invariance"-property) *If G and G' are isomorphic with respect to $\bar{D}$ and $\bar{D}'$ as well as σ and σ', then $(G, \bar{D}, \sigma) \equiv_{\text{dec}} (G', \bar{D}', \sigma')$.*
- ("quasi-completeness"-property) *If $(G, \bar{D}, \sigma) \equiv_{\text{dec}} (G', \bar{D}', \sigma')$, then G and G' are isomorphic with respect to σ and σ'.*

The ordering $\prec_{\text{dec}}$ is defined in order to satisfy the "quasi-completeness"-property, but not a "completeness"-property saying that $(G, \bar{D}, \sigma) \equiv_{\text{dec}} (G', \bar{D}', \sigma')$ implies that G and G' are isomorphic with respect to σ and σ' as well as $\bar{D}$ and $\bar{D}'$, too. The reason behind this lies in the fact that deciding an ordering of this kind for nested decompositions of a bounded width is as hard as (general) ISOMORPHISM. (This can be proved by a reduction that turns graphs into pairs of independent sets and nested decompositions, which encode the edges).

6 Computing the Ordering for Nested Tree Decompositions in Logspace

We now investigate methods to space-efficiently evaluate the isomorphism-based ordering described in the previous section. The nested decompositions we are working with always have a bounded width. This makes it possible to implement the "recursive"-comparison of the isomorphism-based ordering space-efficiently. If the child decompositions are small enough (more precisely, they are smaller by a constant fraction in comparison to their parent), then it is possible to store a constant amount of information, and in particular to store orderings of the size-bounded root bag, before descending into recursion, without exceeding a desired logarithmic space bound. If there is a large child decomposition, of which there can be only one, then we can use Lindell's classic technique of precomputing the recursive information before storing anything at all. However, for the "refinement"-comparison, a space-efficient approach turns out to be more challenging. In this case, the ordering asks us to compare various refinements of the root bag. Cycling through these refinements as part of a recursive approach requires too much space, even if the number of decompositions is bounded by a polynomial in the size of the root bag. While it is not clear how to remedy this difficulty in general, the nested decompositions we construct in the proofs of our main theorems satisfy an additional technical condition, called p-boundedness below. This makes it possible to find a trade-off between the recursive space requirement and the space required for cycling through the refinements.

Let $\bar{D}$ be a nested decomposition. Consider a bag n with $|\mathcal{D}_n| > 1$. Let $c_1, \ldots, c_t$ be the children of n sorted by monotonically decreasing size of the respecting subdecompositions $D_1, \ldots, D_t$. If it exists, let $j \in [t]$ be maximal such that $G[A_n]$ with $A_n := (B_n \cap B_{c_1}) \cup \cdots \cup (B_n \cap B_{c_j})$ is a clique, and $|D_j| > |D_{j+1}|$ holds or $j = t$ holds. Otherwise, set $j := 0$ and $A_n := \emptyset$. We call the children $c_1, \ldots, c_j$ of n the *special children* and A_n is the *attachment clique of the special children*. A nested decomposition $\bar{D}$ is *p-bounded* for a polynomial $p : \mathbb{N} \to \mathbb{N}$ if for every $n \in V(T)$ and non-special child c of n we have $|\mathcal{D}_n| \leq p(|\bar{D}|/|\bar{D}_c|)$. For non-special nodes we use the p-boundedness condition to trade the number of candidate refining decompositions with the size of subdecompositions. This enables an overall space-efficient recursion leading to a proof of the following lemma.

▶ **Lemma 8.** *For every $k \in \mathbb{N}$ and polynomial $p \colon \mathbb{N} \to \mathbb{N}$, there is a logspace* DTM *that, on input of graphs G and G' along with respective nested decompositions $\bar{D}$ and $\bar{D}'$ and ordered root sets σ and σ' where $\bar{D}$ and $\bar{D}'$ (1) have width at most k, and (2) are p-bounded, decides $(G, \bar{D}, \sigma) \prec_{\text{dec}} (G', \bar{D}', \sigma')$.*

7 Testing Isomorphism for and Canonizing Bounded Tree Width Graphs

We first show how to compute isomorphism-invariant width-bounded and p-bounded nested decompositions and, then, apply this to prove Theorems 1 and 2.

▶ **Lemma 9.** *For every $k \in \mathbb{N}$, there is a $k' \in \mathbb{N}$, a polynomial $p\colon \mathbb{N} \to \mathbb{N}$, and a logspace-computable and isomorphism-invariant mapping that turns every graph G of tree width at most k into a nested decomposition $\bar{D}$ for G that (1) has width at most k', and (2) is p-bounded.*

Proof. Instead of the original input graph G, we work with its improved version, which we can compute in logspace since the tree width of G is bounded. Mapping the input graph to its improved version is isomorphism-invariant and the improved version has the same tree decompositions. In the following, we denote the improved version of the input graph by G.

Let $D = (T, \mathcal{B})$ be the isomorphism-invariant tree decomposition we get from G by applying Lemma 3. Since the lemma guarantees that in D the adhesion sets are cliques, the torso of each bag is equal to the bag itself. To turn D into a nested decomposition it thus suffices to find a family of tree decompositions of width at most some constant $k' \in \mathbb{N}$ for each bag. We will apply Lemma 4 to find such a family. Since D decomposes an improved graph and the adhesion sets are cliques, every $G[B_n]$ for $n \in V(T)$ is also improved.

Thus, based on D, we construct a nested decomposition $\bar{D}$ by considering every node n of D and defining an isomorphism-invariant family $\mathcal{D}_n$ of tree decompositions of the bag B_n. If B_n has size at most $k+1$, we let the family $\mathcal{D}_n$ consist of a single tree decomposition that is just B_n. Note that by this choice, the bag B_n satisfies both the width bounded and the p-boundedness restriction (for every polynomial p with $p(i) \geq 1$ for all $i \in \mathbb{N}$). If the size of B_n exceeds $k+1$, we would like to apply Lemma 4 to further decompose B_n. However, for the lemma, we need a pair $\{u, v\} \notin E(G)$ in B_n to serve as the root of the decomposition. We cannot simply iterate over all $\{u, v\} \notin E(G)$ in B_n since the result may violate the p-boundedness condition. We proceed as follows: Let $c_1, \ldots, c_t$ be the children of n sorted by decreasing size of the respecting child decompositions $D_{c_1}, \ldots, D_{c_t}$. If it exists, let $j \in [t]$ be the maximum, such that $G[A_n]$ with $A_n := (B_n \cap B_{c_1}) \cup \cdots \cup (B_n \cap B_{c_j})$ is a clique, and $|D_{c_j}| > |D_{c_{j+1}}|$ holds or $j = t$ holds. Otherwise, set $j := 0$ and $A_n := \emptyset$. Thus, A_n is the attachment clique of the special children as defined above. We construct a collection of tree decompositions $\mathcal{D}_n$ for B_n based on whether we have $j < t$ or $j = t$. If $j < t$, let $m \geq 1$ be the largest integer with $|D_{c_{j+1}}| = |D_{c_{j+m}}|$. By construction, we can find at least one and at most $((k+1)(m+1))^2$ pairs of nonadjacent vertices $\{u, v\}$ in $G[A'_n]$ for $A'_n := A_n \cup (B_n \cap B_{c_{j+1}}) \cup \cdots \cup (B_n \cap B_{c_{j+m}})$.

We define $\mathcal{D}_n$ to be the collection of tree decompositions we obtained by applying Lemma 4 to $G[B_n]$ with pairs $\{u, v\}$ of nonadjacent vertices in $G[A'_n]$. We have $|\mathcal{D}_n| \leq ((k+1)(m+1))^2$. This set of decompositions satisfies the p-boundedness restriction with the polynomial $p(m) = ((k+1)(m+1))^2$. If $j = t$, we consider every pair of nonadjacent vertices $\{u, v\}$ in B_n. Again, for every such $\{u, v\}$, we construct a decomposition for $G[B]$ using Lemma 4. We have $1 \leq |\mathcal{D}_n| \leq |B_n|^2$ in this case, satisfying the p-boundedness condition, since B_n only has special children. Since the construction of the collections $\mathcal{D}_n$ is isomorphism-invariant, the entire construction is isomorphism-invariant. ◀

Proof of Theorem 1. Given two graphs G and G', by Lemma 9 we can compute in logarithmic space isomorphism-invariant p-bounded nested decompositions $\bar{D}$ and $\bar{D}'$. By Lemma 7, the graphs are isomorphic if and only if there exist ordered root sets σ and σ' with $(G, \bar{D}, \sigma) \equiv_{\text{dec}} (G', \bar{D}', \sigma')$. By Lemma 8, this can be checked in logarithmic space by iterating over all suitable choices of σ and σ'. The L-hardness for every positive $k \in \mathbb{N}$ follows from the L-hardness of the isomorphism problem for trees (connected graphs of tree width at most 1) proved by Jenner et al. [16]. ◀

Proof of Theorem 2. We use the isomorphism-invariant mapping from Lemma 9 to turn G into a width-bounded and p-bounded nested decomposition $\bar{D} = (T, \mathcal{B}, \mathcal{D})$. The canonical sequence of G's vertices is based on $(G, \bar{D}, \sigma)$ where σ is the empty vertex sequence. In order to compute a canonical sequence with respect to $\prec_{\text{dec}}$ in logspace, we repeatedly apply Lemma 8.

If $|\mathcal{D}_r| = 0$, let $\bar{D}_1, \ldots, \bar{D}_s$ be the child decompositions of G containing at least one vertex that is not in σ. We obtain an order on them by defining $\bar{D}_i < \bar{D}_j$ if $\{((G_i, \bar{D}_i, \tau), (G, \sigma\tau)) \mid \tau \in \Pi(c_i)\} \prec_{(\text{dec,seq})} \{((G_j, \bar{D}_j, \tau), (G, \sigma\tau)) \mid \tau \in \Pi(c_j)\}$. Ties are broken arbitrarily, for example by considering the smallest vertex in the child according to the input ordering. For each child $\bar{D}_i$ we compute an ordering $\tau_i \in \Pi(c_i)$ that minimizes $(G_i, \bar{D}_i, \tau_i)$. We recursively create a canonical sequence outputting the canonical sequence of $(G_i, \bar{D}_i, \tau_i)$ for each child in the order of children just defined. If $|\mathcal{D}_r| > 0$, we iterate over all decompositions in $\mathcal{D}_r$ choosing a tuple from $\{(G, \bar{D}_{D,\sigma}, \sigma) \mid D \in \mathcal{D}_B)\}$ that is minimal with respect to $\prec_{\text{dec}}$. Ties are, again, broken based on the input ordering. For computing the canonical sequence we continue recursively on a minimal $(G, \bar{D}_{D,\sigma}, \sigma)$ only. In order to obtain a canonical sequence, we alter the nested decomposition slightly whenever we go into the recursion using colored edges. More specifically, Lemma 9 constructs D based on two vertices u and v that form a distinguished non-edge. We insert an edge between u and v and color it with a color that does not appear in G (for example, we use -2). In other words, we set $\text{col}_G(u, v) := -2$. This modification is isomorphism-invariant based on the choice of D. The new edge is covered by a bag of D by construction. Inserting the edge only depends on D and, thus, it is stored recursively in an implicit way. The modification has the consequence that distinguished edges are preserved under isomorphism.

To prove that the sequence is canonical, we show that whenever a tie is broken arbitrarily between two options, then the two options are equivalent. There are two situations when a tie can occur: First, assume $\{((G_i, \bar{D}_i, \tau), (G, \sigma\tau)) \mid \tau \in \Pi(c_i)\} \equiv_{(\text{dec,seq})} \{((G_j, \bar{D}_j, \tau), (G, \sigma\tau)) \mid \tau' \in \Pi(c_j)\}$ for two child decompositions both containing a vertex not in σ. By Lemma 7, there is an isomorphism from the graph induced by the vertices in $\bar{D}_i$ to the graph induced by the vertices in $\bar{D}_j$ fixing σ. This extends to an automorphism of G by fixing all vertices neither in $\bar{D}_i$ nor $\bar{D}_j$. Since $\bar{D}$ is isomorphism-invariant this automorphism respects $\bar{D}$ therefore mapping $\bar{D}_i$ to $\bar{D}_j$. Second, assume $(G_i, (\bar{D}_i)_{D,\sigma}, \sigma) \equiv_{(\text{dec,seq})} ((G_j), (\bar{D}_j)_{D',\sigma}, \sigma)$. By Lemma 7, there is an isomorphism from G_i to G_j, which preserves the distinguished edge. It extends to an automorphism of G that fixes all vertices that neither appear in $(\bar{D}_i)_{D,\sigma}$ nor in $(\bar{D}_j)_{D',\sigma}$. Since $\bar{D}$ is isomorphism-invariant, this automorphism of G respects $\bar{D}$ and since the distinguished edge is preserved it maps $(\bar{D}_i)_{D,\sigma}$ to $(\bar{D}_j)_{D',\sigma}$. ◀

8 Conclusion

We showed how to canonize and compute canonical labelings for graphs of bounded tree width in logspace, and this implies that deciding isomorphic graphs and computing isomorphisms can be done in logspace for graphs of bounded tree width. For the proof we first developed a tree decomposition into clique-separator-free subgraphs that is isomorphism-invariant and logspace-computable. Then we showed how to compute, for each bag, an isomorphism-invariant family of width-bounded tree decompositions in logspace. Finally, we combined both decomposition approaches to construct nested tree decompositions and developed a recursive canonization procedure that works on nested tree decompositions.

Deciding ISOMORPHISM for graphs embeddable into the plane [7] or fixed surfaces [10] is in logspace. These graph classes can be described in terms of forbidding fixed minors,

which also holds for classes of graphs with bounded tree width. This opens up the question of whether these logspace results generalize to any class of graphs excluding fixed minors. For these classes polynomial-time ISOMORPHISM procedures are known [23]. Partial results are known for graphs that exclude the minors K_5 or $K_{3,3}$ [8].

References

1 Vikraman Arvind, Bireswar Das, Johannes Köbler, and Sebastian Kuhnert. The isomorphism problem for k-trees is complete for logspace. *Information and Computation*, 217:1–11, 2012. doi:10.1016/j.ic.2012.04.002.

2 Vikraman Arvind, Bireswar Das, and Johannes Köbler. A logspace algorithm for partial 2-tree canonization. In *Proceedings of the 3rd International Computer Science Symposium in Russia (CSR 2008)*, number 5010 in Lecture Notes in Computer Science, pages 40–51. Springer, 2008. doi:10.1007/978-3-540-79709-8_8.

3 Hans L Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *J. Algorithms*, 11(4):631–643, 1990. doi:10.1016/0196-6774(90)90013-5.

4 R. B. Boppana, J. Hastad, and S. Zachos. Does co-NP have short interactive proofs? *Inf. Process. Lett.*, 25(2):127–132, May 1987. doi:10.1016/0020-0190(87)90232-8.

5 Bireswar Das, MuraliKrishna Enduri, and I.Vinod Reddy. Logspace and FPT algorithms for graph isomorphism for subclasses of bounded tree-width graphs. In *9th International Workshop on Algorithms and Computation (WALCOM 2015)*, volume 8973 of *Lecture Notes in Computer Science*, pages 329–334. Springer, 2015. doi:10.1007/978-3-319-15612-5_30.

6 Bireswar Das, Jacobo Torán, and Fabian Wagner. Restricted space algorithms for isomorphism on bounded treewidth graphs. *Information and Computation*, 217:71–83, 2012. doi:10.1016/j.ic.2012.05.003.

7 Samir Datta, Nutan Limaye, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Planar graph isomorphism is in log-space. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity (CCC 2009)*, pages 203–214. IEEE Computer Society, 2009. doi:10.1109/CCC.2009.16.

8 Samir Datta, Prajakta Nimbhorka, Thomas Thierauf, and Fabian Wagner. Graph isomorphism for $K_{3,3}$-free and K_5-free graphs is in log-space. In *Proceedings of the 29th Annual IARCS Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2009)*, volume 4 of *LIPIcs*, pages 145–156. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2009. doi:10.4230/LIPIcs.FSTTCS.2009.2314.

9 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010)*, pages 143–152. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.21.

10 Michael Elberfeld and Ken-ichi Kawarabayashi. Embedding and canonizing graphs of bounded genus in logspace. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC 2014)*, pages 383–392, New York, NY, USA, 2014. ACM. doi:10.1145/2591796.2591865.

11 Michael Elberfeld and Pascal Schweitzer. Canonizing graphs of bounded tree width in logspace. *CoRR*, abs/1506.07810, 2015. URL: http://arxiv.org/abs/1506.07810.

12 Ion S. Filotti and Jack N. Mayer. A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC 1980)*, pages 236–243, New York, NY, USA, 1980. ACM. doi:10.1145/800141.804671.

13 Martin Grohe. Isomorphism testing for embeddable graphs through definability. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC 2000)*, pages 63–72. ACM, 2000. doi:10.1145/335305.335313.

14 Martin Grohe and Oleg Verbitsky. Testing graph isomorphism in parallel by playing a game. In *Proceedings of 33rd International Colloquium on Automata, Languages and Programming (ICALP 2006)*, pages 3–14, 2006. doi:10.1007/11786986_2.

15 J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing (STOC 1974)*, pages 172–184, New York, NY, USA, 1974. ACM. doi:10.1145/800119.803896.

16 Birgit Jenner, Johannes Köbler, Pierre McKenzie, and Jacobo Torán. Completeness results for graph isomorphism. *J. Comput. Syst. Sci.*, 66(3):549–566, 2003. doi:10.1016/S0022-0000(03)00042-4.

17 Hanns-Georg Leimer. Optimal decomposition by clique separators. *Discrete Mathematics*, 113(1–3):99–123, 1993. doi:10.1016/0012-365X(93)90510-Z.

18 Steven Lindell. A logspace algorithm for tree canonization (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC 1992)*, pages 400–404, New York, NY, USA, 1992. ACM. doi:10.1145/129712.129750.

19 Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth. In *Proceedings of the 55th IEEE Symposium on Foundations of Computer Science (FOCS 2014)*, pages 186–195. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.28.

20 Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth. *CoRR*, abs/1404.0818, 2014. URL: http://arxiv.org/abs/1404.0818.

21 Gary L. Miller. Isomorphism testing for graphs of bounded genus. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC 1980)*, pages 225–235, New York, NY, USA, 1980. ACM. doi:10.1145/800141.804670.

22 Yota Otachi and Pascal Schweitzer. Reduction techniques for graph isomorphism in the context of width parameters. In *Proceddings of the 14th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2014)*, pages 368–379, 2014. doi:10.1007/978-3-319-08404-6_32.

23 I. N. Ponomarenko. The isomorphism problem for classes of graphs closed under contraction. *Journal of Mathematical Sciences*, 55(2):1621–1643, 1991. doi:10.1007/BF01098279.

24 Uwe Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323, 1988. doi:10.1016/0022-0000(88)90010-4.

25 Robert Endre Tarjan. A V^2 algorithm for determining isomorphism of planar graphs. *Information Processing Letters*, 1(1):32–34, 1971. doi:10.1016/0020-0190(71)90019-6.

26 Jacobo Torán. On the hardness of graph isomorphism. *SIAM Journal on Computing*, 33(5):1093–1108, 2004. doi:10.1137/S009753970241096X.

27 Fabian Wagner. Graphs of bounded treewidth can be canonized in AC^1. In *Proceedings of the 6th International Computer Science Symposium in Russia (CSR 2011)*, number 6651 in Lecture Notes in Computer Science, pages 209–222. Springer, 2011. doi:10.1007/978-3-642-20712-9_16.

Preprocessing Under Uncertainty

Stefan Fafianie[1], Stefan Kratsch[2], and Vuong Anh Quyen[3]

1 **University of Bonn, Germany**
`fafianie@cs.uni-bonn.de`
2 **University of Bonn, Germany**
`kratsch@cs.uni-bonn.de`
3 **University of Bonn, Germany**
`vuong@cs.uni-bonn.de`

Abstract

In this work we study preprocessing for tractable problems when part of the input is unknown or uncertain. This comes up naturally if, e.g., the load of some machines or the congestion of some roads is not known far enough in advance, or if we have to regularly solve a problem over instances that are largely similar, e.g., daily airport scheduling with few charter flights. Unlike robust optimization, which also studies settings like this, our goal lies not in computing solutions that are (approximately) good for every instantiation. Rather, we seek to preprocess the known parts of the input, to speed up finding an optimal solution once the missing data is known.

We present efficient algorithms that given an instance with partially uncertain input generate an instance of size polynomial in the amount of uncertain data that is equivalent for every instantiation of the unknown part. Concretely, we obtain such algorithms for MINIMUM SPANNING TREE, MINIMUM WEIGHT MATROID BASIS, and MAXIMUM CARDINALITY BIPARTITE MATCHING, where respectively the weight of edges, weight of elements, and the availability of vertices is unknown for part of the input. Furthermore, we show that there are tractable problems, such as SMALL CONNECTED VERTEX COVER, for which one cannot hope to obtain similar results.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases preprocessing, uncertainty, spanning trees, matroids, matchings

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.33

1 Introduction

In many applications we are faced with inputs that are partially uncertain or incomplete. For example, a crucial part of the input, like availability of particular machines or current congestion of some network or road links, may only be available at short notice and may be subject to frequent change. Similarly, we may have to regularly solve instances of some problem that are very similar except for small modifications, e.g., airport gate scheduling when there are only few irregular flights.

A natural approach to this is to come up with solutions that are robust in the sense that they are close to optimal no matter what instantiation the unknown or uncertain part takes. Intuitively it is clear that one cannot hope to always find a solution that is optimal for all instantiations since then the missing parts would always need to be irrelevant. Similarly, if one has to commit to some solution containing uncertain weights/values, then changing these values can in general rule out any good ratio of robustness.

To avoid this issue, in the present work, when given an instance with missing or uncertain information we do not seek to already commit to a solution but to determine how much of the input we can solve or preprocess without knowing the missing or uncertain parts. In


© Stefan Fafianie, Stefan Kratsch, and Vuong A. Quyen;
licensed under Creative Commons License CC-BY
33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 33; pp. 33:1–33:13

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


particular, this approach permits us to still perform computations once the entire input is known/certain. Thus, there is no general argument that would rule out the possibility of finding optimal solutions since we could always do nothing and just keep the instance as is.

Our question is rather, assuming that we always want to get an optimal solution, how much of the certain part of the input do we need to keep (including any other derived information that we could choose to compute). Clearly, we should expect that increasing the amount of uncertain data should drive up the amount of information that we need to keep. Conversely, in many settings the instantiations of some k bits (e.g. presence of certain k edges in a graph) "only" create 2^k different possible instances. Thus, size exponential in the amount of uncertain data is likely to be easy to achieve, e.g., by hardwiring optimal solutions for all instantiations. In contrast, we are interested in spending only polynomial time (too little to precompute an exponential number of solutions) and preprocessing to a size that is polynomial in the amount of uncertain data.

As an easy positive example, consider a road network modeled by a graph $G = (V, E)$ with weights $w\colon E \to \mathbb{R}_{\geq 0}$ capturing the time to travel along the corresponding road. If all weights are given/certain, then we can easily compute a shortest s,t-path. If, say, weights for edges in some set $F \subseteq E$ are not known (yet) or if they are subject to change (e.g. by road congestion) then we cannot for sure determine a shortest path. Moreover, we cannot in general find a path that will be within any bounded factor of the shortest path: If we have to pick one of two parallel edges, then letting the other one have cost ϵ and ours have cost 1 gives ratio $1/\epsilon$ for arbitrary small ϵ. Preprocessing for this setting, however, is straightforward: The final shortest path will consist in some arbitrary way of edges in F and shortest u,v-subpaths containing no edge of F for $u, v \in \{s, t\} \cup V(F)$. The required u,v-paths can be precomputed by taking shortest paths in $G - F$. All distance information can be stored in a smaller graph on vertex set $\{s, t\} \cup V(F)$ by letting weight of $\{u, v\}$ be equal to the length of a shortest u,v-path in $G - F$. The edges of F are then additional parallel edges and the actual shortest s,t-path can be computed once their weights are known. (One may label edges $\{u, v\}$ by the interior vertices of the shortest u,v-path to quickly extract a shortest s,t-path.) Thus, instead of having to find a shortest path in $G = (V, E)$ once all weights are known it suffices to solve the problem on a graph with at most $2 + 2|F|$ vertices.

Our results. We study similar preprocessing questions for several fundamental problems, namely MINIMUM SPANNING TREE, MINIMUM WEIGHT MATROID BASIS, and MAXIMUM CARDINALITY BIPARTITE MATCHING. In the first two problems, the uncertainty lies in the weight of some of the edges of the input graph respectively elements of the ground set of the matroid. The matroid basis problem of course generalizes the MINIMUM SPANNING TREE question but the latter is probably more accessible and uses essentially the same ideas. For MAXIMUM CARDINALITY BIPARTITE MATCHING we study the setting that in the given bipartite graph $G = (L, R; E)$ there are sets of vertices $L_0 \subseteq L$ and $R_0 \subseteq R$ some of which will not be available (but we do not know yet). To some extent, the latter problem can also be handled by the result for MINIMUM WEIGHT MATROID BASIS, but the output would not be an instance of bipartite matching (see end of Section 4). For all three problems, we give efficient algorithms that derive an appropriate form of equivalent instance such that an optimal solution can be found using just this instance plus the missing input data. Finally, we show that there are problems for which we cannot find efficient compressions that capture all possible scenarios, even if the running time of the algorithm is allowed to be unbounded; these are SMALL CONNECTED VERTEX COVER and some LP-related problems.

Related work. The effect of uncertainty in instances on optimal solutions has long been considered and there are several approaches to deal with it. An early approach is *stochastic optimization* (SO), starting at least from Dantzig's original paper [4], which assumes that the uncertainty has a probabilistic description. A more recent approach to optimization under uncertainty is *robust optimization* (RO). In contrast to SO, the uncertainty model in RO is not stochastic, but rather deterministic and set-based. More precisely, in RO, we want to find a solution to optimize the value of a *certain* function which subjects to a list of *uncertain* constraints. Each of these constraints may depend on some uncertain parameters whose values are in a given domain which may be infinitely large or continuous. The solution is required to satisfy every constraint for *all* values of its uncertain parameters. We refer interested readers to a survey [2] for more information about RO. Both of the two approaches (SO and RO) only work with the uncertainty of the *value* of parameters in an instance, but neither with the uncertainty of *appearance* of any factor in an instance (variables, constraints, vertices, edges, etc.), nor with the uncertainty of the objective function. Moreover, the focus is on finding optimal or approximate solutions rather than preprocessing.

Concerning preprocessing, a related concept is that of *kernelization* from parameterized complexity. In brief, a kernelization for a *decision* problem is an efficient algorithm that compresses each input instance to an instance with smaller size (if possible) and ensures that the answer does not change. Note that the target of this approach is NP-hard problems and sizes of compressed instances are measured by some problem-specific parameter instead of the input size since one cannot hope to efficiently shrink all inputs of some NP-hard problem, unless $\mathsf{P} = \mathsf{NP}$. To the best of our knowledge there has not been much research in this area regarding uncertainty or robustness. Two recent results on kernelization nevertheless use intermediate results that are in line with the present work, and that have in part inspired it:

(1) A nice result of Pilipczuk et al. [13] is the following: Given a plane graph G with outer face B, one can efficiently compress the inner part of G to obtain a smaller graph H such that H contains an optimal Steiner tree connecting terminal set S for *every* subset $S \subseteq B$ of the outer face. Note that for any fixed set $S \subseteq B$ this is a polynomial-time problem, but there are of course $2^{|B|}$ many possible sets S. Pilipczuk et al. use this result to obtain polynomial kernels for several problems on planar graphs, e.g., PLANAR STEINER TREE.

(2) Kratsch and Wahlström [8] obtained the following result on cut-covering sets: Given a graph and two vertex sets S and T, there exists a small vertex set Z such that Z contains a minimum vertex (A, B)-cut for *every* $A \subseteq S$ and $B \subseteq T$, moreover Z can be computed in randomized polynomial time with small error probability. Again, for each choice of A and B this is polynomial-time solvable, but there is an exponential number of choices. We will make use of this result in Section 5 and, furthermore, it directly yields another positive example for the MIN CUT problem with uncertain vertices: Given a graph $G = (V, E)$ with two vertices $s, t \in V$ and $U \subseteq V \setminus \{s, t\}$, we can apply the result for $S = U \cup \{s\}$ and $T = U \cup \{t\}$ to compute a cut-covering set Z. Now, for every $U' \subseteq U$, the set Z contains a minimum $U' \cup \{s\}$, $U' \cup \{t\}$ cut C. Clearly, $U' \subseteq C$ and thus $C \setminus U'$ must be a minimum (s, t)-cut in $G - U'$. This observation together with a technique called *torso operation* (see [10]), which will be explained in Section 5, allows us to compress the certain part of G efficiently. By Menger's theorem, this carries over to the problem of computing the maximum number of vertex-disjoint paths between two vertices in a graph. Thus, one can also obtain a preprocessing for MAX FLOW when all capacities are small, in the sense that the guaranteed size depends polynomially on the maximum capacity.

(3) In [1] Assadi et al. also considered MAXIMUM MATCHING and MINIMUM SPANNING TREE with a similar approach but their model, which they called "The dynamic sketching

model", has two main differences: First, it only captures the uncertainty of *appearance* of edges but not their weights. Second, output of an algorithm in the model may be only a compact structure ("a sketch" in their words) but not an instance of the considered problem.

Other somewhat related paradigms are *online algorithms* and *dynamic algorithms*. In the former, the input is revealed piece-by-piece and the algorithm needs to commit to decisions without knowing the remaining input; the goal is to optimize the ratio between the online solution and an offline optimum. This is quite different from our setting because it requires to commit to a solution. In the dynamic setting a complete instance is given but modifications to it are given in further rounds; the goal is to adapt quickly to the modifications and to find a solution for the modified instance (faster than computing from scratch). This is closer to our setting, by allowing a different solution in each round, but differs by having a complete input in each round and not necessarily restricting the parts of the input that may change.

The problem of finding a minimum spanning tree when edge weights are uncertain has also been explored in a different setting [7, 11]. In this case, all edge weights fall in prespecified intervals and there is a cost for finding out the weight of a specific edge. The goal is to find a cost-efficient query strategy for finding a minimum spanning tree.

Organization. We will start with preliminaries in Section 2 and consider MINIMUM SPANNING TREE in Section 3 as a warm-up. We present our algorithms for MINIMUM WEIGHT MATROID BASIS and MAXIMUM CARDINALITY BIPARTITE MATCHING in Sections 4 and 5 respectively. Finally, we present our lower bounds in Section 6. Proofs omitted in this extended abstract can be found in [6].

2 Preliminaries

Graphs. We mostly follow graph notation as given by Diestel [5]. A *walk* in a graph G is a sequence of vertices $(v_0, v_1, \ldots, v_k)$ such that for every $i = 0, \ldots, k-1$ the vertices v_i and v_{i+1} are adjacent; if G is a directed graph then it is required that there is an arc directed from v_i to v_{i+1}. A *path* in G is a walk $(v_0, v_1, \ldots, v_k)$ such that v_i and v_j are distinct for every $i \neq j$. In this case, v_0 and v_k are called the first vertex and the last vertex of the path respectively; all other vertices are called *internal vertices*. A (u, v)*-path* is a path whose first vertex is u and whose last vertex is v. If S is a vertex set of a graph G, then we denote by $G - S$ the graph obtained from G by removing all vertices in S and their incident edges.

Matroids. *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 $B \cup \{e\} \in \mathcal{I}$; this is called the *augmentation property*.

By the augmentation property, all (inclusion-wise) maximal independent sets have the same cardinality; each of them is called *a basis*. The (inclusion-wise) minimal dependent sets are called *circuits*. 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 deleting elements in F, i.e., the matroid on ground set $E \setminus F$ whose independent sets are the independent sets of $\mathcal{M}$ that are disjoint from F. If F is an independent set of $\mathcal{M}$ then we denote by $\mathcal{M}/F$ the matroid obtained from $\mathcal{M}$ by contracting F, i.e., the matroid on ground set $E \setminus F$ such that a set I is independent if and only if $I \cup F$ is an independent set of $\mathcal{M}$. A matroid $\mathcal{M}'$ obtained from $\mathcal{M}$ by a sequence of deletion and contraction operations is called a *minor* of $\mathcal{M}$.

A matrix M over a field gives rise to a matroid $\mathcal{M}$ whose ground set is the set of columns of M and a set of columns is an independent set of $\mathcal{M}$ if and only if it is linearly independent as a set of vectors. In this case, we say that $\mathcal{M}$ is *represented by* M or M is a *representation matrix* of $\mathcal{M}$. Note that there may be different representation matrices of the same matroid and there exist matroids which cannot be represented over any field.

Because there can be as many as $2^{|E|}$ independent sets in a matroid $\mathcal{M} = (E, \mathcal{I})$, to achieve time polynomial in $|E|$ it is necessary to use a more succinct representation, rather than listing all sets in $\mathcal{I}$ explicitly. Two common ways are representing $\mathcal{M}$ by a matrix (not possible for all matroids) or assuming that an independence oracle for $\mathcal{I}$ is provided:

(i) If our matroid is given by a representation matrix over some field, the output of our algorithm should again be a representation matrix. It is known that a representation for any minor of $\mathcal{M}$ can be computed in polynomial time from a representation of $\mathcal{M}$ (cf. Marx [9]).

(ii) If our matroid is given by a ground set and an independence oracle, i.e., a blackbox algorithm which tells us whether an arbitrary subset of the ground set is independent or not, then the time for the oracle is not taken into account. In this case, the output should again be a ground set together with an oracle. Since the oracle is blackbox, the output oracle will be a frontend to the initial oracle and make queries to it.

Further notation. Given two functions $f_1 \colon X_1 \to \mathbb{N}$ and $f_2 \colon X_2 \to \mathbb{N}$ with $X_1 \cap X_2 = \emptyset$, the *union* of f_1 and f_2, denoted by $f_1 \cup f_2$, is the function $f \colon X_1 \cup X_2 \to \mathbb{N}$ defined by $f(x) = f_i(x)$ if $x \in X_i$ for $i = 1, 2$. For convenience, we use $+$ and $-$ instead of $\cup$ and $\setminus$ for singleton sets, e.g., $S + e$ and $S - e$ instead of $S \cup \{e\}$ and $S \setminus \{e\}$. We also abuse notation by using $f(e)$ to represent a function f whose domain is $\{e\}$, e.g., we write $f(e) \cup g$ to clarify that we take the union of two functions f and g where the function f has a singleton domain.

3 Minimum Spanning Tree in Graphs With Some Unknown Weights

For a connected graph $G = (V, E)$ and a weight function $\omega \colon E \to \mathbb{N}$ one can efficiently compute a spanning tree of minimum total edge weight. If, however, the weight of some edges is not known (yet), then in general we cannot (yet) solve the instance. Nevertheless, we may be able to preprocess the known part of the instance in order to save time later.

Say we are given a connected graph $G = (V, E \cup F)$ and a weight function $\omega_E \colon E \to \mathbb{N}$. Over different choices of weights $\omega_F \colon F \to \mathbb{N}$ for edges in F there may be an exponential number of different minimum weight spanning trees. We will show how to efficiently generate a new instance on which we may solve the problem for any instantiation of ω_F, i.e., computing the weight of a minimum spanning tree relative to weights $\omega = \omega_E \cup \omega_F$.

In slight abuse of notation we assume that edges that are originally in F can be identified in the new instance after the operations (edge contractions) performed in our algorithm. That is, given an edge in F, we can find the corresponding edge in the new instance even if the endpoints of this edge have changed. More formally this could also be captured by an appropriate bijection from F to a set of edges F' that appear in the new instance.

▶ **Theorem 1.** *There is a polynomial-time algorithm that, given a connected graph $G = (V, E \cup F)$ and a weight function $\omega_E \colon E \to \mathbb{N}$, computes a connected graph $G' = (V', E' \cup F)$ with $|E'| \leq |F|$, a weight function $\omega_{E'} \colon E' \to \mathbb{N}$, and $k \in \mathbb{N}$, such that, for any $\omega_F \colon F \to \mathbb{N}$, the graph G has minimum spanning tree weight l relative to $\omega_E \cup \omega_F \colon E \cup F \to \mathbb{N}$ if and only if G' has minimum spanning tree weight $l' = l - k$ relative to $\omega_{E'} \cup \omega_F \colon E' \cup F \to \mathbb{N}$.*

Let MSF denote a minimum weight spanning forest of $G - F$ and let $G_1 = (V, \mathrm{MSF} \cup F)$.

The following lemma shows that the weight of a minimum spanning tree in G_1 is equal to that of a minimum spanning tree in G for any weight function on F.

▶ **Lemma 2.** *For any weight function $\omega_F\colon F \to \mathbb{N}$, there is a minimum spanning tree MST_{ω_F} in $(G, \omega_E \cup \omega_F)$ such that $\mathrm{MST}_{\omega_F} \subseteq \mathrm{MSF} \cup F$.*

Accordingly, the first part of the simplification of (G, ω_E) consists of replacing G by G_1 and restricting ω_E to the edges of G_1, i.e., to the edges of $\mathrm{MSF} \cup F$.

Let $\omega_0\colon F \to \mathbb{N}\colon f \mapsto 0$. If an edge $e \in E$ is used by a minimum spanning tree for $(G, \omega_E \cup \omega_0)$ in which all edges in F have zero weight, then, intuitively, using e is also a good choice for spanning trees with other weight functions $\omega_F\colon F \to \mathbb{N}$.

▶ **Lemma 3.** *Let MST_{ω_0} a minimum spanning tree for $(G, \omega_E \cup \omega_0)$. For every $\omega_F\colon F \to \mathbb{N}$, there is a minimum spanning tree MST_{ω_F} for $(G, \omega_E \cup \omega_F)$ that uses all edges in $\mathrm{MST}_{\omega_0} \setminus F$.*

It follows that we can further simplify G_1 by contracting edges of $\mathrm{MST}_{\omega_0} \setminus F$, since for any weight function $\omega_F\colon F \to \mathbb{N}$ there is a minimum spanning tree that uses these edges.

▶ **Lemma 4.** *Let MST_{ω_0} be a minimum spanning tree in $(G, \omega_E \cup \omega_0)$ and let $e \in \mathrm{MST}_{\omega_0} \setminus F$. Let the graph $G_2 = (V', E')$ be obtained by contracting e, where $E' = E - e$, and let $\omega_{E'}$ be ω_E restricted to E'. For any weight function $\omega_F\colon F \to \mathbb{N}$, $(G, \omega_E \cup \omega_F)$ has a spanning tree with weight l if and only if $(G_2, \omega_{E'} \cup \omega_F)$ has a spanning tree MST'_{ω_F} of weight $l - \omega_E(e)$.*

We compute a minimum spanning tree MST_{ω_0} for $(G_1, \omega_E \cup \omega_0)$ and create a graph $G' = (V', E' \cup F')$ by contracting every edge in $\mathrm{MST}_{\omega_0} \setminus F$. Let k be the combined weight of the contracted edges. All edges in E' correspond to edges E after the contractions. We define ω'_E accordingly. Note that $|E'| \leq |F|$ since at most $|\mathrm{MSF}| \leq n - 1$ edges of E remain in G_1, of which we contract at least $|\mathrm{MST}_{\omega_0} \setminus F| \geq |\mathrm{MST}_{\omega_0}| - |F| = n - 1 - |F|$ edges in order to obtain G'. As a result of Lemmas 2 through 4 we have that G', ω'_E, and k satisfy the required properties in Theorem 1 and the result follows.

4 Minimum Weight Basis in Matroids With Some Unknown Weights

Given a matroid $\mathcal{M} = (E, \mathcal{I})$, we know that for each fixed weight function $w\colon E \to \mathbb{N}$ we can find a minimum weight basis of $\mathcal{M}$ in polynomial time by the greedy algorithm. Now suppose that there is a subset $F \subseteq E$ of elements with unknown weights, i.e., we are only given a partial weight function $w\colon E \setminus F \to \mathbb{N}$. We want to reduce the known part of the input such that when given any weights for elements in F we can compute a minimum weight basis.

In the rest of this section, we prove the following result:

▶ **Theorem 5.** *There is a polynomial-time algorithm that given a matroid $\mathcal{M} = (E, \mathcal{I})$, by matrix representation or independence oracle, together with a set $F \subseteq E$ and a partial weight function $w\colon E \setminus F \to \mathbb{N}$, outputs a matroid $\mathcal{M}' = (E', \mathcal{I}')$, a partial weight function $w'\colon E' \setminus F \to \mathbb{N}$, and a number $k \in \mathbb{N}$ such that: (1) E' contains F and has at most $2|F|$ elements. (2) For every weight function on F, the matroid $\mathcal{M}$ has a minimum weight basis of weight l if and only if $\mathcal{M}'$ has a minimum weight basis of weight $l - k$.*

We start by recalling some basics about the interplay of circuits and bases in a matroid.

▶ **Lemma 6** (cf. Oxley [12, Corollary 1.2.6])**.** *Let $\mathcal{M} = (E, \mathcal{I})$ a matroid and $B \in \mathcal{I}$ a basis of $\mathcal{M}$. For every $e \in E \setminus B$ there is a unique circuit C in $B + e$, and this circuit contains e. Moreover, for every $e' \in C - e$, the set $B + e - e'$ is also a basis of $\mathcal{M}$.*

The next lemma is about the relation between minimum weight bases of a given matroid $\mathcal{M}$ and the ones in a sub-matroid $\mathcal{M}'$ of $\mathcal{M}$.

▶ **Lemma 7.** *Let $\mathcal{M} = (E, \mathcal{I})$ a matroid and let $F \subseteq E$. For every weight function, if I is a minimum weight basis of $\mathcal{M} - F$, then there is a minimum weight basis of $\mathcal{M}$ that is contained in $I \cup F$.*

Given a matroid with a non-empty subset F of elements, there are infinitely many weight functions on F. However, since we are considering a minimization problem, there is a special one: the weight function which assigns value zero to every element in F. Intuitively, because elements in F have "cheapest cost" in this case, if an element not in F appears in a minimum weight basis with respect to this weight function then it should also appear in some minimum weight basis with respect to other weight functions. The next lemma verifies this intuition.

▶ **Lemma 8.** *Let $\mathcal{M} = (E, \mathcal{I})$ a matroid, let $F \subseteq E$, and let $\hat{w} \colon E \setminus F \to \mathbb{N}$. If I_0 is a minimum weight basis of $\mathcal{M}$ subject to $w_0 = \hat{w} \cup \hat{w}_0$ where $\hat{w}_0 \colon F \to \mathbb{N} \colon f \mapsto 0$, then for every weight function $w_F \colon F \to \mathbb{N}$ there is a minimum weight basis of $\mathcal{M}$ subject to $w = \hat{w} \cup w_F$ that contains $I_0 \setminus F$.*

Now we describe our algorithm. Given a matroid $\mathcal{M} = (E, \mathcal{I})$ together with a subset $F \subseteq E$ and a partial weight function $w \colon E \setminus F \to \mathbb{N}$, we compute $\mathcal{M}''$ as follows:

1. Compute a minimum weight basis B of $\mathcal{M} - F$.
2. Compute $\mathcal{M}' = \mathcal{M}[B \cup F]$. If $\mathcal{M}$ is given by a representation matrix M, then $\mathcal{M}'$ can be represented by the matrix M' obtained from M by taking only the columns corresponding to the elements in $B \cup F$. If $\mathcal{M}$ is given by an oracle O, then an oracle O' for $\mathcal{M}'$ can be obtained easily: Given a set I, first check whether I is a subset of $B \cup F$, else return no. Query O for whether I is independent in $\mathcal{M}$ and return the answer.
3. Compute a minimum weight basis B_0 of $\mathcal{M}'$ corresponding to the weight function $w_0 \colon E \to \mathbb{N}$ with $w_0(e) = 0$ for all $e \in F$ and $w_0(e) = w(e)$ for $e \in E \setminus F$.
4. Compute $\mathcal{M}'' = \mathcal{M}'/(B_0 \setminus F)$ and $k = w(B_0 \setminus F)$. If $\mathcal{M}'$ is represented by a matrix M', then a matrix representation for $\mathcal{M}''$ can be derived from M' in polynomial time (cf. [9]). If $\mathcal{M}'$ is given by an oracle O', then an oracle O'' for $\mathcal{M}''$ can be obtained as follows: Given a set I first check whether I is a subset of the ground set of $\mathcal{M}''$, else return no. Query O' for whether $I \cup (B_0 \setminus F)$ is independent in $\mathcal{M}'$ and return the answer.

It is easy to see that our algorithm runs in polynomial time. Because B_0 is a basis in $\mathcal{M}'$ and B is an independent set in $\mathcal{M}'$, we have $B_0 \subseteq B \cup F$ and $|B| \leq |B_0|$. The number of elements of $\mathcal{M}'$ not in F is

$$|B \setminus B_0| \leq |(B \cup F) \setminus B_0| = |B \cup F| - |B_0| = |B| + |F| - |B_0| \leq |F|.$$

The final lemma, about the correctness of our algorithm, finishes the proof of Theorem 5.

▶ **Lemma 9.** *For every weight function, the minimum weight of a basis in $\mathcal{M}$ is l if and only if the minimum weight of a basis in $\mathcal{M}''$ is $l - k$.*

Our result can also be applied for the case of maximum weight basis with one additional condition: There is a fixed upper bound for all weights, i.e., we may not know weights of elements in F but we do know that they cannot be larger than some constant c. In that case, if for each element e of $\mathcal{M}$, we replace its weight $w(e)$ by $w'(e) = c - w(e)$ then for every basis I we have $w'(I) = c \cdot |I| - w(I)$. Because all bases in a matroid have the same size, a maximum weight basis with respect to the original weight function must be a minimum weight basis with respect to the new one.

Theorem 5 can be applied for several specific matroid classes. For example, Theorem 1 can be obtained by an application to the class of graphic matroids, noting that these are closed under deletion and contraction. We finish this section discussing an application for *transversal matroids*. Given a bipartite graph $G = (L, R; E)$ the transversal matroid $\mathcal{M} = (R, \mathcal{I})$ has as its independent sets exactly those subsets of R that have a matching into L (equivalently, that are the endpoints of some bipartite matching). If we assign weight 1 for every element of the matroid then the weight of a maximum weight basis in the matroid is the size of a maximum matching in the original bipartite graph. Uncertain vertices in R can be easily simulated by making their weights be uncertain and using weight 0 to mean that they are not available. Observe that we have an upper bound for uncertain weights, so by the above arguments, we can apply the result for maximum weight basis to compress our uncertain instance for maximum matching in bipartite graph. Uncertain vertices in L can be handled by giving each a private neighbor that has uncertain weight: We can set these weights very high (and adjust the target weight of the basis that we are looking for) to enforce that the corresponding uncertain vertex is used to match the private neighbor, thereby preventing a matching with other R vertices. If the vertex is available then set the weight of this private neighbor to 0. However, the output would not be an instance of bipartite matching. Unlike graphic matroids, transversal matroids are not closed under contraction (which would give the larger class of gammoids), and thus it seems unlikely that one could directly extract an appropriate graph. We address this by studying the bipartite matching problem with uncertain vertices directly in the following section, using other techniques.

5 Maximum Matching in Bipartite Graphs With Uncertain Vertices

Given a bipartite graph $G = (L, R; E)$, a maximum matching of G can be found in polynomial time. Now suppose that there are vertex subsets $L_0 \subseteq L$ and $R_0 \subseteq R$ and some arbitrary vertex sets $L' \subseteq L_0$ and $R' \subseteq R_0$ may not be available in the final input, i.e., we will be asked for a maximum matching in $G - (L' \cup R')$. Thus, there are $2^{|L_0|+|R_0|}$ possible instances. How much can we simplify and shrink G when knowing only L_0 and R_0, but not L' and R'? We show that, despite the exponential number of possible final instances, a graph G' with polynomial in $|L_0| + |R_0|$ many vertices is sufficient.

▶ **Theorem 10.** *There is a randomized polynomial-time algorithm that, given a bipartite graph $G = (L, R; E)$, $L_0 \subseteq L$, and $R_0 \subseteq R$, returns a bipartite graph G' with $\mathcal{O}((|L_0|+|R_0|)^4)$ vertices and $k \in \mathbb{N}$ such that for any $L' \subseteq L_0$ and $R' \subseteq R_0$, the graph $G - (L' \cup R')$ has maximum matching size l if and only if $G' - (L' \cup R')$ has maximum matching size $l' = l - k$.*

Let us first recall the concept of *augmenting paths*.

▶ **Definition 11.** Let M a matching in a G. An M-augmenting path is a path in G s.t.
(i) the first and last vertices of the path are not incident to any edge in M and
(ii) edges on the path are alternatingly in M and not in M.

It is well known that if an augmenting path P exists, then we can obtain a matching from M that has one more edge by replacing in M the matched edges on P with the non-matched edges on P. This extends in a natural way to packings of vertex-disjoint augmenting paths.

▶ **Lemma 12.** *Let G be a graph, let M be any matching in G, let M_0 be a maximum matching in G, and let r denote the maximum number of vertex-disjoint M-augmenting paths in G. We have that $r = |M_0| - |M|$.*

We now fix a maximum matching M in $G - (L_0 \cup R_0)$ and use it in the remainder of the section. We direct the edges of G to obtain a directed bipartite graph $H = (L, R; A)$ as follows: Every edge in M is directed from R to L and every edge not in M is directed from L to R. This type of directed graph is part of a folklore approach for finding augmenting paths. Let F_L (resp. F_R) denote the vertices of $L \setminus L_0$ (resp. $R \setminus R_0$) which are not covered by M, and note that $V(M)$, L_0, R_0, F_L, and F_R are pairwise disjoint.

▶ **Observation 1.** For any $L' \subseteq L_0$ and $R' \subseteq R_0$, there is a one-to-one correspondence between directed paths in $H - (L' \cup R')$ from $F_L \cup (L_0 \setminus L')$ to $F_R \cup (R_0 \setminus R')$ and M-augmenting paths in $G - (L' \cup R')$. This is because $F_L \cup (L_0 \setminus L')$ and $F_R \cup (R_0 \setminus R')$ are exactly the vertices that are free in the matching, while a directed path must visit matched edges alternatingly, since these are exactly the edges from R to L.

▶ **Observation 2.** For any $L' \subseteq L_0$ and $R' \subseteq R_0$, there is no M-augmenting path in $G - (L' \cup R')$ that starts in F_L and ends in F_R. This follows from maximality of M, since such a path could be used to obtain a bigger matching in $G - (L_0 \cup R_0)$. By Observation 1 we have that there is no directed path in $H - (L' \cup R')$ from F_L to F_R.

Given the relation between augmenting paths and directed paths we now consider minimum cuts in the graph H. The following theorem of Kratsch and Wahlström [8] provides a small set of vertices that contains minimum cuts for a specified type of requested A, B-cuts.

▶ **Theorem 13** (Kratsch and Wahlström [8])**.** *Let $G = (V, E)$ be a directed graph and let $S, T \subseteq V$. Let r denote the size of a minimum (S, T)-vertex cut (which may intersect S and T). There exists a set $X \subseteq V$ of size $|X| = \mathcal{O}(|S| \cdot |T| \cdot r)$ such that for any $A \subseteq S$ and $B \subseteq T$ the set X contains a minimum (A, B)-vertex cut. Such a set X can be computed in randomized polynomial time with error probability $\mathcal{O}(2^{-n})$.*

The following lemma adapts Theorem 13 to our application, mainly taking care not to inflate the cut size overly much. (We ask for minimum $(F_L \cup (L_0 \setminus L'), F_R \cup (R_0 \setminus R'))$-vertex cuts, but without having size $\Omega(|F_L| + |F_R|)$.)

▶ **Lemma 14.** *There exists a set $X \subseteq L \cup R$ of size $|X| = \mathcal{O}((|L_0| + |R_0|)^3)$ such that for any $L' \subseteq L_0$ and $R' \subseteq R_0$, X contains a minimum $(F_L \cup (L_0 \setminus L'), F_R \cup (R_0 \setminus R'))$-vertex cut in $H - (L' \cup R')$; it can be found in randomized polynomial time with error probability $\mathcal{O}(2^{-n})$.*

▶ **Definition 15.** Let $D = (V, A)$ a directed graph and $Z \subseteq V$. By applying to D the *torso operation* on Z we mean to derive a new graph, denoted by $\text{torso}(D, Z)$, by adding to $D[Z]$ an arc (u, v) for every pair $u, v \in Z$ if there is a directed (u, v)-path in D with no internal vertices from Z. If an arc of $\text{torso}(D, Z)$ is not in $D[Z]$, then we call it a *shortcut arc*.

We now construct a set Z, starting from X as obtained from Lemma 14: Let M_X be the set of edges in M with at least one endpoint in X and let $X' = X \cup L_0 \cup R_0 \cup V(M_X)$; note that $X' \cap (F_L \cup F_R) = \emptyset$. For each $v \in X' \cap R$ (resp. $v \in X' \cap L$), let $F_v \subseteq F_L$ (resp. $F_v \subseteq F_R$) denote the set of vertices that can be reached from v (resp. can reach v) by a directed path in H with no internal vertices from X'. If $|F_v| \leq |L_0| + |R_0|$, then let $W_v = F_v$; otherwise let W_v be an arbitrary subset of F_v of size $|L_0| + |R_0|$. Finally, let $Z = X' \cup \bigcup_{v \in X'} W_v$.

Let $H' = \text{torso}(H, Z)$. By construction there is no matching edge in M with exactly one vertex in Z, which ensures that H' is also a bipartite graph: By construction, a directed path connecting two vertices of the same side must either start or end with an edge in M and therefore that path cannot have only internal vertices in $V \setminus Z$. Thus, the torso operation does not add a shortcut edge between two vertices that are on the same side.

Now let G' be the underlying undirected graph corresponding to H' and let $k = |M[(V \setminus Z]| = |M| - |M_X|$, i.e., the number of edges of M outside of Z. We show that the maximum matching size of $G - (L' \cup R')$, for $L' \subseteq L_0$ and $R' \subseteq R_0$, can also be computed in G'.

▶ **Lemma 16.** *For every $L' \subseteq L_0$ and $R' \subseteq R_0$, the graph $G - (L' \cup R')$ has maximum matching size l if and only if $G' - (L' \cup R')$ has maximum matching size $l - k$.*

Proof. Let us fix $L' \subseteq L_0$ and $R' \subseteq R_0$ and denote $S = F_L \cup (L_0 \setminus L')$, $T = F_R \cup (R_0 \setminus R')$.

($\Rightarrow$) Let us first assume that $G - (L' \cup R')$ has a maximum matching M'_0 of size l. By Lemma 12 we have that there is a vertex-disjoint packing of M-augmenting paths $\mathcal{P}$ of size $l - |M|$ which we can use to augment M to a (maximum) matching M_0 of size $|M'_0|$. Note that M_0 agrees with M except for the augmenting paths in $\mathcal{P}$. (The same is not necessarily true for M'_0 since there could, e.g., be alternating cycles in $M'_0 \Delta M$.) Since every M-augmenting path corresponds to a directed path from S to T and $X \subseteq X' \subseteq Z$ contains a minimum (S, T)-cut we have that every path in $\mathcal{P}$ contains at least one vertex of Z. Furthermore $|\mathcal{P}| \leq |L_0| + |R_0|$ because every augmenting path must contain at least one vertex of $L_0 \cup R_0$ by M being maximum matching in $G - (L_0 \cup R_0)$ (see Observation 2).

We consider maximal subpaths of $\mathcal{P}$ with internal vertices not from Z and having at least one internal vertex. (Note that vertices in $(F_L \cup F_R) \setminus Z \subseteq F_L \cup F_R$ are M-unmatched, so paths in $\mathcal{P}$ cannot have such vertices as internal vertices: Those in F_L have no incoming edges and those in F_R have no outgoing edges.) If an internal vertex v on a path in $\mathcal{P}$ is in Z, then $v \in X' \subseteq Z$ since $Z \setminus X' \subseteq F_L \cup F_R$. If an internal vertex v of a path in $\mathcal{P}$ is in Z, and thus $v \in X'$, then we have that at least one neighbor of v on the path was also included in $X' \subseteq Z$ since v is incident with M (v is an internal vertex of an M-augmenting path). Therefore, these subpaths are vertex-disjoint. Let us consider each such subpath $P = (p, \ldots, q)$. We distinguish three cases, based on whether p and/or q are contained in Z, and apply a replacement operation to M_0 for each of them.

If $p, q \in Z$, then we have an edge $\{p, q\}$ in G' because it was either in the original graph, or the corresponding shortcut arc (p, q) was added to H' during the torso operation. Since no edge in M has exactly one vertex in Z, we have that P starts and ends with an edge of M_0, and there is exactly one less edge of M on P than there are edges of M_0 on P. We modify M_0 by removing all edges of M_0 on P, adding all edges of M on P, and adding $\{p, q\}$. After this modification the size of M_0 is the same as before and it is still a matching.

If $p \in Z$ and $q \notin Z$, then we have that $W_p \geq |L_0| + |R_0|$, since otherwise we would have $q \in F_v = W_v \subseteq Z$ since it is reachable from p with no internal vertices in $Z \supseteq X'$. We again have that the first and last edge of P start and end with an edge of M_0, since the first edge cannot be an edge of M because it has only one endpoint in Z, while the last edge is a final edge of a path in $\mathcal{P}$ by maximality of P and P ends in an M-unmatched vertex of F_R. (It cannot end in F_L because those vertices have no incoming edges.) We modify M_0 by removing all edges of M_0 on P, adding all edges of M on P and adding one edge from p into W_p. Because this case can occur at most $|L_0| + |R_0|$ times (at most once for every path in $\mathcal{P}$) we have that there is always a free vertex in W_p. Again, the size of M_0 remains the same.

We handle the case where $p \notin Z$ and $q \in Z$ similarly. Note that $p, q \notin Z$ cannot occur because then P would not a be maximal subpath with internal vertices not from Z since every path in $\mathcal{P}$ visits at least one vertex in Z. After handling every maximal subpath in this fashion there are no edges of M_0 with only one endpoint in Z. Furthermore, M_0 and M agree on all edges not incident to Z and thus $|M_0[V \setminus Z]| = |M[V \setminus Z]$. Hence, $M_0[Z]$, the restriction of M_0 to Z, is a matching in $G' - (L' \cup R')$ of size $|M_0[Z]| = |M_0| - |M_0[V \setminus Z]| = l - |M[V \setminus Z]| = l - k$.

($\Leftarrow$) Assume that $G' - (L' \cup R')$ has a matching M_0 of size $l - k$. Let $M' = M[Z] = M_X$ denote the restriction of M to Z. Since there are no edges of M with exactly one vertex in Z,

the set M' is a matching in $G' - (L' \cup R')$ of size $|M| - k$. By Lemma 12, there is a packing $\mathcal{P}$ of $r = (l-k) - |M'| = l - |M|$ vertex-disjoint M'-augmenting paths in $G' - (L' \cup R')$, which correspond to r vertex-disjoint directed paths from $S \cap Z$ to $T \cap Z$ in $H' - (L' \cup R')$.

By construction X contains a minimum (S,T)-cut Y in $H - (L' \cup R')$; suppose that $|Y| < r = |\mathcal{P}|$. There must be a path in $\mathcal{P}$ which avoids Y. This path corresponds to a directed path P from $S \cap Z$ to $T \cap Z$ in $H' - (L' \cup R')$. Arcs of P are either arcs in $H - (L' \cup R')$ or shortcut arcs which are added by the torso operation. Note that each shortcut arc corresponds to a directed path with no internal vertices from Z, which therefore also avoids $Y \subseteq X \subseteq Z$. Hence, if we replace shortcut arcs in P by corresponding paths in $H - (L' \cup R')$, then we obtain a directed walk from $S \cap Z$ to $T \cap Z$ in $H - (L' \cup R')$, which contradicts that Y is a (S,T)-cut in $H - (L' \cup R')$. Hence we have $|Y| \geq r$, which implies that there are at least r vertex-disjoint (S,T)-paths in $H - (L' \cup R')$. These paths correspond to M-augmenting paths in $G - (L' \cup R')$. Thus, $G - (L' \cup R')$ has a matching of size at least $|M| + r = |M| + (l - |M|) = l$. ◀

The size of X is polynomial $\mathcal{O}((|L_0| + |R_0|)^3)$. Therefore $|X'| = \mathcal{O}((|L_0| + |R_0|)^3)$ since we add at most $|X|$ more vertices that are an endpoint to a matched edge in M incident to X. To obtain Z, at most $|L_0| + |R_0|$ vertices are added for every vertex in X'. Thus, Z is of size $\mathcal{O}((|L_0| + |R_0|)^4)$ and therefore G' has at most $\mathcal{O}((|L_0| + |R_0|)^4)$ vertices. All operations required to obtain G' can be performed in polynomial time by using appropriate flow calculations. Correctness follows from Lemma 16, completing Theorem 10.

Uncertain edges. Our result can also be extended to the case when there are some *edges* of the input graph that may not be available; we call these edges *uncertain edges*. We proceed as follows: For each uncertain edge uv, we subdivide it into three edges $uu', u'v'$ and $v'v$ (note that this does not change the bipartiteness of the graph); then instead of considering uv as an uncertain edge, we consider u' and v' as uncertain vertices. The auxiliary vertices u' and v' can be used to control the availability of uv in the graph. We remove u' and v' to make uv unavailable and include both u and v in the final input to make uv available. Now we can repeat the algorithm in Theorem 10 with a small modification: Before applying the torso operation, we add u, u', v' and v to the cut-covering set. This ensures that the new edges and vertices that correspond to uncertain edges are not affected by the torso operation. After the torso operation has been applied, we obtain a new compressed graph with some uncertain vertices, and moreover all endpoints of uncertain edges and their subdivisions in the input graph are preserved. Finally, we may contract subdivided edges with auxiliary (uncertain) vertices to get the original uncertain edges.

The final thing we need to be careful about is how the size of maximum matching may change under subdivision (and contraction). Given a graph G, if we subdivide an edge uv of G into three edges $uu', u'v'$ and $v'v$ to obtain a graph G', then we increase the size of maximum matching of G by one. Indeed, we just need to see how to construct a matching in the new graph from a maximum matching M of G. If M contains uv then we just need to replace uv by uu' and vv'. Otherwise, we add $u'v'$ to M; in both cases the size increases by one, as claimed. Thus, we can conclude our result carries over to the case where for a subset of edges and vertices it is unknown if they appear in the final input.

6 Lower Bounds

In this section we present some unconditional lower bounds for preprocessing instances of some tractable problems in which part of the input is uncertain. We derive these lower

bounds from the Membership communication game which is defined as follows. We have two players, Alice and Bob. Alice has a subset $S \subseteq [n]$ and Bob has an integer l. The objective of the game is for Bob to find out if $l \in S$. It is well known that a one-way communication protocol from Alice to Bob has a cost of at least n bits of information. We show that for certain problems the existence of algorithms that are similar to those presented in this paper would give a more efficient protocol. We start with Connected Vertex Cover which is solvable in time $2^b|V|^{\mathcal{O}(1)}$ if we are looking for a solution of size at most b [3]. Therefore, it is solvable in polynomial time for $b \leq \log|V|$ in which case we refer to the problem as Small Connected Vertex Cover. The following theorem shows that we cannot find a succinct equivalent instance if for a set W of vertices it is uncertain if they appear in the final input.

▶ **Theorem 17.** *There is no algorithm that, given an instance $G = (V \cup W, E)$ of* Small Connected Vertex Cover*, outputs a graph G' of size $|W|^{\mathcal{O}(1)}$ and $k \in \mathbb{N}$ such that, for any $W' \subseteq W$, the graph $G - W'$ has a connected vertex cover of size at most $b \leq \log|V|$ if and only if $G' - W'$ has a connected vertex cover of size at most $b - k$.*

Note that this lower bound holds even if the time to compute G' and k is unbounded. Furthermore, we remark that by a similar information theoretic argument, any encoding from which we can extract the answer to Small Connected Vertex Cover for any $W' \subseteq W$ requires at least $2^{|W|/2} = n$ bits: Otherwise, since there are 2^n possible subsets of $[n]$, by the pigeonhole principle (we have fewer than 2^n distinct bit-strings of length smaller than n) there must be at least 2 subsets $S \neq S'$ of $[n]$ for which the encoding of the instance produced in the proof of Theorem 17 is the same. Now there must be an element i that is, w.l.o.g., in S but not in S' for which plugging in the corresponding W' produces the same answer, which is clearly wrong. We show a similar lower bound for Linear Programming where for part of the constraints it is uncertain if they appear in the final input.

▶ **Theorem 18.** *There is no algorithm that, given an LP with objective function $\boldsymbol{c}^T\boldsymbol{x}$ and constraints $A\boldsymbol{x} \leq \boldsymbol{b}, B\boldsymbol{x} \leq \boldsymbol{d}, \boldsymbol{x} \geq 0$, outputs an encoding of size smaller than $2^{c/2}$ from which we can correctly determine feasibility of the LP for any subset of constraints in $B\boldsymbol{x} \leq \boldsymbol{d}$, where c is the number of constraints in $B\boldsymbol{x} \leq \boldsymbol{d}$.*

Let us remark that a similar lower bound can be obtained for linear programming instances if for part of the constraints the right-hand side is unknown, and we would like to obtain an instance that is equivalent with respect to feasibility for any assignment of the right-hand side in these constraints. This follows from the proof of Theorem 18 when according to $s(i)$, we set $x_i \geq 1$ and $x_i \leq 1$ (resp. $x_i \geq 0$ and $x_i \leq 0$) if the i-th bit of $s(i)$ is set to 1 (resp. 0).

Finally, in a setting where the objective function of the LP is unknown we have a lower bound of 2^c where c is the number of variables: Alice uses the same constraints $Ax \leq b$ as in Theorem 18. Now, target functions $y_1 + \ldots + y_p$ can take value p if and only if there is no constraint $y_1 + \ldots + y_p \leq p - 0.5$, i.e., if and only if the corresponding element j is not in S.

7 Conclusion

We have initiated a programmatic study of preprocessing for efficiently solvable problems for the setting that not the entire input is known or that parts of the input are not certain, inspired by recent results [13, 8] obtained in the context of kernelization. Intuitively, incomplete or partially uncertain instances correspond to a possibly exponentially large family of similar instances. Thus, it is not clear (and as shown it does not hold in general) that one can efficiently generate an instance of size polynomial in the amount of uncertain data that

allows to compute optimal solutions for the initial input for each instantiation (and without access to the initial input of course). This direction of research seems to apply for a variety of polynomial-time solvable problems (and of course also for NP-hard ones). In particular, different notions of missing or uncertain data may be suitable for the same problem. We have considered weight functions whose values are unknown for certain items as well as graphs in which some vertices or edges may or may not appear in the instantiation.

Natural problems for future research are for example matchings in general graphs (with uncertain vertices, edges, and or weights) and MAX FLOW with arbitrary and uncertain capacities. It would also be interesting whether there are particular lower bound techniques for this form of preprocessing, e.g., to prove that a certain amount of information is optimal.

References

1. Sepehr Assadi, Sanjeev Khanna, Yang Li, and Val Tannen. Dynamic sketching for graph optimization problems with applications to cut-preserving sketches. *CoRR*, abs/1510.03252, 2015. URL: http://arxiv.org/abs/1510.03252.
2. 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.
3. Marek Cygan. Deterministic parameterized connected vertex cover. In *SWAT 2012*, volume 7357 of *LNCS*, pages 95–106. Springer, 2012. doi:10.1007/978-3-642-31155-0_9.
4. George B. Dantzig. Linear programming under uncertainty. *Management Science*, 50(12-Supplement):1764–1769, 2004. doi:10.1287/mnsc.1040.0261.
5. Reinhard Diestel. *Graph theory (Graduate texts in mathematics)*. Springer Heidelberg, 2005.
6. Stefan Fafianie, Stefan Kratsch, and Vuong Anh Quyen. Preprocessing under uncertainty. *CoRR*, abs/1510.05503, 2015. URL: http://arxiv.org/abs/1510.05503.
7. Michael Hoffmann, Thomas Erlebach, Danny Krizanc, Matús Mihalák, and Rajeev Raman. Computing minimum spanning trees with uncertainty. In *STACS 2008*, volume 1 of *LIPIcs*, pages 277–288, 2008. doi:10.4230/LIPIcs.STACS.2008.1358.
8. Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *FOCS 2012*, pages 450–459. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.46.
9. 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.
10. Dániel Marx, Barry O'Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Transactions on Algorithms*, 9(4):30, 2013. doi:10.1145/2500119.
11. Nicole Megow, Julie Meißner, and Martin Skutella. Randomization helps computing a minimum spanning tree under uncertainty. In *ESA 2015*, volume 9294 of *LNCS*, pages 878–890. Springer, 2015. doi:10.1007/978-3-662-48350-3_73.
12. James Oxley. *Matroid Theory*. Oxford University Press, 2011.
13. 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.

Characterisation of an Algebraic Algorithm for Probabilistic Automata

Nathanaël Fijalkow

University of Oxford, United Kingdom; and
LIAFA, Paris 7, France; and
University of Warsaw, Poland
nathanael.fijalkow@cs.ox.ac.uk

Abstract

We consider the value 1 problem for probabilistic automata over finite words: it asks whether a given probabilistic automaton accepts words with probability arbitrarily close to 1. This problem is known to be undecidable. However, different algorithms have been proposed to partially solve it; it has been recently shown that the Markov Monoid algorithm, based on algebra, is the most correct algorithm so far. The first contribution of this paper is to give a characterisation of the Markov Monoid algorithm.

The second contribution is to develop a profinite theory for probabilistic automata, called the prostochastic theory. This new framework gives a topological account of the value 1 problem, which in this context is cast as an emptiness problem. The above characterisation is reformulated using the prostochastic theory, allowing to give a modular proof.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Probabilistic Automata, Value 1 Problem, Markov Monoid Algorithm, Algebraic Algorithm, Profinite Theory, Topology in Computer Science

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.34

1 Introduction

Rabin [9] introduced the notion of probabilistic automata, which are finite automata with randomised transitions. This powerful model has been widely studied since then and has applications, for instance in image processing, computational biology and speech processing. This paper follows a long line of work that studies the algorithmic properties of probabilistic automata. We consider the value 1 problem: it asks, given a probabilistic automaton, whether there exist words accepted with probability arbitrarily close to 1.

This problem has been shown undecidable [7]. Different approaches led to construct subclasses of probabilistic automata for which the value 1 problem is decidable; the first class was ♯-acylic automata [7], then concurrently simple automata [2] and leaktight automata [4]. It has been shown in [3] that the so-called Markov Monoid algorithm introduced in [4] is the most correct algorithm of the three algorithms. Indeed, both ♯-acylic and simple automata are strictly subsumed by leaktight automata, for which the Markov Monoid algorithm correctly solves the value 1 problem.

Yet we were missing a good understanding of the computations realised by the Markov Monoid algorithm. The aim of this paper is to provide such an insight by giving a characterisation of this algebraic algorithm. We show the existence of *convergence speeds* phenomena, which can be polynomial or exponential. Our main technical contribution is to prove that the Markov Monoid algorithm captures exactly *polynomial behaviours*.


© Nathanaël Fijalkow;
licensed under Creative Commons License CC-BY
33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 34; pp. 34:1–34:13

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


Proving this characterisation amounts to give precise bounds on convergences of non-homogeneous Markov chains. Our second contribution is to define a new framework allowing to rephrase this characterisation and to give a modular proof for it, using techniques from topological and linear algebra. We develop a profinite approach for probabilistic automata, called prostochastic theory. This is inspired by the profinite approach for (classical) automata [1, 8, 6], and for automata with counters [10].

Section 3 is devoted to defining the Markov Monoid algorithm and stating the characterisation: it answers "YES" if, and only if, the probabilistic automaton accepts some polynomial sequence.

In Section 4, we introduce a new framework, the prostochastic theory, which is used to restate and prove the above characterisation. We first construct a space called the free prostochastic monoid, whose elements are called prostochastic words. We define the acceptance of a prostochastic word by a probabilistic automaton, and show that the value 1 problem can be reformulated as the emptiness problem for probabilistic automata over prostochastic words. We then explain how to construct non-trivial prostochastic words, by defining a limit operator ω, leading to the definition of polynomial prostochastic words. The above characterisation above reads in the realm of prostochastic theory as follows: the Markov Monoid algorithm answers "YES" if, and only if, the probabilistic automaton accepts some polynomial prostochastic word.

Section 5 concludes by showing how this characterisation, combined with an improved undecidability result, supports the claim that the Markov Monoid algorithm is *in some sense* optimal.

2 Probabilistic Automata and the Value 1 Problem

Let Q be a finite set of states.

A distribution over Q is a function $\delta : Q \to [0, 1]$ such that $\sum_{q \in Q} \delta(q) = 1$. We denote $\mathcal{D}(Q)$ the set of distributions over Q, which we often consider as vectors indexed by Q.

For $E \subseteq \mathbb{R}$, we denote $\mathcal{M}_{Q\times Q}(E)$ the set of (square) matrices indexed by Q over E. The space $\mathcal{M}_{Q\times Q}(\mathbb{R})$ is equipped with the norm $||\cdot||$ defined by

$$||M|| = \max_{s \in Q} \sum_{t \in Q} |M(s, t)|.$$

This induces the standard Euclidean topology on $\mathcal{M}_{Q\times Q}(\mathbb{R})$. We denote I the identity matrix. A matrix $M \in \mathcal{M}_{Q\times Q}(\mathbb{R})$ is stochastic if each line is a distribution over Q; the restriction to stochastic matrices is denoted $\mathcal{S}_{Q\times Q}(\mathbb{R})$. The following classical properties will be useful:

▶ **Fact 1** (Topology of the stochastic matrices).

- *For every matrix $M \in \mathcal{S}_{Q\times Q}(\mathbb{R})$, we have $||M|| = 1$,*
- *For every matrices $M, M' \in \mathcal{M}_{Q\times Q}(\mathbb{R})$, we have $||M \cdot M'|| \leq ||M|| \cdot ||M'||$,*
- *The monoid $\mathcal{S}_{Q\times Q}(\mathbb{R})$ is compact.*

▶ **Definition 2** (Probabilistic automaton). A *probabilistic automaton* $\mathcal{A}$ is given by a finite set of states Q, a transition function $\phi : A \to \mathcal{S}_{Q\times Q}(\mathbb{Q})$, an initial state $q_0 \in Q$ and a set of final states $F \subseteq Q$.

Observe that it generalises the definition for classical deterministic automata, in which transitions functions are $\phi : A \to \mathcal{S}_{Q\times Q}(\{0, 1\})$.

A transition function $\phi : A \to \mathcal{S}_{Q\times Q}(\mathbb{Q})$ naturally induces a morphism $\phi : A^* \to \mathcal{S}_{Q\times Q}(\mathbb{Q})$[1].

We denote $P_{\mathcal{A}}(s \xrightarrow{w} t)$ the probability to go from state s to state t reading w on the automaton $\mathcal{A}$, *i.e.* $\phi(w)(s,t)$. We extend the notation: for a subset T of the set of states, $P_{\mathcal{A}}(s \xrightarrow{w} T)$ is defined by $\phi(w)(s,T) = \sum_{t\in T} \phi(w)(s,t)$.

The *acceptance probability* of a word $w \in A^*$ by $\mathcal{A}$ is $P_{\mathcal{A}}(q_0 \xrightarrow{w} F)$, denoted $P_{\mathcal{A}}(w)$. In words, it is the probability that a run starting from the initial state q_0 ends in a final state (*i.e.* a state in F).

The *value* of a probabilistic automaton $\mathcal{A}$ is $\mathrm{val}(\mathcal{A}) = \sup\{P_{\mathcal{A}}(w) \mid w \in A^*\}$, the supremum over all words of the acceptance probability.

▶ **Definition 3** (Value 1 Problem)**.** The value 1 problem is the following decision problem: given a probabilistic automaton $\mathcal{A}$ as input, determine whether $\mathrm{val}(\mathcal{A}) = 1$, *i.e.* whether there exist words whose acceptance probability is arbitrarily close to 1.

Equivalently, the value 1 problem asks for the existence of a sequence of words $(u_n)_{n\in\mathbb{N}}$ such that $\lim_n P_{\mathcal{A}}(u_n) = 1$.

3 Characterisation of the Markov Monoid Algorithm

3.1 Definition

The Markov Monoid algorithm was introduced in [4], we give here a different yet equivalent presentation. Consider $\mathcal{A}$ a probabilistic automaton, the Markov Monoid algorithm consists in computing, by a saturation process, the Markov Monoid of $\mathcal{A}$. It is a monoid of Boolean matrices: all numerical values are projected away to Boolean values.

We give a few definitions and conventions. Throughout the paper, M denotes a matrix in $\mathcal{S}_{Q\times Q}(\mathbb{R})$, and m a Boolean matrix. The following definitions mimick the notions of recurrent and transient states from Markov chain theory.

▶ **Definition 4** (Idempotent Boolean matrix, recurrent and transient state)**.** Consider a matrix $M \in \mathcal{S}_{Q\times Q}(\mathbb{R})$, its Boolean projection $\pi(M)$ is the Boolean matrix such that $\pi(M)(s,t) = 1$ if $M(s,t) > 0$, and $\pi(M)(s,t) = 0$ otherwise.

Let m be a Boolean matrix. It is idempotent if $m \cdot m = m$.

Assume m is idempotent. We say that:

- the state $s \in Q$ is m-recurrent if for all $t \in Q$, if $m(s,t) = 1$, then $m(t,s) = 1$,
- the m-recurrent states $s, t \in Q$ belong to the same recurrence class if $m(s,t) = 1$,
- the state $s \in Q$ is m-transient if it is not m-recurrent.

Since the Markov Monoid only considers Boolean values, one can consider the underlying non-deterministic automaton $\pi(\mathcal{A})$ instead of the probabilistic automaton $\mathcal{A}$. Formally, $\pi(A)$ is defined as $\mathcal{A}$, except that its transitions are given by $\pi(\phi(a))$ for the letter $a \in A$.

The Markov Monoid of $\pi(\mathcal{A})$ contains the transition monoid of $\pi(\mathcal{A})$, which is the monoid generated by $\{\pi(\phi(a)) \mid a \in A\}$ and closed under (Boolean matrix) products. Informally speaking, the transition monoid accounts for the Boolean action of every finite word. Formally, for a word $w \in A^*$, the element $\langle w\rangle$ of the transition monoid of $\pi(\mathcal{A})$ satisfies the following: $\langle w\rangle(s,t) = 1$ if, and only if, there exists a run from s to t reading w on $\pi(\mathcal{A})$.

[1] Note that we use "morphism" for "monoid homomorphism" throughout the paper.

The Markov Monoid extends the transition monoid by introducing a new operator, the stabilisation. On the intuitive level first: let $M \in \mathcal{S}_{Q\times Q}(\mathbb{R})$, it can be interpreted as a Markov chain; its Boolean projection $\pi(M)$ give the structural properties of this Markov chain. The stabilisation $\pi(M)^\sharp$ accounts for $\lim_n M^n$, *i.e.* the behaviour of the Markov chain M in the limit. The formal definition of the stabilisation operator is as follows:

▶ **Definition 5** (Stabilisation)**.** Let m be a Boolean idempotent matrix.
The stabilisation of m is denoted $m^\sharp$ and defined by:

$$m^\sharp(s,t) = \begin{cases} 1 & \text{if } m(s,t) = 1 \text{ and } t \text{ is } m\text{-recurrent,} \\ 0 & \text{otherwise.} \end{cases}$$

The definition of the stabilisation matches the intuition that in the Markov chain $\lim_n M^n$, the probability to be in non-recurrent states converges to 0.

▶ **Definition 6** (Markov Monoid)**.** The Markov Monoid of an automaton $\mathcal{A}$ is the smallest set of Boolean matrices containing $\{\pi(\phi(a)) \mid a \in A\}$ and closed under product and stabilisation of idempotents.

Algorithm 1: The Markov Monoid algorithm.

Data: A probabilistic automaton.
$\mathcal{M} \leftarrow \{\pi(\phi(a)) \mid a \in A\} \cup \{I\}$.
repeat
  if *there is* $m, m' \in \mathcal{M}$ *such that* $m \cdot m' \notin \mathcal{M}$ **then**
    add $m \cdot m'$ to $\mathcal{M}$
  end
  if *there is* $m \in \mathcal{M}$ *such that* m *is idempotent and* $m^\sharp \notin \mathcal{M}$ **then**
    add $m^\sharp$ to $\mathcal{M}$
  end
until *there is nothing to add*;
if *there is a value* 1 *witness in* $\mathcal{M}$ **then**
  return YES;
else
  return NO;
end

On an intuitive level, a Boolean matrix in the Markov Monoid reflects the asymptotic effect of a sequence of finite words.

The Markov Monoid algorithm computes the Markov Monoid, and looks for *value* 1 *witnesses*:

▶ **Definition 7** (Value 1 witness)**.** Let $\mathcal{A}$ be a probabilistic automaton.
A Boolean matrix m is a value 1 witness if: for all states $t \in Q$, if $m(q_0, t) = 1$, then $t \in F$.

The Markov Monoid algorithm answers "YES" if there exists a value 1 witness in the Markov Monoid, and "NO" otherwise.

3.2 An Example

We apply the Markov Monoid algorithm on an example. As explained, the Markov Monoid does not take into account the numerical values of the probabilistic transition, so as input

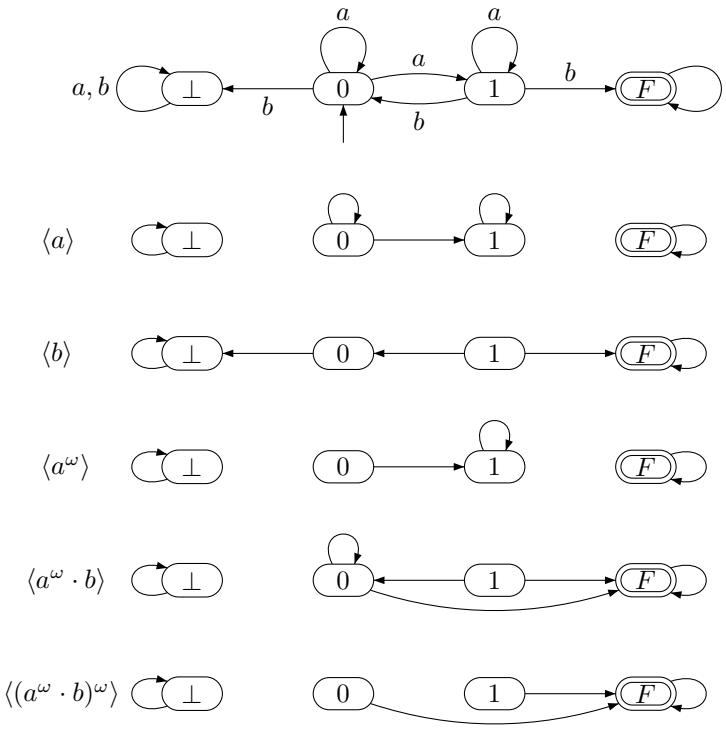


Figure 1 A non-deterministic automaton and part of its Markov Monoid.

we can consider the underlying non-deterministic automaton of a probabilistic automaton. This is what we do in figure 1: the non-deterministic automaton is represented at the top.

We do not represent here all elements of its Markov Monoid; the tool ACME computed it [5], it contains 42 elements. We chose to represent here only 5 elements:

- The first two correspond to the letters a and b.
- The third one is $\langle a^\omega \rangle$, its represents the behaviour of $(a^n)_{n\in\mathbb{N}}$. Indeed, when reading a from the state 0, two events happen with positive probability: looping around the state 0 and going to state 1. So, reading the word a^n from 0 gives a probability to remain in the state 0 converging to 0 when n goes to infinity. Formally, this is reflected by the fact that the state 0 is not $\langle a \rangle$-recurrent.
- The fourth one is $\langle a^\omega \cdot b \rangle$, it illustrates the concatenation between $\langle a^\omega \rangle$ and $\langle b \rangle$. Note that it is not a value 1 witness: from the only initial state 0, we have $\langle a^\omega \cdot b \rangle(0,0) = 1$, and 0 is not final.
- The fifth one is $\langle (a^\omega \cdot b)^\omega \rangle$, it illustrates that stabilisation can be nested. Observe that it is a value 1 witness. This matches the calculation showing that $\lim_n P_{\mathcal{A}}((a^n \cdot b)^n) = 1$, implying that $\mathcal{A}$ has value 1.

3.3 Characterisation

Our main technical result is the following theorem, which is a characterisation of the Markov Monoid algorithm. It relies on the notion of *polynomial sequences of words*.

We define two operations for sequences of words, mimicking the operations of the Markov Monoid:

- the first is concatenation: given $(u_n)_{n\in\mathbb{N}}$ and $(v_n)_{n\in\mathbb{N}}$, the concatenation is the sequence $(u_n \cdot v_n)_{n\in\mathbb{N}}$,
- the second is iteration: given $(u_n)_{n\in\mathbb{N}}$, its iteration is the sequence $(u_n^n)_{n\in\mathbb{N}}$; the n^{th} word is repeated n times.

▶ **Definition 8** (Polynomial sequence)**.** The class of polynomial sequences is the smallest class of sequences containing the constant sequences $(\varepsilon)_{n\in\mathbb{N}}$ and $(a)_{n\in\mathbb{N}}$ for $a \in A$, and closed under concatenation and iteration.

A typical example of a polynomial sequence is $((a^n b)^n)_{n\in\mathbb{N}}$, and a typical example of a sequence which is not polynomial is $\left((a^n b)^{2^n}\right)_{n\in\mathbb{N}}$.

We proceed to our main result:

▶ **Theorem 9** (Characterisation of the Markov Monoid algorithm)**.** *The Markov Monoid algorithm answers "YES" on input $\mathcal{A}$ if, and only if, there exists a polynomial sequence $(u_n)_{n\in\mathbb{N}}$ such that $\lim_n P_{\mathcal{A}}(u_n) = 1$.*

This result could be proved directly, without appealing to the prostochastic theory developed in the next section. The proof relies on technically intricate calculations over non-homogeneous Markov chains; the prostochastic theory allows to simplify its presentation, making it more modular. We will give the proof of Theorem 9 in Subsection 4.4, after restating it using the prostochastic theory.

A second advantage of using the prostochastic theory is to give a more natural and robust definition of polynomial sequences, which in the prostochastic theory correspond to polynomial prostochastic words.

A direct corollary of Theorem 9 is the absence of false negatives:

▶ **Corollary 10** (No false negatives for the Markov Monoid algorithm)**.** *If the Markov Monoid algorithm answers "YES" on input $\mathcal{A}$, then $\mathcal{A}$ has value 1.*

4 The Prostochastic Theory

In this section, we introduce the prostochastic theory, which draws from profinite theory to give a topological account of probabilistic automata. We construct the free prostochastic monoid in Subsection 4.1.

The aim of this theory is to give a topological account of the value 1 problem; we show in Subsection 4.2 that the value 1 problem can be reformulated as an emptiness problem for prostochastic words.

In Subsection 4.3 we define the notion of polynomial prostochastic words.

The characterisation given in Section 3 is stated and proved in this new framework in Subsection 4.4.

4.1 The Free Prostochastic Monoid

The purpose of the prostochastic theory is to construct a compact monoid $\mathcal{P}A^*$ together with a continuous injective morphism $\iota : A^* \to \mathcal{P}A^*$, called the free prostochastic monoid, satisfying the following universal property:

"Every morphism $\phi : A^* \to \mathcal{S}_{Q\times Q}(\mathbb{R})$ extends uniquely
to a continuous morphism $\widehat{\phi} : \mathcal{P}A^* \to \mathcal{S}_{Q\times Q}(\mathbb{R})$."

Here, by "$\widehat{\phi}$ extends ϕ" we mean $\phi = \widehat{\phi} \circ \iota$.

We give two statements about $\mathcal{P}A^*$, the first will be weaker but enough for our purposes in this paper, and the second more precise, and justifying the name "free prostochastic monoid". The reason for giving two statements is that the first avoids a number of technical points that will not play any further role, so the reader interested in the applications to the Markov Monoid algorithm may skip this second statement.

▶ **Theorem 11** (Existence of the free prostochastic monoid – weaker statement)**.** *For every finite alphabet A, there exists a compact monoid $\mathcal{P}A^*$ and a continuous injective morphism $\iota : A^* \to \mathcal{P}A^*$ such that every morphism $\phi : A^* \to \mathcal{S}_{Q\times Q}(\mathbb{R})$ extends uniquely to a continuous morphism $\widehat{\phi} : \mathcal{P}A^* \to \mathcal{S}_{Q\times Q}(\mathbb{R})$.*

We construct $\mathcal{P}A^*$ and ι. Consider $X = \prod_{\phi:A^*\to\mathcal{S}_{Q\times Q}(\mathbb{R})} \mathcal{S}_{Q\times Q}(\mathbb{R})$, the product of several copies of $\mathcal{S}_{Q\times Q}(\mathbb{R})$, one for each morphism $\phi : A^* \to \mathcal{S}_{Q\times Q}(\mathbb{R})$. An element m of X is denoted $(m(\phi))_{\phi:A^*\to\mathcal{S}_{Q\times Q}(\mathbb{R})}$: it is given by an element $m(\phi)$ of $\mathcal{S}_{Q\times Q}(\mathbb{R})$ for each morphism $\phi : A^* \to \mathcal{S}_{Q\times Q}(\mathbb{R})$. Thanks to Tychonoff's theorem, the monoid X equipped with the product topology is compact[2].

Consider the map $\iota : A \to X$ defined by $\iota(a) = (\phi(a))_{\phi:A\to\mathcal{P}}$, it induces a continuous injective morphism $\iota : A^* \to X$. To simplify notations, we sometimes assume that $A \subseteq X$ and denote a for $\iota(a)$.

Denote $\mathcal{P}A^* = \overline{A^*}$, the closure of $A^* \subseteq X$. Note that it is a compact monoid: the compactness follows from the fact that it is closed in X. By definition, an element $\overline{u}$ of $\mathcal{P}A^*$, called a *prostochastic word*, is obtained as the limit in $\mathcal{P}A^*$ of a sequence $\mathbf{u}$ of finite words. In this case we write $\lim \mathbf{u} = \overline{u}$ and say that $\mathbf{u}$ induces $\overline{u}$.

Note that by definition of the product topology on X, a sequence of finite words $\mathbf{u}$ converges in X if, and only if, for every morphism $\phi : A^* \to \mathcal{S}_{Q\times Q}(\mathbb{R})$, the sequence of stochastic matrices $\phi(\mathbf{u})$ converges.

We say that two converging sequences of finite words $\mathbf{u}$ and $\mathbf{v}$ are equivalent if they induce the same prostochastic word, *i.e.* if $\lim \mathbf{u} = \lim \mathbf{v}$. Equivalently, two converging sequences of finite words $\mathbf{u}$ and $\mathbf{v}$ are equivalent if, and only if, for every morphism $\phi : A^* \to \mathcal{S}_{Q\times Q}(\mathbb{R})$, we have $\lim \phi(\mathbf{u}) = \lim \phi(\mathbf{v})$.

We give a second, stronger statement about $\mathcal{P}A^*$, which in particular justifies the name "free prostochastic monoid".

From now on, by "monoid" we mean "compact topological monoids". The term topological means that the product function is continuous:

$$\begin{cases} \mathcal{P}\times\mathcal{P} & \to & \mathcal{P} \\ (s,t) & \mapsto & s\cdot t \end{cases}$$

A monoid is profinite if any two elements can be distinguished by a morphism into a finite monoid, *i.e.* by a finite automaton. (Formally speaking, this is the definition of residually finite monoids, which coincide with profinite monoids for compact monoids, see [1].)

To define prostochastic monoids, we use a stronger distinguishing feature, namely probabilistic automata. Probabilistic automata correspond to stochastic matrices over the rationals; here we use stochastic matrices over the reals, since $\mathcal{S}_{Q\times Q}(\mathbb{R})$ is compact, while $\mathcal{S}_{Q\times Q}(\mathbb{Q})$ is not.

[2] Note that here by compact we mean Hausdorff compact: distinct points have disjoint neighbourhoods.

▶ **Definition 12** (Prostochastic monoid)**.** A monoid $\mathcal{P}$ is prostochastic if for every elements $s \neq t$ in $\mathcal{P}$, there exists a continuous morphism $\psi : \mathcal{P} \to \mathcal{S}_{Q\times Q}(\mathbb{R})$ such that $\psi(s) \neq \psi(t)$.

There are many more prostochastic monoids than profinite monoids. Indeed, $\mathcal{S}_{Q\times Q}(\mathbb{R})$ is prostochastic, but not profinite in general.

The following theorem extends Theorem 11. The statement is the same as in the profinite theory, replacing "profinite monoid" by "prostochastic monoid".

▶ **Theorem 13** (Existence of the free prostochastic monoid – stronger statement)**.** *For every finite alphabet A,*

1. *There exists a prostochastic monoid $\mathcal{P}A^*$ and a continuous injective morphism $\iota : A^* \to \mathcal{P}A^*$ such that every morphism $\phi : A^* \to \mathcal{P}$, where $\mathcal{P}$ is a prostochastic monoid, extends uniquely to a continuous morphism $\widehat{\phi} : \mathcal{P}A^* \to \mathcal{P}$.*
2. *All prostochastic monoids satisfying this universal property are homeomorphic.*

The unique prostochastic monoid satisfying the universal property stated in item 1. is called the free prostochastic monoid, and denoted $\mathcal{P}A^*$.

▶ Remark. The free prostochastic monoid $\mathcal{P}A^*$ contains the free profinite monoid $\widehat{A^*}$. To see this, we start by recalling some properties of $\widehat{A^*}$, which is the set of *converging* sequences up to *equivalence*, where:

- a sequence of finite words $\mathbf{u}$ is converging if, and only if, for every deterministic automaton $\mathcal{A}$, the sequence is either ultimately accepted by $\mathcal{A}$ or ultimately rejected by $\mathcal{A}$, *i.e.* there exists $N \in \mathbb{N}$ such that either for all $n \geq N$, the word u_n is accepted by $\mathcal{A}$, or for all $n \geq N$, the word u_n is rejected by $\mathcal{A}$,
- two sequences of finite words $\mathbf{u}$ and $\mathbf{v}$ are equivalent if for every deterministic automaton $\mathcal{A}$, either both sequences are ultimately accepted by $\mathcal{A}$, or both sequences are ultimately rejected by $\mathcal{A}$.

Clearly:

- if a sequence of finite words is converging with respect to $\mathcal{P}A^*$, then it is converging with respect to $\widehat{A^*}$, as deterministic automata form a subclass of probabilistic automata,
- if two sequences of finite words are equivalent with respect to $\mathcal{P}A^*$, then they are equivalent with respect to $\widehat{A^*}$.

Every profinite word induces at least one prostochastic word: by compactness of $\mathcal{P}A^*$, every sequence of finite words $\mathbf{u}$ contains a converging subsequence with respect to $\mathcal{P}A^*$. This defines an injection from $\widehat{A^*}$ into $\mathcal{P}A^*$. In particular, this implies that $\mathcal{P}A^*$ is uncountable.

4.2 Reformulation of the Value 1 Problem

The aim of this subsection is to show that the value 1 problem, which talks about sequences of finite words, can be reformualted as an emptiness problem over prostochastic words.

▶ **Definition 14** (Prostochastic language of a probabilistic automaton)**.** Let $\mathcal{A}$ be a probabilistic automaton. The prostochastic language of $\mathcal{A}$ is:

$$L(\mathcal{A}) = \{\overline{u} \mid \widehat{\phi}(\overline{u})(q_0, F) = 1\}.$$

We say that $\mathcal{A}$ accepts a prostochastic word $\overline{u}$ if $\overline{u} \in L(\mathcal{A})$.

▶ **Theorem 15** (Reformulation of the value 1 problem)**.** *Let $\mathcal{A}$ be a probabilistic automaton. The following are equivalent:*

- $\mathrm{val}(\mathcal{A}) = 1$,
- *$L(\mathcal{A})$ is non-empty.*

Proof. Assume $\mathrm{val}(\mathcal{A}) = 1$, then there exists a sequence of words $\mathbf{u}$ such that $\lim P_{\mathcal{A}}(\mathbf{u}) = 1$. We see $\mathbf{u}$ as a sequence of prostochastic words. By compactness of $\mathcal{P}A^*$ it contains a converging subsequence. The prostochastic word induced by this subsequence belongs to $L(\mathcal{A})$.

Conversely, let $\overline{u}$ in $L(\mathcal{A})$, *i.e.* such that $\widehat{\phi}(\overline{u})(q_0, F) = 1$. Consider a sequence of finite words $\mathbf{u}$ inducing $\overline{u}$. By definition, we have $\lim \phi(\mathbf{u})(q_0, F) = 1$, *i.e.* $\lim P_{\mathcal{A}}(\mathbf{u}) = 1$, implying that $\mathrm{val}(\mathcal{A}) = 1$. ◀

4.3 The Limit Operator, Fast and Polynomial Prostochastic Words

We show in this subsection how to construct non-trivial prostochastic words, and in particular the polynomial prostochastic words. To this end, we need to better understand *convergence speeds phenomena*: different limit behaviours can occur, depending on how fast the underlying Markov chains converge.

We define a limit operator ω. Consider the function $f : \mathbb{N} \to \mathbb{N}$ defined by $f(n) = k!$, where k is maximal such that $k! \leq n$. The function f grows linearly: roughly, $f(n) \sim n$. The choice of n is arbitrary; one could replace n by any polynomial, or even by any subexponential function, see Remark 4.3.

The operator ω takes as input a sequence of finite words, and outputs a sequence of finite words. Formally, let $\mathbf{u}$ be a sequence of finite words, define:

$$\mathbf{u}^\omega = (u_n^{f(n)})_{n \in \mathbb{N}}.$$

It is not true in general that if $\mathbf{u}$ converges, then $\mathbf{u}^\omega$ converges. We will show that a sufficient condition is that $\mathbf{u}$ is fast.

We say that a sequence $(M_n)_{n \in \mathbb{N}}$ converges exponentially fast to M if there exists a constant $C > 1$ such that for all n large enough, $||M_n - M|| \leq C^{-n}$.

▶ **Definition 16** (Fast sequence)**.** A sequence of finite words $\mathbf{u}$ is fast if it converges (we denote $\overline{u}$ the prostochastic word it induces), and for every morphism $\phi : A^* \to \mathcal{S}_{Q \times Q}(\mathbb{R})$, the sequence $(\phi(u_n))_{n \in \mathbb{N}}$ converges exponentially fast.

A prostochastic word is *fast* if it is induced by *some* fast sequence. We denote by $\mathcal{P}A^*_f$ the set of fast prostochastic words. Note that a priori, not all prostochastic words are induced by some fast sequence.

We first prove that $\mathcal{P}A^*_f$ is a submonoid of $\mathcal{P}A^*$.

▶ **Lemma 17** (The concatenation of two fast sequences is fast)**.** *Let* $\mathbf{u}, \mathbf{v}$ *be two fast sequences. The sequence* $\mathbf{u} \cdot \mathbf{v} = (u_n \cdot v_n)_{n \in \mathbb{N}}$ *is fast.*

Let $\overline{u}$ and $\overline{v}$ be two fast prostochastic words, thanks to Lemma 17, the prostochastic word $\overline{u} \cdot \overline{v}$ is fast.

The remainder of this subsection is devoted to proving that ω is an operator $\mathcal{P}A^*_f \to \mathcal{P}A^*_f$. This is the key technical point of our characterisation. Indeed, we will define polynomial prostochastic words using concatenation and the operator ω, mimicking the definition of polynomial sequences of finite words. The fact that ω preserves the fast property of prostochastic words allows to obtain a perfect correspondence between polynomial sequences of finite words and polynomial prostochastic words.

Recall that M denotes a matrix in $\mathcal{S}_{Q \times Q}(\mathbb{R})$, and m a Boolean matrix. Note that when considering stochastic matrices we compute in the real semiring, and when considering

Boolean matrices, we compute products in the Boolean semiring, leading to two distinct notions of idempotent matrices.

The main technical tool is the following theorem, stating the exponentially fast convergence of the powers of a stochastic matrix.

▶ **Theorem 18** (Powers of a stochastic matrix). *Let $M \in \mathcal{S}_{Q\times Q}(\mathbb{R})$. Denote $P = M^{|Q|!}$. Then the sequence $(P^n)_{n\in\mathbb{N}}$ converges exponentially fast to a matrix M^ω, satisfying:*

$$\pi(M^\omega)(s,t) = \begin{cases} 1 & \text{if } \pi(P)(s,t) = 1 \text{ and } t \text{ is } \pi(P)\text{-recurrent,} \\ 0 & \text{otherwise.} \end{cases}$$

The following lemma shows that the ω operator is well defined by fast sequences. The second item shows that ω commutes with morphisms.

▶ **Lemma 19** (Limit operator for fast sequences). *Let $\mathbf{u}, \mathbf{v}$ be two equivalent fast sequences, inducing the fast prostochastic word $\overline{u}$. Then the sequences $\mathbf{u}^\omega$ and $\mathbf{v}^\omega$ are fast and equivalent, inducing the fast prostochastic word denoted $\overline{u}^\omega$.*

Furthermore, for every morphism $\phi : A^* \to \mathcal{S}_{Q\times Q}(\mathbb{R})$, we have $\widehat{\phi}(\overline{u}^\omega) = \widehat{\phi}(\overline{u})^\omega$.

We can now define polynomial prostochastic words.

First, ω-expressions are described by the following grammar:

$$E \quad \longrightarrow \quad a \quad | \quad E \cdot E \quad | \quad E^\omega.$$

We define an interpretation $\overline{\cdot}$ of ω-expressions into fast prostochastic words:

- $\overline{a}$ is prostochastic word induced by the constant sequence of the one letter word a,
- $\overline{E_1 \cdot E_2} = \overline{E_1} \cdot \overline{E_2}$,
- $\overline{E^\omega} = \overline{E}^\omega$.

The following definition of polynomial prostochastic words is in one-to-one correspondence with the definition of polynomial sequences of finite words.

▶ **Definition 20** (Polynomial prostochastic word). The set of *polynomial prostochastic words* is $\{\overline{E} \mid E \text{ is an } \omega\text{-expression}\}$.

▶ Remark. Why the term polynomial? Consider an ω-expression E, say $(a^\omega b)^\omega$, and the prostochastic word $\overline{(a^\omega b)^\omega}$, which is induced by the sequence of finite words $((a^{f(n)}b)^{f(n)})_{n\in\mathbb{N}}$. Roughly speaking $f(n) \sim n$, so this sequence represents a polynomial behaviour. Furthermore, the proofs above yield the following robustness property: all converging sequences of finite words $((a^{g(n)}b)^{h(n)})_{n\in\mathbb{N}}$, where $g, h : \mathbb{N} \to \mathbb{N}$ are subexponential functions, are equivalent, so they induce the same polynomial prostochastic word $\overline{(a^\omega b)^\omega}$. We say that a function $g : \mathbb{N} \to \mathbb{N}$ is subexponential if for all constants $C > 1$ we have $\lim_n g(n) \cdot C^{-n} = 0$; all polynomial functions are subexponential.

This justifies the terminology; we say that the polynomial prostochastic words represent all polynomial behaviours.

4.4 Reformulating the Characterisation

For proof purposes, we give an equivalent presentation of the Markov Monoid through ω-expressions. Given a probabilistic automaton $\mathcal{A}$, we define an interpretation $\langle\cdot\rangle$ of ω-expressions into Boolean matrices:

- $\langle a \rangle$ is $\pi(\phi(a))$,
- $\langle E_1 \cdot E_2 \rangle$ is $\langle E_1 \rangle \cdot \langle E_2 \rangle$,
- $\langle E^\omega \rangle$ is $\langle E \rangle^\sharp$, only defined if $\langle E \rangle$ is idempotent.

Then the Markov Monoid of $\mathcal{A}$ is $\{\langle E \rangle \mid E \text{ an } \omega\text{-expression}\}$.

The following theorem is a reformulation of Theorem 9, using the prostochastic theory. It clearly implies Theorem 9: indeed, a polynomial prostochastic word induces a polynomial sequence, and vice-versa.

▶ **Theorem 21** (Characterisation of the Markov Monoid algorithm). *The Markov Monoid algorithm answers "YES" on input $\mathcal{A}$ if, and only if, there exists a polynomial prostochastic word accepted by $\mathcal{A}$.*

The proof relies on the notion of reification, used in the following proposition, from which follows Theorem 21.

▶ **Definition 22** ((Reification)). Let $\mathcal{A}$ be a probabilistic automaton.

A sequence $(u_n)_{n\in\mathbb{N}}$ of words reifies a Boolean matrix m if for all states $s, t \in Q$, the sequence $\left(P_{\mathcal{A}}(s \xrightarrow{u_n} t)\right)_{n\in\mathbb{N}}$ converges and:

$$m(s,t) = 1 \iff \lim_n P_{\mathcal{A}}(s \xrightarrow{u_n} t) > 0.$$

▶ **Proposition 23** (Characterisation of the Markov Monoid algorithm). *For every ω-expression E, for every $\phi : A \to \mathcal{S}_{Q\times Q}(\mathbb{R})$, we have*

$$\pi(\widehat{\phi}(\overline{E})) = \langle E \rangle.$$

Consequently, for every probabilistic automaton $\mathcal{A}$:

- *any sequence inducing the polynomial prostochastic word $\overline{E}$ reifies $\langle E \rangle$,*
- *the element $\langle E \rangle$ of the Markov Monoid is a value 1 witness if, and only if, the polynomial prostochastic word $\overline{E}$ is accepted by $\mathcal{A}$.*

5 Towards an Optimality Argument

It was shown in [3] that the Markov Monoid algorithm subsumes all previous known algorithms to solve the value 1 problem. Indeed, it was proved that it is correct for the subclass of leaktight automata, and that the class of leaktight automata strictly contains all subclasses for which the value 1 problem has been shown to be decidable.

At this point, the Markov Monoid algorithm is the best algorithm *so far*. But can we go further? If we cannot, then what is an optimality argument? It consists in constructing a maximal subclass of probabilistic automata for which the problem is decidable. We can reverse the point of view, and equivalently construct an optimal algorithm, *i.e.* an algorithm that correctly solves a subset of the instances, such that no algorithm correctly solves a superset of these instances. However, it is clear that no such strong statement holds, as one can always from any algorithm obtain a better algorithm by precomputing finitely many instances.

Since there is no strong optimality argument, we can only give a subjective argument. We argue that the combination of our characterisation from Section 3 and the undecidability of the following problem, called the two-tier value 1 problem, supports the claim that the Markov Monoid algorithm is *in some sense* optimal.

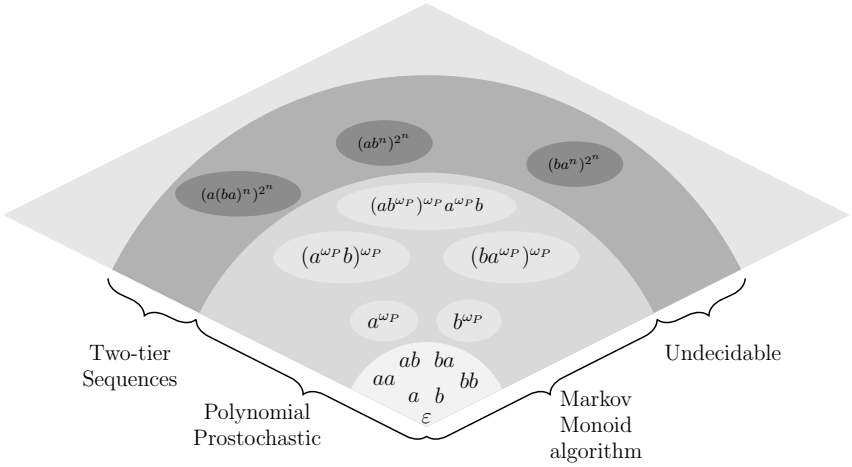


Figure 2 Optimality of the Markov Monoid algorithm.

▶ **Theorem 24** (Undecidability of the two-tier value 1 problem). *The following problem is undecidable: given a probabilistic automaton $\mathcal{A}$, determine whether there exist two finite words u, v such that $\lim_n P_{\mathcal{A}}((u \cdot v^n)^{2^n}) = 1$.*

Note that the two-tier value 1 problem is a priori much easier than the value 1 problem, as it restricts the set of sequences of finite words to very simple sequences. We call such sequences two-tier, because they exhibit two different behaviours: the word v is repeated a linear number of times, namely n, while the word $u \cdot v^n$ is repeated an exponential number of times, namely 2^n. The proof is obtained using the same reduction as for the undecidability of the value 1 problem, from [7], with a refined analysis.

To conclude:

- The characterisation says that the Markov Monoid algorithm captures exactly all polynomial behaviours.
- The undecidability result says that the undecidability of the value 1 problem arises when polynomial and exponential behaviours are combined.

So, the Markov Monoid algorithm is optimal in the sense that it captures a *large* set of behaviours, namely polynomial behaviours, and that no algorithm can capture both polynomial and exponential behaviours.

Acknowledgments. This paper and its author owe a lot to Szymon Toruńczyk's PhD thesis and its author, to Sam van Gool for his expertise on Profinite Theory, to Mikołaj Bojańczyk for his insightful remarks and to Jean-Éric Pin for his numerous questions and comments. The opportunity to present partial results on this topic in several scientific meetings has been a fruitful experience, and I thank everyone that took part in it.

References

1 Jorge Almeida. Profinite semigroups and applications. *Structural Theory of Automata, Semigroups, and Universal Algebra*, 207:1–45, 2005.

2 Krishnendu Chatterjee and Mathieu Tracol. Decidable problems for probabilistic automata on infinite words. In *LICS*, pages 185–194, 2012. doi:10.1109/LICS.2012.29.

3 Nathanaël Fijalkow, Hugo Gimbert, Edon Kelmendi, and Youssouf Oualhadj. Deciding the value 1 problem for probabilistic leaktight automata. *Logical Methods in Computer Science*, 11(1), 2015.

4 Nathanaël Fijalkow, Hugo Gimbert, and Youssouf Oualhadj. Deciding the value 1 problem for probabilistic leaktight automata. In *LICS*, pages 295–304, 2012. doi:10.1109/LICS.2012.40.

5 Nathanaël Fijalkow and Denis Kuperberg. ACME: automata with counters, monoids and equivalence. In *ATVA*, pages 163–167, 2014. doi:10.1007/978-3-319-11936-6_12.

6 Mai Gehrke, Serge Grigorieff, and Jean-Éric Pin. A topological approach to recognition. In *ICALP (2)*, pages 151–162, 2010. doi:10.1007/978-3-642-14162-1_13.

7 Hugo Gimbert and Youssouf Oualhadj. Probabilistic automata on finite words: Decidable and undecidable problems. In *ICALP (2)*, pages 527–538, 2010. doi:10.1007/978-3-642-14162-1_44.

8 Jean-Éric Pin. Profinite methods in automata theory. In *STACS*, pages 31–50, 2009. doi:10.4230/LIPIcs.STACS.2009.1856.

9 Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963. doi:10.1016/S0019-9958(63)90290-0.

10 Szymon Toruńczyk. *Languages of Profinite Words and the Limitedness Problem*. PhD thesis, University of Warsaw, 2011.

Semantic Versus Syntactic Cutting Planes

Yuval Filmus[1], Pavel Hrubeš[2], and Massimo Lauria[3]

1 **Technion – Israel Institute of Technology, Haifa, Israel**
`yuvalfi@cs.technion.ac.il`
2 **Institute of Mathematics of ASCR, Prague, Czech Republic**
`pahrubes@gmail.com`
3 **Universitat Politècnica de Catalunya, Barcelona, Catalonia, Spain**
`lauria.massimo@gmail.com`

Abstract

In this paper, we compare the strength of the semantic and syntactic version of the *cutting planes* proof system.

First, we show that the lower bound technique of Pudlák applies also to semantic cutting planes: the proof system has feasible interpolation via monotone real circuits, which gives an exponential lower bound on lengths of semantic cutting planes refutations.

Second, we show that semantic refutations are stronger than syntactic ones. In particular, we give a formula for which any refutation in syntactic cutting planes requires exponential length, while there is a polynomial length refutation in semantic cutting planes. In other words, syntactic cutting planes does not p-simulate semantic cutting planes. We also give two incompatible integer inequalities which require exponential length refutation in syntactic cutting planes.

Finally, we pose the following problem, which arises in connection with semantic inference of arity larger than two: can every multivariate non-decreasing real function be expressed as a composition of non-decreasing real functions in two variables?

1998 ACM Subject Classification F.2.2 Complexity of proof procedures

Keywords and phrases proof complexity, cutting planes, lower bounds

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.35

1 Introduction

Cutting planes is a proof system designed to show that a given set of linear inequalities has no $0,1$-solution. After the resolution system, it is one of the best known proof systems. As a procedure for solving integer linear programs, it was considered by Gomory and Chvátal [13, 7]. The idea is to compute the optimum of the program as if it were a linear program. If the optimum is achieved at a fractional point, it is possible to deduce an inequality which can be rounded in order to remove the point from the set of feasible solutions. Another way to describe the rounding rule is as follows: if the inequality $\sum_i a_i x_i \geq b$ holds and all a_i are integers divisible by $c > 0$, then any integer solution will also satisfy $\sum_i \frac{a_i}{c} x_i \geq \lceil \frac{b}{c} \rceil$. Cutting planes was later proposed as a proof system in [10]. Indeed, it is possible to view the previous optimization process as a sequence of inferences: a new inequality is obtained either as a non-negative linear combination or by a rounding of previously derived inequalities. In a finite number of steps, cutting planes can prove the false inequality "$0 \geq 1$" from an unsatisfiable integer program. For further information about cutting planes refutations and the notion of rank (also called Chvátal rank) we refer the reader to [16, Chapter 19].

Analysing the length of such proofs is a way of studying the running time of integer programming solvers based on the rounding rule. The complexity of cutting planes proofs


© Yuval Filmus, Pavel Hrubeš, and Massimo Lauria;
licensed under Creative Commons License CC-BY
33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 35; pp. 35:1–35:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

has been intensively studied. A lower bound for cutting planes with small coefficients was obtained in [4] and [18], and [15] gave a lower bound for the tree-like version of the system. The strongest result is due to Pudlák [23], who proved that there exists a set of unsatisfiable linear inequalities which require exponential size cutting planes refutations (moreover, the inequalities represent a Boolean formula in conjunctive normal form). His proof is a beautiful example of the so-called "feasible interpolation technique" (used also by [4, 18]), and it required extending monotone Boolean circuit lower bounds of Razborov [24] to the new class of real monotone circuits.

It is interesting that the aforementioned lower bound for tree-like cutting planes works for any kind of deduction rule, no matter how strong. In this paper, we consider the proof system *semantic cutting planes* for which the deduction rule is the following: from any two linear inequalities L_1 and L_2 we can deduce any inequality L which is a sound consequence assuming $\{0,1\}$-assignments. Semantic inferences of similar kind were investigated earlier, in [18, 4, 19, 3]. In [18, 4], Krajíček and (independently) Beame, Pitassi and Raz consider a restricted version of semantic cutting planes in which coefficients are restricted to polynomial size, and prove exponential lower bounds for this restricted version. In [3], Beame, Pitassi and Segerlind consider semantic inferences using polynomial inequalities of degree k. Their results, together with the new lower bounds on communication complexity of disjointness [20, 6, 26], imply exponential lower bounds on the *tree-like* version of such systems – including the tree-like semantic cutting planes.

The semantic system is clearly as strong as *syntactic* cutting planes, and – as we show in this paper – it is in fact stronger. The latter is suggested by the fact that it is coNP-hard to check whether a semantic inference is correct (the subset-sum problem can be stated in terms of just two inequalities). Nevertheless, we show that there exist unsatisfiable inequalities which require exponential semantic cutting planes refutations:

▶ **Theorem 1** (Lower bound). *For every n, there exists an unsatisfiable CNF of polynomial size which requires semantic cutting planes refutations with $2^{n^{\Omega(1)}}$ proof lines.*

As in Pudlák's lower bound, we show that the semantic cutting planes system has feasible interpolation via monotone real circuits. In fact, our proof is a straightforward adaptation of Pudlák's original proof; the changes are all but cosmetic. Second, we prove a separation between the semantic and syntactic version of cutting planes:

▶ **Theorem 2** (Separation). *For every n, there exists an unsatisfiable CNF of polynomial size which has a semantic cutting planes refutation of polynomial size but every syntactic cutting planes refutation has $2^{n^{\Omega(1)}}$ proof lines.*

Theorem 1 is proved in Section 3, Theorem 2 in Section 4. In Section 5, we discuss semantic inferences which can use more than two assumptions. In this context, we come across the following problem: can every real multivariate non-decreasing function be expressed as a composition of non-decreasing real functions in two variables? This is analogous to Hilbert's 13th problem where the same question is posed for algebraic or continuous functions[1].

[1] Although Hilbert expected the answer to be negative, Kolmogorov [17] and Arnold [2] showed that every continuous function of any number of variables can be expressed as a composition of continuous functions of two variables.

2 Preliminaries

A (linear) inequality in variables $x_1, \ldots, x_n$ is an expression of the form

$$a_1 x_1 + \cdots + a_n x_n \geq b\,, \quad \text{with } a_1, \ldots, a_n, b \in \mathbb{Z}\,.$$

We say that a $0,1$-assignment $\sigma \in \{0,1\}^n$ *satisfies* the inequality, if $\sum_{i=1}^n a_i \sigma_i \geq b$. A set of inequalities $\mathcal{L}$ is called *satisfiable*, if there exists a $0,1$-assignment which satisfies every inequality in $\mathcal{L}$.

As is customary, we will often use inequalities with "$\leq$" instead of "$\geq$", or with constants and variables appearing on both sides of the inequality. In this case, we identify $\sum_i a_i x_i \leq b$ with $\sum_i -a_i x_i \geq -b$, and $\sum_i a_i x_i + b \geq \sum_i a'_i x_i + b'$ with $\sum_i (a_i - a'_i) x_i \geq b' - b$, etc.

We now describe the two systems for refuting unsatisfiable linear inequalities.

Syntactic Cutting Planes

Let $\mathcal{L}$ be a set of inequalities. *A syntactic cutting planes proof* of an inequality L from $\mathcal{L}$ is a sequence of inequalities $L_1, \ldots, L_m$ such that $L_m = L$, and for every $i \in \{1, \ldots, m\}$,

1. $L_i \in \mathcal{L}$, or it is a *Boolean axiom*

   $$x \geq 0\,, -x \geq -1\,,$$

   where x is a variable , or
2. there exist $j_1, j_2 < i$ such that L_i is obtained from L_{j_1}, L_{j_2} by means of the following rules:

   (Sum) $$\frac{\sum_i a_i x_i \geq b\,, \qquad \sum_i a'_i x_i \geq b'}{\sum_i (\alpha a_i + \beta a'_i) x_i \geq \alpha b + \beta b'}\,, \qquad \text{for } \alpha, \beta \in \mathbb{N},$$

   (Division) $$\frac{\sum_i a_i x_i \geq b}{\sum_i \frac{a_i}{c} x_i \geq \left\lceil \frac{b}{c} \right\rceil}\,, \qquad \text{when } 0 < c \in \mathbb{N} \text{ divides all } a_i.$$

The division rule is only valid for integer values of x_i, so it may cut away unwanted fractional solutions.

The *length* of the proof is m, i.e., the number of proof lines. A syntactic cutting planes *refutation of* $\mathcal{L}$ is a proof of $0 \geq b$ from $\mathcal{L}$, where b is any positive integer.

Semantic Cutting Planes

Semantic cutting planes proofs and refutations are defined as above, except we can use the rule

$$\frac{L',\ L''}{L'''}\,,$$

where L', L'' and L''' are such that L''' semantically follows from L' and L'': every $0,1$-assignment which satisfies both L' and L'' satisfies also L'''. Note that the Boolean axioms semantically follow from any inequality and do not have to be introduced separately.

Clearly, a syntactic cutting planes proof is automatically also a semantic cutting planes proof. Semantic inference is very powerful, and there is no efficient way to verify of even

witness its soundness, unless NP = coNP. Observe that the equation $\sum a_i x_i = b$ has no $0,1$-solution iff the following is a correct semantic inference:

$$\frac{\sum_i a_i x_i \geq b \qquad \sum_i a_i x_i \leq b}{0 \geq 1}.$$

However, deciding if $\sum_i a_i x_i = b$ has a $0,1$-solution is the NP-hard subset-sum problem. This shows that semantic cutting planes is not a proof system in the sense of Cook and Reckhow [9] (unless P = NP), who require proofs to be efficiently verifiable.

Krajíček [18] and Beame, Pitassi and Raz [4] consider a restricted version of semantic cutting planes in which all lines have polynomially bounded coefficients. Such a semantic inference can be checked in polynomial time using dynamic programming, and so is a proof system in the sense of Cook and Reckhow.

Size of Coefficients

We measure the complexity of a proof in terms of the number of inferences. However, the coefficients in the linear inequalities can be quite large and the bit representation of a proof can be much larger than the number of proof lines. Fortunately, Buss and Clote [5] proved that any syntactic cutting planes refutation can be transformed into another one in which the coefficients are at most exponential in the number of variables. Hence, each coefficient can be represented with a linear number of bits. For semantic cutting planes, we can use a more general argument: every threshold function over $\{0,1\}^n$ can be represented as a linear inequality with coefficients of bit length $O(n \log n)$ [22]. This also means that in semantic cutting planes, we can use arbitrary real coefficients instead of integer coefficients, without changing the strength of the system.

Syntactic Simulation of Semantic Inferences

Syntactic cutting planes is a complete proof system, as shown by Chvátal [7]. Results of Chvátal, Cook and Hartmann [8] and Eisenbrand and Schulz [11] show that any semantic cutting planes inference, even with an unbounded number of premises, can be simulated by a syntactic cutting planes proof of length $\exp \tilde{O}(n^2)$. This simulation is general but very inefficient. One of the main results of this paper is that an efficient simulation does not exist.

Propositional Logic and CNF Encoding

In proof complexity, we are chiefly interested in refutations of propositional formulas, more specifically formulas in conjunctive normal form. Given a CNF A, we can represent it as a set of linear inequalities $\mathcal{L}_A$ as follows. A disjunction such as $x_1 \vee \neg x_2 \vee \neg x_3$ is represented as the inequality $x_1 + (1 - x_2) + (1 - x_3) \geq 1$ (or rather, $x_1 - x_2 - x_3 \geq -1$), and $\mathcal{L}_A$ consists of all the inequalities corresponding to the clauses in A. Clearly, an assignment satisfies A iff it satisfies $\mathcal{L}_A$. We will refer to $\mathcal{L}_A$ as the *standard encoding of* A. This allows us to talk about cutting planes refutations of CNFs: a refutation of A is a refutation of the standard encoding of A.

3 Feasible Interpolation for Semantic Cutting Planes

In this section, we prove Theorem 1. This is achieved by showing that semantic cutting planes have feasible interpolation via monotone real circuits, as was shown in [23] for syntactic cutting planes.

Let X, Y_1, Y_2 be disjoint sets of variables with $X = \{x_1, \ldots, x_n\}$. An inequality L of the form $U \geq b$ in the variables $X \cup Y_1 \cup Y_2$ can be uniquely written as $U^x + U^{y_1} + U^{y_2} \geq b$, where U^x, U^{y_1} and U^{y_2} depend only on the variables X, Y_1, Y_2, respectively. If $\sigma \in \{0,1\}^n$ is an assignment to the variables X, $L(\sigma)$ will denote the inequality

$$U^{y_1} + U^{y_2} \geq b - U^x(\sigma)\,. \tag{1}$$

Let $\mathcal{L}_1 = \{L_1, \ldots, L_p\}$ and $\mathcal{L}_2 = \{L'_1, \ldots, L'_q\}$ be two sets of inequalities, such that every inequality in $\mathcal{L}_1$ depends only the variables $X \cup Y_1$, and every inequality in $\mathcal{L}_2$ depends only the variables $X \cup Y_2$. We assume that the sets $\mathcal{L}_1$ and $\mathcal{L}_2$ are contradictory: no assignment satisfies $\mathcal{L}_1 \cup \mathcal{L}_2$. We say that a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ *interpolates* $\mathcal{L}_1$ and $\mathcal{L}_2$, if for every $\sigma \in \{0,1\}^n$

1. if $f(\sigma) = 0$ then the set $\mathcal{L}_1(\sigma) = \{L_1(\sigma), \ldots, L_p(\sigma)\}$ is unsatisfiable, and
2. if $f(\sigma) = 1$ then the set $\mathcal{L}_2(\sigma) = \{L'_1(\sigma), \ldots, L'_q(\sigma)\}$ is unsatisfiable.

Recall the definition of monotone real circuit from [23]. A monotone real circuit C computes a nondecreasing function $f\colon \mathbb{R}^n \to \mathbb{R}$. A gate can be *any* nondecreasing function $\mathbb{R} \to \mathbb{R}$ or $\mathbb{R}^2 \to \mathbb{R}$. If $f(\{0,1\}^n) \subseteq \{0,1\}$, C is said to compute the Boolean function $f|_{\{0,1\}^n}$. Clearly, the Boolean function must be monotone.

We will prove the following:

▶ **Theorem 3.** *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be as above. Assume that the variables X have non-positive coefficients in every inequality in $\mathcal{L}_2$ (or non-negative coefficients in $\mathcal{L}_1$), and that $\mathcal{L}_1 \cup \mathcal{L}_2$ has a semantic cutting planes refutation with m proof lines. Then there exists a Boolean function which interpolates $\mathcal{L}_1$ and $\mathcal{L}_2$ and which can be computed by a monotone real circuit of size $O(m + (p+q)n)$.*

Fortunately, Pudlák has also provided an exponential lower bound on the size of real monotone circuits interpolating the "clique versus coloring" tautologies.

▶ **Theorem 4** ([23])**.** *Let $f\colon \{0,1\}^{\binom{n}{2}} \to \{0,1\}$ be a monotone Boolean function which rejects all $k-1$-colorable graphs and accepts all graphs with a k-clique, with $k = \lceil (n/\log n)^{2/3}/8 \rceil$. Then every monotone real circuit computing f has size $2^{\Omega(n/\log n)^{1/3}}$.*

In order to deduce a lower bound on semantic cutting planes from Theorem 3 and Theorem 4, it is enough to find suitable formulas Color_n and Clique_n expressing that an n-vertex graph is $(k-1)$-colorable, and that it has a k-clique, respectively. We write them down for completeness.

The formula Clique_n is a conjunction of the following clauses (k is the parameter from the theorem):

1. $\bigvee_{i \in [n]} y_{j,i}$, for every $j \in [k]$, $\neg y_{j_1,i} \vee \neg y_{j_2,i}$, for every $j_1 \neq j_2 \in [k]$, $i \in [n]$,
2. $\neg y_{j_1,i_1} \vee \neg y_{j_2,i_2} \vee x_{i_1,i_2}$, for every $j_1 \neq j_2 \in [k], i_1 < i_2 \in [n]$.

Color_n is a conjunction of the following clauses:

1. $\bigvee_{j \in [k-1]} z_{i,j}$, for every $i \in [n]$, $\neg z_{i,j_1} \vee \neg z_{i,j_2}$, for every $i \in [n]$, $j_1 \neq j_2 \in [k-1]$,
2. $\neg z_{i_1,j} \vee \neg z_{i_2,j} \vee \neg x_{i_1,i_2}$, for every $j \in [k-1], i_1 < i_2 \in [n]$.

The formulas are in variables $X = \{x_{i_1,i_2} : i_1 < i_2 \in [n]\}$, $Y = \{y_{j,i} : j \in [k], i \in [n]\}$, $Z = \{z_{i,j} : i \in [n], j \in [k-1]\}$. We think of X as representing edges of an n-vertex graph, Y as picking a clique in the graph, and Z as defining a coloring of the graph.

▶ **Corollary 5.** *Every semantic cutting planes refutation of $\mathit{Clique}_n \wedge \mathit{Color}_n$ has at least $2^{\Omega((n/\log n)^{1/3})}$ lines.*

Proof. The particular formulation of the clique and color formulas is quite irrelevant. It matters that, first, the variables X occur only positively in Clique_n (and only negatively in Color_n), and, second, that every interpolant of Clique_n and Color_n must reject on $(k-1)$-colorable graphs and accept on graphs with k-clique. ◀

Proof of Theorem 3

Let us first imagine that $X = \emptyset$. That is, the sets of inequalities $\mathcal{L}_1$ and $\mathcal{L}_2$ depend on disjoint sets of variables Y_1 and Y_2, respectively. Assume we have a refutation R of $\mathcal{L}_1 \cup \mathcal{L}_2$ with m proof lines. This means that at least one of $\mathcal{L}_1$ or $\mathcal{L}_2$ is unsatisfiable. We will prove a stronger statement, that at least one of $\mathcal{L}_1, \mathcal{L}_2$ has a refutation with m proof lines:

▶ **Claim 6.** *There exists $e \in \{1,2\}$ and a refutation R_e of $\mathcal{L}_e$ with m proof lines.*

Proof. Let R be the sequence $U_1 \geq b_1, \ldots, U_m \geq b_m$ with $U_m = 0$ and b_m positive. For $e \in \{1,2\}$ Let R_e be the sequence of inequalities

$$U_1^{y_e} \geq c_1^e, \ldots, U_m^{y_e} \geq c_m^e\,,$$

where the constants $c_1^e, \ldots, c_m^e$ are defined as follows:

1. if $(U_i \geq b_i) \in \mathcal{L}_e$, let $c_i^e := b_i$, else
2. if $(U_i \geq b_i) \in \mathcal{L}_{e'}$ for $e' \neq e$, let $c_i^e := 0$, else
3. if $U_i \geq b_i$ semantically follows from $U_{j_1} \geq b_{j_1}$ and $U_{j_2} \geq b_{j_2}$ with $j_1, j_2 < i$, then c_m^e is the largest possible integer such that $U_{j_1}^{y_e} \geq c_{j_1}^e$ and $U_{j_2}^{y_e} \geq c_{j_2}^e$ imply $U_i^{y_e} \geq c_i^e$. In symbols,

$$c_i^e := \min\{U_i^{y_e}(\rho) : \rho \in \{0,1\}^{|Y_e|}, U_{j_1}^{y_e}(\rho) \geq c_{j_1}^e, U_{j_2}^{y_e}(\rho) \geq c_{j_2}^e\}\,.$$

   If the minimum is over the empty set, let $c_i^e := \infty$ (or rather, a fixed but large enough real number).

The construction guarantees that

- **(a)** for $e \in \{1,2\}$, R_e is a correct proof of $0 \geq c_m^e$ from $\mathcal{L}_e$, and
- **(b)** for every $i \in \{1, \ldots, m\}$, $c_i^1 + c_i^2 \geq b_i$, unless $U_i \geq b_i$ is *vacuous*: i.e., $U_i = 0$ and b_i is negative.

The statement (a) is true by definition. Part (b) is proved by induction on $i \in \{1, \ldots, m\}$. In case 1 and case 2 equality holds, except when $(U_i \geq b_i) \in \mathcal{L}_1 \cap \mathcal{L}_2$. Then $U_i = 0$ and $c_i^1 = c_i^2 = b_i$, and so $c_i^1 + c_i^2 = 2b_i$. Hence $c_i^1 + c_i^2 \geq b_i$ unless b_i is negative, in which case $U_i \geq b_i$ is indeed vacuous. For case 3, the non-trivial case is when none of $U_i \geq b_i, U_{j_1} \geq b_{j_1}, U_{j_2} \geq b_{j_2}$ is vacuous and $c_i^1, c_i^2 < \infty$. Then there exist $\rho_1 \in \{0,1\}^{|Y_1|}$ and $\rho_2 \in \{0,1\}^{|Y_2|}$ such that $c_i^1 = U_i^{y_1}(\rho_1)$ and $c_i^2 = U_i^{y_2}(\rho_2)$, and

$$U_{j_1}^{y_1}(\rho_1) \geq c_{j_1}^1,\ U_{j_2}^{y_1}(\rho_1) \geq c_{j_2}^1\,,$$
$$U_{j_1}^{y_2}(\rho_2) \geq c_{j_1}^2,\ U_{j_2}^{y_2}(\rho_2) \geq c_{j_2}^2\,.$$

Since $c_{j_1}^1 + c_{j_1}^2 \geq b_{j_1}$ and $c_{j_2}^1 + c_{j_2}^2 \geq b_{j_2}$, we have

$$U_{j_1}^{y_1}(\rho_1) + U_{j_1}^{y_2}(\rho_2) \geq b_{j_1}\,,\ \text{and}\ U_{j_2}^{y_1}(\rho_1) + U_{j_2}^{y_2}(\rho_2) \geq b_{j_2}\,.$$

Since $U_i \geq b_i$ semantically follows from $U_{j_1} \geq b_{j_1}$ and $U_{j_2} \geq b_{j_2}$, we have

$$b_i \leq U_i^{y_1}(\rho_1) + U_i^{y_2}(\rho_2) = c_i^1 + c_i^2\,.$$

Finally, $b_m > 0$ and (b) show that either c_m^1 or c_m^2 is positive, and hence R_1 is a refutation of $\mathcal{L}_1$, or R_2 is a refutation of $\mathcal{L}_2$. ◀

To prove the theorem, the main observation is that in case 3, c_i is a non-decreasing function of c_{j_1} and c_{j_2}: increasing c_{j_1} or c_{j_2} means that in case 3, the minimum is taken over a smaller set.

Let $\mathcal{L}_1$, $\mathcal{L}_2$ be as in the statement of the theorem, and R a refutation of $\mathcal{L}_1 \cup \mathcal{L}_2$ with m lines. For an assignment σ to the variables X, let $R(\sigma)$ be the refutation obtained by replacing every line L in R by $L(\sigma)$. It is indeed a correct refutation of $\mathcal{L}_1(\sigma) \cup \mathcal{L}_2(\sigma)$, where the two sets now have disjoint variables. Let R_1^σ, R_2^σ be the two proofs constructed in the Claim, and consider c_m^1 and c_m^2 as functions of σ. By (a), if $c_m^2(\sigma) > 0$ then R_2^σ is a refutation of $\mathcal{L}_2(\sigma)$ and so $\mathcal{L}_2(\sigma)$ is unsatisfiable. If $c_m^2(\sigma) \leq 0$ then, by (b), $c_m^1(\sigma) > 0$ and so $\mathcal{L}_1(\sigma)$ is unsatisfiable. In other words, if we define the Boolean function f by

$$f(\sigma) = 1 \quad \text{iff} \quad c_m^2(\sigma) > 0\,,$$

then f interpolates $\mathcal{L}_1$ and $\mathcal{L}_2$. Moreover, if X have non-positive coefficients in $\mathcal{L}_2$, the function f can be computed by a monotone real circuit with $O(m + pn)$ gates. This is because in case 1, $c_i^2(\sigma)$ is a linear function with non-negative coefficients (in (1), $U^x(\sigma)$ is moved to the right hand side), in case 2, it is a constant, and in case 3, c_i^2 is a non-decreasing function of $c_{j_1}^2$ of $c_{j_2}^2$.

4 Separation Between Semantic and Syntactic Cutting Planes

In this section, we separate semantic and syntactic cutting planes, proving Theorem 2. In order to do that, we modify the "clique versus coloring" contradiction in such a way that any refutation in syntactic cutting planes must remain long, while there is a short refutation in semantic cutting planes. The main observation is that systems of unsatisfiable linear *equations* have short semantic refutations. Hence, it will be enough to restate the "clique versus coloring" as a set of linear equations, in a way that its hardness for syntactic proofs is preserved.

4.1 Equations in Cutting Planes

In the following, we will allow cutting planes to use linear equations as well as inequalities. Formally, we will treat an equation $U = b$ as a pair of inequalities $U \geq b$ and $U \leq b$. Hence, a refutation of a set of equations or inequalities is understood as a refutation of the underlying set of inequalities.

▶ **Proposition 7.** *If a set of m linear equations is unsatisfiable then it has a semantic cutting planes refutation with $O(m)$ lines.*

Proof. Assume that the equations are $\sum_i a_{j,i} x_i = b_j$, $j \in [m]$. Let $M = 1 + \max_j\{|b_j| + \sum_i |a_{j,i}|\}$. From the given equations we can deduce the following equation:

$$\sum_{j=1}^{m} \left(\sum_i a_{j,i} x_i - b_j \right) M^{j-1} = 0\,, \tag{2}$$

using only integer scalar multiplications and sums, by separately deriving the two corresponding inequalities $0 \leq \sum_{j=1}^m \left(\sum_i a_{j,i} x_i - b_j\right) M^{j-1} \leq 0$. The equation is unsatisfiable (exercise). Hence, we can deduce $0 \geq 1$ from (2) in a single step of semantic refutation. ◀

The next proposition shows that a pair of inequalities $b \leq U \leq b + 1$ can be replaced by a single equality $U = b + \sigma$, where σ is a fresh variable, without changing length of syntactic proofs.

▶ **Proposition 8.** *Let $\mathcal{L} = \mathcal{L}_0 \cup \{\sum a_i x_i \geq b\}$ be a set of linear inequalities such that $\sum_i a_i x_i \leq b+1$ has a syntactic proof of length s from $\mathcal{L}$. Let*

$$\mathcal{L}' = \mathcal{L}_0 \cup \{\textstyle\sum_i a_i x_i = b + \sigma\},$$

where σ is a variable not appearing in $\mathcal{L}$. In syntactic cutting planes, the lengths of the shortest refutations of $\mathcal{L}'$ and $\mathcal{L}$ differ at most by an additive term of $O(s)$.

Proof. Consider a refutation of $\mathcal{L}$. We want to get a refutation of $\mathcal{L}'$ of similar length. The only missing axiom in $\mathcal{L} \setminus \mathcal{L}'$ is $\sum_i a_i x_i \geq b$, which can be derived from $\sum_i a_i x_i \geq b + \sigma$ and $\sigma \geq 0$, the former being an axiom in $\mathcal{L}'$ and the latter a Boolean axiom.

In the opposite direction, start with a refutation R of $\mathcal{L}'$ with r lines and consider the substitution $\sigma \mapsto \sum_i a_i x_i - b$ applied to its lines. After this substitution, we construct a refutation of $\mathcal{L}$ with $r + s$ lines.

Axioms not mentioning σ stay the same. The substitution in the remaining axioms is

$$\begin{array}{lcl} \sum_i a_i x_i = b + \sigma & \mapsto & 0 = 0; \\ \sigma \geq 0 & \mapsto & \sum_i a_i x_i \geq b; \\ \sigma \leq 1 & \mapsto & \sum_i a_i x_i \leq b + 1; \end{array}$$

where the second is an axiom in $\mathcal{L}$ and the third has a derivation with s lines. After the substitution, the sum of two lines and the product by a scalar remain correct inference steps. For the division step, consider $0 < c \in \mathbb{N}$ and the inference

$$\frac{\sum_i c a'_i x_i + cp\sigma \geq q}{\sum_i a'_i x_i + p\sigma \geq \lceil \frac{q}{c} \rceil}.$$

The assumption and the conclusion of the rule are transformed as

$$\begin{array}{lcllcl} \sum c a'_i x_i + cp\sigma \geq q & \mapsto & \sum c a'_i x_i + cp(\sum a_i x_i - b) \geq q & \equiv & \sum (c a'_i + cp a_i) x_i \geq q + cpb. \\ \sum a'_i x_i + p\sigma \geq \lceil \frac{q}{c} \rceil & \mapsto & \sum a'_i x_i + p(\sum_i a_i x_i - b) \geq \lceil \frac{q}{c} \rceil & \equiv & \sum (a'_i + p a_i) x_i \geq \lceil \frac{q}{c} \rceil + pb. \end{array}$$

Since b is an integer, $\left\lceil \frac{q+cpb}{c} \right\rceil = \lceil \frac{q}{c} \rceil + pb$, and so substitution after rounding is the same as rounding after substitution. ◀

4.2 The Separating Formula

Let A be a CNF

$$A = \bigwedge_{i \in [k]} (u_{i,1} \vee \cdots \vee u_{i,m_i}), \tag{3}$$

where each $u_{i,j}$ is a literal, i.e., a variable or its negation. We will define three reformulations of A: $T(A)$, $S(A)$ and $F(A)$, where $T(A)$ is a set of equations and inequalities, $S(A)$ is a set of equations only, and $F(A)$ is the CNF corresponding to $S(A)$. It is the last CNF which is used in the separation between semantic and syntactic proofs.

$T(A)$ and $S(A)$

For every $i \in [k], j \in [m_i]$, introduce a new variable $\eta_{i,j}$. Then $T(A)$ is the union, over all $i \in [k]$ and $j \in [m_i]$, of the following:

$$\eta_{i,1} + \cdots + \eta_{i,m_i} = 1, \tag{4}$$

$$u_{i,j} - \eta_{i,j} \geq 0. \tag{5}$$

In (5) we identify $\neg x$ with $1 - x$, if $u_{i,j}$ is the literal $\neg x$ in A. Furthermore, let $S(A)$ be the set of equations obtained by replacing every inequality in (5) by the equation

$$u_{i,j} - \eta_{i,j} = \sigma_{i,j}\,, \tag{6}$$

where $\sigma_{i,j}$ are fresh variables.

It is easy to see that A is unsatisfiable iff $T(A)$ is unsatisfiable iff $S(A)$ is unsatisfiable. Moreover, we note that:

▶ **Lemma 9.** *Let A be an unsatisfiable CNF as in (3), with $m := \max m_i$. Then:*

1. *$S(A)$ has a semantic refutation with $O(mk)$ lines.*
2. *If $S(A)$ has a syntactic refutation with s lines, then $T(A)$ has a syntactic refutation with $s + O(mk)$ lines.*
3. *If A is the "clique versus coloring" CNF as in Section 3, then every semantic (hence, also syntactic) refutation of $T(A)$ requires $2^{\Omega((n/\log n)^{1/3})}$ lines.*

Proof. Item 1 follows from Proposition 7, since $S(A)$ is an unsatisfiable set of equations.

Item 2. For every inequality $u_{i,j} - \eta_{i,j} \geq 0$ in (5), the inequality $u_{i,j} - \eta_{i,j} \leq 1$ has a constant size syntactic proof. Hence, we can apply Proposition 8 to eliminate the variables $\sigma_{i,j}$.

Item 3 follows from Theorem 3 and Theorem 4 in the same manner as Corollary 5. In $Clique_n$, the variables $X = \{x_{i,j} : i < j \in [n]\}$ occur only positively. Hence, in the translation $T(Clique_n)$, they have only non-negative coefficients (similarly, X have non-positive coefficients in $T(Color_n)$). ◀

Lemma 9 already implies that for the "clique versus color" contradiction, $S(A)$ has a polynomial size semantic refutation, whereas it requires an exponential size syntactic refutation. However, $S(A)$ is a system of linear inequalities rather than a CNF. In order to fix this, we define the CNF $F(A)$, equivalent to $S(A)$.

The Formula $F(A)$

For three variables u, η, σ, let $\Gamma(u, \eta, \sigma)$ be the CNF expressing that $u - \eta = \sigma$; i.e., Γ is satisfied by $u, \eta, \sigma \in \{0, 1\}$ iff $u - \eta = \sigma$. Let A be a CNF as in (3). Then $F(A)$ is the conjunction, over all $i \in [k]$ and $j \in [m_i]$, of

$$\eta_{i,1} \vee \cdots \vee \eta_{i,m_i}\,, \neg\eta_{i,j_1} \vee \neg\eta_{i,j_2}\,, \text{ for every } j_1 \neq j_2 \in [m_i]\,, \tag{7}$$

$$\Gamma(u_{i,j}, \eta_{i,j}, \sigma_{i,j})\,. \tag{8}$$

▶ **Lemma 10.** *$S(A)$ and the standard cutting planes encoding of $F(A)$ mutually deduce each other with a polynomial length syntactic cutting planes derivation.*

Proof. Equation (6) has a constant size proof from the encoding of (8) and vice versa, because syntactic cutting planes is a complete proof system[2]. The encoding of (7) is the set of inequalities

$$\eta_{i,1} + \cdots + \eta_{i,m_i} \geq 1\,,\ \eta_{i,j_1} + \eta_{i,j_2} \leq 1\,,\ \text{for every } j_1 \neq j_2 \in [m_i]\,.$$

Comparing this with (4), it is enough to show:

[2] Completeness means in this case that if an inequality L semantically follows from inequalities $L_1, \ldots, L_n$, then this can be proved in syntactic cutting planes. This is a standard fact proved in [13, 7].

▶ **Claim 11** ([25]). *The inequality* $\sum_{i=1}^{n} x_i \leq 1$ *has a polynomial size syntactic cutting planes proof from the inequalities* $\{x_i + x_j \leq 1 : i < j \in [n]\}$. *The opposite direction also holds.*

Proof. For the forward direction, we prove by induction on $l-k$ that $\sum_{i=k}^{l} x_i \leq 1$. The cases $l-k \leq 2$ follow directly from the axioms. For $l-k > 2$, consider the sum of $\sum_{i=k}^{l-1} x_i \leq 1$, $\sum_{i=k+1}^{l} x_i \leq 1$ and $x_k + x_l \leq 1$, which is $\sum_{i=k}^{l} 2x_i \leq 3$. A division step concludes the proof.

The opposite direction is easier: given $\sum_{i=1}^{n} x_i \leq 1$ and two indices k and l, we can add $-x_i \leq 0$ for each $i \notin \{k, l\}$ to get $x_k + x_l \leq 1$. ◀

This completes the proof of Lemma 10. ◀

We are now ready to prove Theorem 2. Recall the "clique versus coloring" formulas in Section 3.

▶ **Theorem 12.** *The CNF formula* $F(Clique_n \wedge Color_n)$ *has a polynomial size semantic cutting planes refutation whereas every syntactic cutting planes refutation requires* $2^{\Omega(n/\log n)^{1/3}}$ *lines.*

Proof. The upper bound follows from Lemma 10 and part 1 of Lemma 9. The lower bound follows from Lemma 10 and parts 2 and 3 of Lemma 9. ◀

Inspecting Proposition 7, this also implies the following:

▶ **Corollary 13.** *There exists an equation* $\sum_{i=1}^{n} a_i x_i = b$ *which has no* $0,1$*-solution, the bit representation of* $a_1, \ldots, a_n, b$ *is polynomial in* n*, and every syntactic cutting planes refutation of* $\sum_{i=1}^{n} a_i x_i \geq b, \sum_{i=1}^{n} a_i x_i \leq b$ *requires* $2^{n^{\Omega(1)}}$ *lines.*

Proof. Consider the set of equations $S(Clique_n \wedge Color_n)$. The proof of Proposition 7 shows how to derive (2), which has no $0,1$-solution, using a syntacting cutting planes derivation of polynomial size. On the other hand, parts 2 and 3 of Lemma 9 imply that every syntactic cutting planes refutation of (2) requires exponentially many lines. ◀

Observe that in the upper bounds of Theorem 12 and Proposition 7, it is crucial that we use exponentially large coefficients. A natural open problem is the following:

▶ **Open Problem 14.** *Is it possible for syntactic cutting planes to polynomially simulate semantic cutting planes proofs with coefficients which are at most polynomial in the number of variables?*

Notice that subset-sum problem with such small coefficients can be solved in polynomial time by dynamic programming.

5 Inferences with Higher Fan-In and Hilbert's 13th Problem

In the definition of semantic cutting planes, we assumed that in a refutation of $\mathcal{L}$, every line is either an element of $\mathcal{L}$ or follows from at most *two* previously proved inequalities. But why not *three* or a *hundred* inequalities? For a fixed $k \in \mathbb{N}$, define a *k-semantic cutting planes refutation of $\mathcal{L}$* (k-SCP refutation, for short), as a refutation in which every line $L_i \notin \mathcal{L}$ semantically follows from some $L_{j_1}, \ldots, L_{j_k}$, with $j_1, \ldots, j_k < i$. The obvious question is whether increasing k makes the proof system more powerful:

▶ **Open Problem 15.** *For* $2 \leq k_1 < k_2$*, can we simulate* k_2*-semantic cutting planes by* k_1*-semantic cutting planes? More exactly, is there a polynomial* p*, such that whenever* $\mathcal{L}$ *has a* k_2*-SCP refutation with* m *proof lines, then it has a* k_1*-SCP refutation with at most* $p(m)$ *proof lines?*

We do not know an answer to this question. On the other hand, we note that Theorem 3 and Corollary 5 can be extended to k-semantic refutations:

- Theorem 3 holds for k-SCP refutations, if we allow monotone real circuits to use non-decreasing k-ary functions as gates.
- Pudlák's lower bound works for monotone real circuits with k-ary gates, for any fixed k.
- Hence Corollary 5 holds also for k-SCP refutations, giving an exponential lower bound on the number of proof lines.

In this context, we come across a related question, which is arguably much more interesting as a mathematical problem:

▶ **Open Problem 16.** *Can every multivariate* non-decreasing *real function be expressed as a composition of* non-decreasing *unary or binary functions?*

In other words, we want to know whether every non-decreasing function can be computed by a monotone real circuit, with gates of fan-in at most two. If this is the case, there must also exist a function $\lambda\colon \mathbb{N} \to \mathbb{N}$ such that every non-decreasing n-ary function is computable by a monotone real circuit of size at most $\lambda(n)$.[3] This would mean that we can simulate any monotone real circuit with k-ary gates by a monotone real circuit with binary gates, with at most a factor $\lambda(k)$ loss in size.

Problem 16 is reminiscent of the solution to Hilbert's 13th Problem due to Arnold and Kolmogorov [17, 2]. They have shown that every multivariate *continuous* function can be expressed as a composition of unary and binary *continuous* functions (see [21, Chapter 11]). In fact, the only binary function needed is addition: any continuous function can be expressed in terms of addition and several unary continuous functions. This is rather surprising; Hilbert's 13th problem tacitly assumes that such a representation of continuous functions is impossible. Moreover, such a representation is indeed impossible for many other classes of functions: there exists an analytic function in three variables which cannot be expressed in terms of analytic functions of two variables; similarly for infinitely differentiable or entire functions (see [1] for further references).

Acknowledgments. This paper subsumes and includes the results of the two manuscripts [12, 14].

Part of this research was done while the first author was visiting KTH Royal Institute of Technology (Stockholm, Sweden) and while he was a member of the Institute for Advanced Study (Princeton, NJ), and while the third author was at KTH Royal Institute of Technology.

The first author received funding from the European Union's Seventh Framework Programme (FP7/2007–2013) under grant agreement no 238381, and from the National Science Foundation under agreement No. DMS-1128155. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors, and do not necessarily reflect the views of the National Science Foundation. The second author was supported by the European Research Council under the European Union's Seventh Framework Program (FP7/2007-2013) / ERC grant agreement no. 339691. The third author was supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no 279611.

[3] Hint: for a fixed n, assume that for every k there exists an n-ary non-decreasing function f_k which cannot be computed by a monotone real circuit of size k. Then we can "amalgamate" the functions $f_1, f_2, \ldots$ into a single $(n+1)$-ary non-decreasing function, which cannot be computed by a monotone real circuit of any size.

References

1. Shigeo Akashi and Satoshi Kodama. A version of Hilbert's 13th problem for infinitely differentiable functions. *Fixed point theory and applications*, 2010.
2. Vladimir N. Arnold. On functions of three variables. *Doklady Akad. Nauk SSSR*, 114:679–681, 1957.
3. Paul Beame, Toniann Pitassi, and Nathan Segerlind. Lower bounds for Lovász-Schrijver systems and beyond follow from multiparty communication complexity. *SIAM J. Comput.*, 37(3):845–869, June 2007.
4. Maria Luisa Bonet, Toniann Pitassi, and Ran Raz. Lower bounds for cutting planes proofs with small coefficients. *The Journal of Symbolic Logic*, 62(3):708–728, 1997. URL: `http://www.jstor.org/stable/2275569`.
5. Samuel R. Buss and Peter Clote. Cutting planes, connectivity, and threshold logic. *Archive for Mathematical Logic*, 35(1):33–62, 1996.
6. Arkadev Chattopadhyay and Anil Ada. Multiparty communication complexity of disjointness. *Electronic Colloquium on Computational Complexity (ECCC)*, 15:2, 208.
7. Vašek Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4(4):305–337, 1973.
8. Vašek Chvátal, William Cook, and Mark Hartmann. On cutting-plane proofs in combinatorial optimization. *Linear Algebra and its Applications*, 114:455–499, 1989.
9. Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36–50, 1979.
10. William Cook, Collette R. Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, 1987.
11. Friederich Eisenbrand and Andreas S. Schulz. Bounds on the Chvátal rank of polytopes in the 0/1-cube. *Integer Programming and Combinatorial Optimization*, pages 137–150, 1999.
12. Yuval Filmus and Massimo Lauria. A separation between semantic and syntactic cutting planes. Manuscript, 2013.
13. Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.
14. Pavel Hrubeš. A note on semantic cutting planes. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:128, 2013. URL: `http://eccc.hpi-web.de/report/2013/128`.
15. Russell Impagliazzo, Toniann Pitassi, and Alasdair Urquhart. Upper and lower bounds for tree-like cutting planes proofs. In *Logic in Computer Science, 1994. LICS'94. Proceedings., Symposium on*, pages 220–228. IEEE, 1994.
16. Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*. Springer-Verlag, 2012.
17. Andrey N. Kolmogorov. On the representations of continuous functions of several variables by superpositions of continuous functions of fewer variables. *Doklady Akad. Nauk SSSR*, 108:179–182, 1956.
18. Jan Krajíček. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *Journal of Symbolic Logic*, 62(2):457–486, 1997.
19. Jan Krajíček. Interpolation and approximate semantic derivations. *Mathematical Logic Quarterly*, 48(4):602–606, 2002.
20. Troy Lee and Adi Shraibman. Disjointness is hard in the multi-party number-on-the-forehead model. *2012 IEEE 27th Conference on Computational Complexity*, 0:81–91, 2008. `doi:http://doi.ieeecomputersociety.org/10.1109/CCC.2008.29`.
21. G. G. Lorentz. *Approximations of functions*. Holt, Rinehart and Winston, New York, 1966.
22. Saburo Muroga, Iwao Toda, and Satoru Takasu. Theory of majority decision elements. *Journal of the Franklin Institute*, 271(5):376–418, 1961.

23 Pavel Pudlák. Lower bounds for Resolution and Cutting Plane proofs and monotone computations. *Journal of Symbolic Logic*, 62(3):981–998, 1997.

24 Alexander A. Razborov. Lower bounds on the monotone complexity of some Boolean functions. *Doklady Akad. Nauk SSSR*, 282:1033–1037, 1985.

25 Martin Rhodes. On the Chvátal rank of the pigeonhole principle. *Theoretical Computer Science*, 410(27-29):2774–2778, 2009.

26 Alexander A. Sherstov. The multiparty communication complexity of set disjointness. In *STOC*, pages 525–548, 2012.

Editing to Connected f-Degree Graph*

Fedor V. Fomin[1], Petr Golovach[1], Fahad Panolan[3], and Saket Saurabh[4]

1 **Department of Informatics, University of Bergen, Norway**
fomin@ii.uib.no
1 **Department of Informatics, University of Bergen, Norway**
petr.golovach@ii.uib.no
3 **The Institute of Mathematical Sciences, India**
fahad@imsc.res.in
4 **Department of Informatics, University of Bergen, Norway; and The Institute of Mathematical Sciences, India**
saket@imsc.res.in

Abstract

In the Edge Editing to Connected f-Degree Graph problem we are given a graph G, an integer k and a function f assigning integers to vertices of G. The task is to decide whether there is a connected graph F on the same vertex set as G, such that for every vertex v, its degree in F is $f(v)$ and the number of edges in $E(G) \triangle E(F)$, the symmetric difference of $E(G)$ and $E(F)$, is at most k. We show that Edge Editing to Connected f-Degree Graph is fixed-parameter tractable (FPT) by providing an algorithm solving the problem on an n-vertex graph in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$. Our FPT algorithm is based on a non-trivial combination of color-coding and fast computations of representative families over direct sum matroid of ℓ-elongation of co-graphic matroid associated with G and uniform matroid over $\overline{E(G)}$, the set of non-edges of G. We believe that this combination could be useful in designing parameterized algorithms for other edge editing problems.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Connected f-factor, FPT, Representative Family, Color Coding

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.36

1 Introduction

A subgraph F of a graph G is a factor of G, if F is a spanning subgraph of G. When a factor F is described in terms of its degrees, it is called a *degree-factor*. For example, one of the most fundamental notions in Graph Theory is 1-factor (or a perfect matching), the case when a factor F has all of its degrees equal to 1. Another example is r-factor, a regular spanning subgraph of degree r. More generally, for a function $f\colon V(G) \to \mathbb{N}$, subgraph F is a f-factor of G if for every $v \in V(G)$, $d_F(v)$, the degree of v in F is exactly equal to $f(v)$. The study of degree factors is one of the mainstays of combinatorics with a long history dating back to 1847 to the works of Kirkman [10], and Petersen [18]. We refer to surveys [1, 19], as well as the book of Lovász and Plummer [13] for an extensive overview of degree factors.

Another broad set of degree-factor problems is obtained by requesting the factor to be connected. The most famous examples are another old classical Graph Theory notion, the

* Supported by the European Research Council (ERC) via grants Rigorous Theory of Preprocessing, reference 267959 and PARAPPROX, reference 306992.

© Fedor V. Fomin, Petr Golovach, Fahad Panolan, and Saket Saurabh; licensed under Creative Commons License CC-BY

Hamiltonian cycle, which is a connected 2-factor, and the Eulerian subgraph, which is a connected even-degree factor. We refer to the survey of Kouider and Vestergaard [11] on connected factors, as well as to the book of Fleischner [6] for a thorough study of Eulerian graphs and related topics.

A natural algorithmic problem concerning (connected) f-factors is for a given graph G and a function f, to decide if G contains a (connected) f-factor. While deciding if a given graph contains an f-factor can be done in polynomial time for any function f [3], deciding the existence of even a connected 2-factor (Hamiltonian cycle) is NP-complete. In this work we study parameterized complexity of the following algorithmic generalization of the problem of finding a connected f-factor.

Edge Editing to Connected f-Degree Graph (EECG)
Input: An undirected graph G, a function $f : V(G) \to \{1, 2, \ldots, d\}$ and $k \in \mathbb{N}$
Question: Does there exist a connected graph F such that for every vertex v, $d_F(v) = f(v)$, and the cardinality of the symmetric difference $|E(G) \triangle E(F)| \leq k$?

Apart from studying a classical problem algorithmically, one of the main motivations for our interest in the generalization of the classical f-factor problem comes from the recent developments in parameterized algorithms for graph modification problems. These recent algorithmic advances were important not only due to the problems they settled but also for the kind of techniques they brought to the area. For example, the work on cut (or edge-deletion) problems of Kawarabayashi and Thorup [9] and Chitnis et al. [4] brought recursive understanding and randomized contraction techniques. The work on Feedback Arc Set in Tournaments [2] led to the chromatic coding. Study of chordal graph completions from [7] triggered the usage of potential maximal cliques in subexponential parameterized algorithms. The main result of our paper is the following theorem.

▶ **Theorem 1.** EECG *is solvable in time* $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ *deterministically.*

The proof of our theorem is based on (i) color-coding and (ii) fast computation of representative family over a linear matroid. While using graphic matroids to resolve different types of connectivity issues has become a popular theme in algorithms, our proof requires the usage of some non-standard matroids. In particular, we use fast representative family computations over the direct sum matroid of ℓ-elongation of the co-graphic matroid associated with G and a uniform matroid over $\overline{E(G)}$, the set of non-edges of G. We believe that this combination could be useful for designing parameterized algorithms for other edge editing problems. To the best of our knowledge this is the first uses of elongation of matroids in designing of parameterized algorithms.

One of the nice properties of using (graphic) matroids is that often they allow us to lift solutions from unweighted problems to problems with costs. It would be natural to suggest that the nice properties of matroids would help us with the "weighted" version of EECG as well. However, in spite of our attempts, we could not extend Theorem 1 to the weighted version of EECG. We proved that the weighted version of EECG when parameterized by $k + d$ is W[1]-hard and its proof is deferred to the full version of the paper due to paucity of space. Also, proofs of lemmata marked with a $\star$ are deferred to the full version of the paper.

Previous work. It was shown by Mathieson and Szeider [15] that the problem of deleting k vertices to transform an input graph into an r-regular graph, where $r \geq 3$ is W[1]-hard parameterized by k. For edge-modification problems to a graph with certain degrees, Mathieson and Szeider have shown in [15] that Edge Editing to f-Degree Graph, the

case when the requirement that the resulting graph F is connected is omitted, is solvable in polynomial time. As with the f-factor problem, the situation changes drastically when one adds the requirement that the resulting graph F is connected. A simple reduction from Hamiltonian cycle shows that in this case deciding if a graph can be edited into a connected 2-degree graph, i.e. a cycle, by changing at most k adjacencies, is NP-complete [8].

Golovach in [8] has shown that when parameterized by the maximum vertex degree d in the resulting graph plus the number of editing operations k, the problem Edge Editing to Connected f-Degree Graph is FPT. In the same paper, it was shown that the variant when the resulting graph F is regular, the problem is FPT parameterized by k. However, prior to our work the complexity status of Edge Editing to Connected f-Degree Graph parameterized by k remained open. Thus Theorem 1 resolves the problem in affirmative. However, we still do not know the kernelization status of this problem and leave it as an interesting open problem.

Our Approach. Each solution to our problem is of the form $D \cup A$ where $D \subseteq E(G)$ and $A \subseteq \overline{E(G)}$. The sets D and A are called a *deletion set* and an *addition set*, respectively, corresponding to the solution $D \cup A$. We start by characterizing our solution in terms of deletion set D, called *nice deletion set*. Nice deletion sets satisfy certain properties and have an accompanying map ψ. This map together with other properties of nice deletion sets allows us to recover the addition set A in polynomial time. Thus, our whole effort is in finding a nice deletion set D with map ψ. This viewpoint allows us to concentrate on finding a nice deletion set with an accompanying map ψ. However, this is not useful for designing the dynamic programming (DP) algorithm. To achieve this we view the solution $D \cup A$ as a system of "alternating walks and alternating even closed walks". Alternating (closed) walks are essentially normal (closed) walks with edges from $D \cup A$ such that they do not have consecutive edges from either D or A, and no edge from $D \cup A$ is repeated in the walk. We take this view point to construct the dynamic programming algorithm as this allows us to proceed between the states of the program by using one edge (either of an addition set or of a deletion set). The number of states can be upper bounded by $2^{\mathcal{O}(k)}$. However, the number of sets $D' \cup A'$ that could satisfy the prerequisite of being in a particular table entry could be as large as $n^{\mathcal{O}(k)}$ and thus this would not lead to an FPT algorithm. However, we follow this template for our algorithm and use some more structural properties to prune the family of partial solutions stored at a DP table entry.

Our pruning is based on the proof that $D \cup A$ is an independent set in some linear matroid. The first observation towards an FPT algorithm is that after we delete the edges in D from the input graph G, the number of connected components can at most be $k - |D| + 1$. This allows us to show that in fact we can think of D being an independent set in the matroid $M_G(\ell)$. That is, ℓ-elongation of the co-graphic matroid, M_G, associated with G, where $\ell = |E(G)| - |V(G)| + k - |D| + 1$ (we refer to preliminaries for the definition). Next we show that for addition set A, all we need to store is some form of disjointness and that can be captured using uniform matroid over the universe $\overline{E(G)}$. Let $U_{m',k-k'}$ be a uniform matroid with ground set $\overline{E(G)}$, where $m' = |\overline{E(G)}|$ and $k' = |D|$. From the definition of $U_{m',k-k'}$, any set A of size at most $k - k'$ is independent in $U_{m',k-k'}$. We have already explained that we view the deletion set D as an independent set in $M_G(\ell)$ where $\ell = |E(G)| - |V(G)| + k - k' + 1$. Thus, to see the solution set $D \cup A$ as an independent set in a single matroid, we consider direct sum of $M_G(\ell)$ and $U_{m',k-k'}$. That is, let $M = M_G(\ell) \oplus U_{m',k-k'}$. In M, a set I is an independent set if and only if $I \cap E(G)$ is an independent set in $M_G(\ell)$ and $I \cap \overline{E(G)}$ is an independent set in $U_{m',k-k'}$. This ensures that any solution $D \cup A$ is an independent

set in M. By viewing any solution of the problem as an independent set in the matroid M (which is linear), we can use fast computation of q-representative families to prune the table. However, we still need to take care of a technical requirement in the definition of a nice deletion set. Towards this, we show that for every deletion set D there exists a set of edges W_D, disjoint from D, of size at most $6k$ such that if these edges are not selected then we can satisfy that technical requirement. To achieve this we apply color coding technique and this can be derandomized using a standard application of universal sets.

2 Preliminaries

Throughout the paper we use ω to denote the exponent in the running time of matrix multiplication, the current best known bound for which $\omega < 2.373$ [20]. We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. For a set U, $\binom{U}{2} = \{(u, v) \mid u \neq v \wedge u, v \in U\}$. For any multiset A and an element x, by $A(x)$ we denote the number occurrences of x in A. For any $W \subseteq U \times \mathbb{N}$, where U is a set, $W(u) = |\{(u, i) \in W \mid i \in \mathbb{N}\}|$. We use "graph" to denote simple graphs without self-loops, directions, or labels. We use standard terminology from the book of Diestel [5] for those graph-related terms which we do not explicitly define. In general we use G to denote a graph. We use $V(G)$ and $E(G)$, respectively, to denote the vertex and edge sets of a graph G. For a graph G, we use $\overline{E(G)}$ to denote the simple non-edge set $\binom{V(G)}{2} \setminus E(G)$. For a vertex $v \in V(G)$, we use $E_G(v)$ to denote the set of edges incident on v.

Now we give definitions related to matroids. A pair $M = (E, \mathcal{I})$, where E is a ground set and $\mathcal{I}$ is a family of subsets (called independent sets) of E, is a *matroid* if it satisfies the following conditions: (i) $\emptyset \in \mathcal{I}$, (ii) if $A' \subseteq A$ and $A \in \mathcal{I}$ then $A' \in \mathcal{I}$; and (iii) if $A, B \in \mathcal{I}$ and $|A| < |B|$, then there is $e \in (B \setminus A)$ such that $A \cup \{e\} \in \mathcal{I}$. An inclusion wise maximal set of $\mathcal{I}$ is called a *basis* of the matroid. This size is called the *rank* of the matroid M, and is denoted by $\mathsf{rank}(M)$. The rank function of a matroid $M = (E, \mathcal{I})$ is a function $r_M : 2^E \to \mathbb{N}^+ \cup \{0\}$ which is defined as follows: $r_M(S)$ for any $S \subseteq E$ is the cardinality of the maximum sized independent set contained in S. For a broader overview on matroids, including linear representations of matroids, we refer to [17].

Direct Sum of Matroids. Let $M_1 = (E_1, \mathcal{I}_1)$, $\ldots$, $M_t = (E_t, \mathcal{I}_t)$ be t matroids with $E_i \cap E_j = \emptyset$ for all $1 \leq i \neq j \leq t$. The direct sum $M_1 \oplus \cdots \oplus M_t$ is a matroid $M = (E, \mathcal{I})$ with $E := \bigcup_{i=1}^{t} E_i$ and $X \subseteq E$ is independent if and only if $X \cap E_i \in \mathcal{I}_i$ for all $i \in [t]$.

▶ **Proposition 2** ([14, Proposition 3.4]). *Given representations of matroids $M_1, \ldots, M_t$ over the same field $\mathbb{F}$, a representation of their direct sum can be found in polynomial time.*

Uniform Matroids. A pair $M = (E, \mathcal{I})$ over an n-element ground set E, is called a uniform matroid if the family of independent sets is given by $\mathcal{I} = \{A \subseteq E \mid |A| \leq k\}$, where k is some constant. This matroid is also denoted as $U_{n,k}$. Every uniform matroid is linear and can be represented over a finite field of size $\geq n + 1$ by a $k \times n$ matrix A_M where $A_M[i, j] = e_j^{i-1}$, and $e_1, \ldots, e_n$ are distinct non-zero elements from the field.

Co-graphic Matroids. Given a graph G with r connected components, the co-graphic matroid associated with G, denoted by M_G, is defined as $(U, \mathcal{I})$, where $U = E(G)$ and $\mathcal{I} = \{S \subseteq E(G) : G - S \text{ has exactly } r \text{ connected components}\}$. Co-graphic matroids are representable over any field of size at least 2 [17].

Elongation of a Matroid. The *ℓ-elongation* of a matroid M, where $\ell \geq \mathsf{rank}(M)$ is defined as a matriod $M' = (E, \mathcal{I}')$ such that $S \subseteq E$ is a basis in M' if and only if, $r_M(S) = r_M(E)$ and $|S| = \ell$. We use $M(\ell)$ to denote the ℓ-elongation of a matroid M.

▶ **Theorem 3** ([12]). *Given a representation of a matroid M and $\ell \geq \mathsf{rank}(M)$, there is a deterministic polynomial time algorithm to compute a representation of ℓ-elongation of M.*

▶ **Definition 4** (q-Representative Family [14]). Given a matroid $M = (E, \mathcal{I})$ and a family $\mathcal{S}$ of subsets of E, we say that a subfamily $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is *q-representative* for $\mathcal{S}$ if the following holds: for every set $Y \subseteq E$ of size at most q, if there is a set $X \in \mathcal{S}$ disjoint from Y with $X \cup Y \in \mathcal{I}$, then there is a set $\widehat{X} \in \widehat{\mathcal{S}}$ disjoint from Y with $\widehat{X} \cup Y \in \mathcal{I}$. If $\widehat{\mathcal{S}} \subseteq \mathcal{S}$ is q-representative for $\mathcal{S}$, then we write $\widehat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$.

▶ **Theorem 5** ([12]). *Let $M = (E, \mathcal{I})$ be a linear matroid of rank n and let $\mathcal{S} = \{S_1, \ldots, S_t\}$ be a family of independent sets, each of size b. Let A be an $n \times |E|$ matrix representing M over a field $\mathbb{F}$, where $\mathbb{F} = \mathbb{F}_{p^\ell}$ or $\mathbb{F}$ is $\mathbb{Q}$. Then there is deterministic algorithm which computes a representative set $\widehat{\mathcal{S}} \subseteq^q_{rep} \mathcal{S}$ of size at most $nb\binom{b+q}{b}$, using $\mathcal{O}\left(\binom{b+q}{b} tb^3n^2 + t\binom{b+q}{b}^{\omega-1}(bn)^{\omega-1}\right) + (n+|E|)^{\mathcal{O}(1)}$ operations over the field $\mathbb{F}$.*

3 An FPT Algorithm for EECG

Let (G, f, k) be an instance of EECG. Our algorithm finds a solution of size *exactly* k (if one such solution exists) and outputs No otherwise. Any solution to our problem is of the form $D \cup A \in \binom{V(G)}{2}$ where $D \subseteq E(G)$ and $A \subseteq \overline{E(G)}$. The sets D and A are called a deletion set and an addition set, respectively, corresponding to the solution $D \cup A$.

Characterizing the solution. The starting point of our algorithm is a characterization of solution in terms of a deletion set D satisfying certain properties (such deletion sets are called *nice deletion sets*). This, allows us to focus on finding nice deletion sets. To describe the main steps and ideas involved in our algorithm we first give a definition of a nice deletion set. Towards this we need definitions of following two sets .

$$\begin{aligned} \mathsf{def}(G,f) &= \{v \mid v \in V(G), f(v) > d_G(v)\} - \text{ A set of deficient vertices.} \\ \mathsf{S}(G,f) &= \{(v,i) \mid v \in V(G), f(v) > d_G(v), i \in \{1, \ldots, f(v) - d_G(v)\}\} \end{aligned}$$

The second set specifies for every vertex $v \in \mathsf{def}(G, f)$ how many edges in addition set must be adjacent to v. Let $W \subseteq V(G) \times \mathbb{N}$. For convenience we denote a pair $(v, i) \in V(G) \times \mathbb{N}$ (or in W) by $v(i)$. Let $\psi : W \to W$ be a bijection. Given ψ, we define a multiset E_ψ as follows. For each $u(i) \in W$ we add (u, v) to E_ψ if $\psi(u(i)) = v(j)$ for some $j \geq i$. We say that ψ is a *proper deficiency map* if ψ is an involution, for any $u \in V(G)$ $(u, u) \notin E_\psi$, E_ψ is a set and not a multiset, and $E_\psi \cap E(G) = \emptyset$. In general we will have proper deficiency map over the domain $\mathsf{S}(G, f)$ or some set related to this. Finally, a set $D \subseteq E(G)$ is called a *nice deletion set* if

- **(i)** For all $v \in V(G)$, $d_{G-D}(v) \leq f(v)$;
- **(ii)** $|\mathsf{S}(G - D, f)| = 2(k - |D|)$;
- **(iii)** The graph $G - D$ has at most $k - |D| + 1$ connected components;
- **(iv)** Each connected component in $G - D$ contains a vertex v deficient in $G - D$, i.e, such that $d_{G-D}(v) < f(v)$.
- **(v)** There exists a proper deficiency map $\psi : \mathsf{S}(G - D, f) \to \mathsf{S}(G - D, f)$.

Our main structural lemma is the following.

▶ **Lemma 6.** *Let (G, f, k) be an instance of* EECG *and let $D \subseteq E(G)$. Then there exists $A \subseteq \overline{E(G)}$, $|A| = k - |D|$ such that $A \cup D$ is a solution to* EECG *if and only if D is a nice deletion set. Moreover, given a nice deletion set $D \subseteq E(G)$ we can find $A \subseteq \overline{E(G)}$ such that $|A| = k - |D|$ and $D \cup A$ is a solution to* EECG *in polynomial time.*

Proof. ($\Rightarrow$) Let $A \subseteq \overline{E(G)}, |A| = k - |D|$ such that $A \cup D$ is a solution to EECG. We need to show that $D \subseteq E(G)$ is a nice deletion set. Since $A \cup D$ is a solution to EECG, we have that $d_{G-D}(v) \leq f(v)$ for all $v \in V(G)$, satisfying condition (i). Furthermore, $A \cup D$ being a solution also implies that $\sum_{\{v\ :\ f(v) > d_{G-D}(v)\}} f(v) - d_{G-D}(v) = 2|A| = 2(k - |D|)$. Hence $|\mathsf{S}(G - D, f)| = 2(k - |D|)$, satisfying condition (ii). Since $G - D + A$ is a connected graph, $G - D$ can have at most $|A| + 1 = k - |D| + 1$ connected components, satisfying condition (iii) in the definition. The graph $G - D + A$ is connected and thus each connected component F in $G - D$ contains a vertex $v \in V(F)$ such that $(v, u) \in A$ for some $u \in V(G)$. Since $D \cup A$ is a solution to EECG, $d_{G-D+A}(v) = f(v)$ and hence $d_{G-D}(v) < f(v)$ (because $(v, u) \in A$), satisfying condition (iv). Finally, we show that D satisfies the last property. Let $A = \{e_1, e_2, \ldots, e_r\} \subseteq \overline{E(G)}$ where $r = k - |D|$. Since $D \cup A$ is a solution to EECG, we have that for any vertex v, there are exactly $f(v) - d_{G-D}(v)$ edges in A which are incident to v. Now we define a bijection $\psi : \mathsf{S}(G - D, f) \to \mathsf{S}(G - D, f)$ as follows: $\psi(u(i)) = v(j)$ if $(u, v) = e_\ell$ such that there are exactly $i - 1$ edges from $\{e_1, \ldots, e_{\ell-1}\}$ are incident on u and there are exactly $j - 1$ edges from $\{e_1, \ldots, e_{\ell-1}\}$ are incident on v. That is, we traverse the edges $e_1, \ldots, e_r$ from left to right and find the i^{th} edge incident to u, say $e_\ell = (u, v)$, and then we assign it $v(j)$, if there are exactly $j - 1$ edges incident to v present among $\{e_1, \ldots, e_{\ell-1}\}$.

▶ **Claim 7.** *$\psi : \mathsf{S}(G - D, f) \to \mathsf{S}(G - D, f)$ is a proper deficiency map.*

Proof. By the definition of ψ, if $\psi(u(i)) = v(j)$ then $\psi(v(j)) = u(i)$, and so ψ is an involution. Since $G - D + A$ is a simple graph, for any $u \in V(G)$, $(u, u) \notin E_\psi$. Now we need to show that E_ψ is not a multiset. Suppose not, then there exists $u, v \in V(G)$ such that $\psi(u(i_1)) = v(j_1)$ and $\psi(u(i_2)) = v(j_2)$ for some $i_1 \neq i_2$ and $j_1 \neq j_2$. This implies that there exist $e_i, e_j \in A$, $i \neq j$ such that $e_i = e_j = (u, v)$. This contradicts the fact that $G - D + A$ is a simple graph. Since A is disjoint from $E(G)$, $E_\psi \cap E(G) = \emptyset$. ◀

($\Leftarrow$) Let $D \subseteq E(G)$ be a nice deletion set. We need to show that we can find $A \subseteq \overline{E(G)}$ such that $|A| = k - |D|$ and $G - D + A$ is a solution to EECG. Properties (i) and (ii) imply that $d_{G-D}(v) \leq f(v)$ for all $v \in V(G)$ and $|\mathsf{S}(G - D, f)| = 2(k - |D|)$. Due to the property (v) in the definition of nice deletion set, we know that there exists a proper deficiency map $\psi : \mathsf{S}(G - D, f) \to \mathsf{S}(G - D, f)$. Define $A_1 = E_\psi$. By the definition of E_ψ, $|A_1| = |E_\psi| = |\mathsf{S}(G - D, f)|/2 = k - |D|$. Also note that $A_1 \cap E(G) = \emptyset$ because ψ is a proper deficiency map. Now consider the graph $G_1 = G - D + A_1$. Note that G_1 is simple graph because ψ is a proper deficiency map. Also by the definition of E_ψ, $d_{G-D+A_1}(v) = f(v)$ for all $v \in V(G)$. So G_1 satisfies the degree constraints. If G_1 is connected then $D \cup A_1$ is a solution to EECG. Thus, we assume that G_1 is not connected. In what follows we give an iterative procedure that finds the desired A. Suppose we are in i^{th} iteration and we have a set A_i such that $G - D + A_i$ satisfies all degree constraints but $G - D + A_i$ is not connected. Then in the next iteration we find another addition set of size $k - |D|$, say A_{i+1}, such that $G - D + A_{i+1}$ satisfies all degree constraints and $G - D + A_{i+1}$ has strictly less number of connected components than in $G - D + A_i$. The procedure is started with $A_1 = E_\psi$. Let $i \geq 1$ and we have A_i such that $G - D + A_i$ satisfies all degree constraints but $G - D + A_i$ is not connected. Since $|A_i| = k - |D|$ and $G - D$ has at most $k - |D| + 1$ connected components, $G_i = G - D + A_i$ has a component F such that there is an edge

$(u_1, v_1) \in A_i$ with the property that $u_1, v_1 \in V(F)$ and (u_1, v_1) is not a bridge in F. Let F' be another connected component in G_i. Since each connected component in $G - D$ contains a vertex $w \in V(G)$ such that $d_{G-D}(w) < f(w)$, there exists an edge $(u_2, v_2) \in A_i$ such that $u_2, v_2 \in V(F')$. Now let $A_{i+1} = (A_i \setminus \{(u_1, v_1), (u_2, v_2)\}) \cup \{(u_1, u_2), (v_1, v_2)\}$. Observe that $G_{i+1} = G - D + A_{i+1}$ is a simple graph with strictly less connected component than in G_i and $d_{G_{i+1}}(v) = f(v)$ for all $v \in V(G)$. Observe that when the procedure stops we would have found the desired A.

Given a nice deletion set D, we can find the desired A using the iterative procedure described in the reverse direction of the proof. Clearly, this procedure can be implemented in polynomial time. This completes the proof of the lemma. ◀

Thus, our problem reduces to finding a nice deletion set $D \subseteq E(G)$ and an accompanying proper deficiency map ψ on $\mathsf{S}(G - D, f)$, if it exists, using dynamic programming (DP).

Towards the states of dynamic programming algorithm. So how to find a nice deletion set D? Throughout this section we will work with a *hypothetical* deletion set D. We partition the vertices of G into Green and Red in the following way. We color $v \in V(G)$ green if $d_G(v) > f(v)$, and red otherwise. Let $E_r = E(G[\mathsf{Red}])$ and $E_g = E(G) \setminus E_r$. We need a quick sanity check. That is, if $\sum_{\{v:d_G(v)\neq f(v)\}} |d_G(v) - f(v)| > 2k$ then we output No, because in this case any solution to EECG requires more than k edge edits (addition/deletion operations). Now we guess the size $k' \leq k$ of D such that $2k' \geq \sum_{v:d_G(v)>f(v)} d_G(v) - f(v)$. Since D is our hypothetical deletion set, we have that for any $v \in \mathsf{Green}$, the number of edges in D which are incident with v is at least $d_G(v) - f(v)$. Now we guess the number k_1 of edges in D which are incident with only green vertices and the number k_2 of edges in D which are incident with at least one vertex in Red. Note that $k_1 + k_2 = k'$. Also note that the number of ways we can guess (k', k_1, k_2) is at most k^2. Now for every $v \in \mathsf{Green}$, we guess the number of edges in D which are incident with v. In particular, we guess a function $\Phi : \mathsf{Green} \to \mathbb{N}$ such that for all $v \in \mathsf{Green}$ we have that $\Phi(v) \geq d_G(v) - f(v)$ and $\sum_{v\in\mathsf{Green}} \Phi(v) \leq 2k_1 + k_2$. The number of possible such functions Φ is upper bounded by $\mathcal{O}(4^k k)$. From now onwards we will assume that we are given function Φ. In other words we have guessed the function Φ corresponding to the hypothetical solution D.

We start with an intuitive explanation of the structure of the solution that is helpful in designing partial solution for the DP algorithm. Given $D \cup A$, we first define a notion of an alternating walk. An *alternating walk* is a sequence of vertices $u_1, u_2, \ldots, u_\ell$ such that consecutive pairs of vertices $((u_i, u_{i+1}), (u_{i+1}, u_{i+2}))$ either belong to $D \times A$ or $A \times D$ and $\{(u_i, u_{i+1}) \mid 1 \leq i < \ell\}$ is a set (not a multiset). That is, an edge from D is followed by an edge from A or vice-versa. In an *alternating even length closed walk*, $u_1 = u_\ell$ and ℓ is even. One might wonder about the definition of alternating odd length closed walk. For our purposes we will think of them as alternating walks that start and end at the same vertex. Essentially, these will be alternating walks that start and end with the same vertex and the first and the last edge either both belong to D or both belong to A. From now onwards whenever we say an alternating closed walk, we mean an alternating even length closed walk. For every intermediate, i.e. not the endpoint, vertex in an alternating walk or in an alternating closed walk, one of the edges incident with it belongs to D and while the second edge belongs to A. Thus the degree of any vertex is not disturbed by an alternating walk where this vertex is intermediate. We define alternating walks for the following reason. Let $D \cup A$ be a solution of EECG that satisfies Φ. We can think of edges in $D \cup A$ forming *a family $\mathcal{P}$ of edge disjoint alternating walks and alternating closed walks* with the following properties.

- For every vertex $v \in V(G)$ and a set $Z \in \{D, A\}$, we define $\mathsf{apdeg}(\mathcal{P}, Z, v)$ as the number of edges from Z that are incident with v and appear as (i) the first edge in alternating walks from $\mathcal{P}$ that start with v; and (ii) the last edge in alternating walks from $\mathcal{P}$ that end in v. Note that, if there is an alternating walk that both starts and ends in v and the start edge as well as the last edge belong to Z then this walk contributes two to $\mathsf{apdeg}(\mathcal{P}, Z, v)$. For every vertex $v \in \mathsf{Green}$, $\mathsf{apdeg}(\mathcal{P}, D, v) = d_G(v) - f(v)$ and $\mathsf{apdeg}(\mathcal{P}, A, v) = 0$. Furthermore, for every vertex $v \in \mathsf{Red}$, $\mathsf{apdeg}(\mathcal{P}, D, v) = 0$ and $\mathsf{apdeg}(\mathcal{P}, A, v) = f(v) - d_G(v)$. When the number of edges in D which are incident with $v \in \mathsf{Green}$ is greater than $d_G(v) - f(v)$, then the number of times v appear as intermediate vertices in alternating (closed) walks is exactly equal to the number of *excess* edges and these excess edges will not contribute to $\mathsf{apdeg}(\mathcal{P}, D, v)$.
- Every vertex $v \in \mathsf{Green}$, appears as an intermediate vertex in an alternating (closed) walk of $\mathcal{P}$ exactly $\Phi(v) - (d_G(v) - f(v))$ times.

For any solution $\mathcal{P} = \{P_1, P_2, \ldots, P_\alpha\}$, without loss of generality we assume that there is η such that $P_1, \ldots, P_\eta$ are alternating walks and $P_{\eta+1}, \ldots, P_\alpha$ are alternating closed walks. This walk system view allows us to make a dynamic programming algorithm where we can move from one state to another using one edge addition or deletion. In particular, the algorithm works by constructing all alternating walks $P_1, \ldots, P_\eta$ first and then construct alternating closed walks $P_{\eta+1}, \ldots, P_\alpha$. Given a partially constructed walk system we try to append an edge to the current walk we are constructing by adding an edge from $\binom{V(G)}{2}$ to it; or declaring that we are finished with the current walk and move to construct a new walk. During this process we also keep a partial proper deficiency map ψ' such that $E_{\psi'}$ are addition edges in the current partial solution. Thus, a state in the DP algorithm is given by our current guesses and a subset of domain of partial proper deficiency map. For a formal description of our DP table we need the following sets: (a) a multiset set T_m containing $d_G(v) - f(v)$ many copies of v for all $v \in \mathsf{Green}$; (b) a set $T_g = \{v(i) \mid v \in \mathsf{Green}, \Phi(v) > d_G(v) - f(v), i \in [\Phi(v) - (d_G(v) - f(v))]\}$; (c) $T_r = \{v(i) \mid v \in \mathsf{Red}, f(v) > d_G(v), i \in [f(v) - d_G(v)]\}$. We also need to keep two sets X - a multiset and a set Y to take care of local deficiencies that occur while constructing our walk system. Let X be a multiset of size 2. Then for all $u \in X$ and $B \subseteq E(G)$ such that $f(u) > d_{G-B}(u) + X(u) - 1$, define $X_{B,f} = \{u(i) \mid u \in X, f(u) - d_{G-B}(u) - X(u) + 1 \leq i \leq f(u) - d_{G-B}(u), i \in \mathbb{N}\}$.

Now we formally define a notion of partial solutions. Given an instance (G, f, k) we define T_m, T_g and T_r as described earlier. Also, recall that we have k_1, k_2 and Φ. For any $T'_m \subseteq T_m, T'_g \subseteq T_g, T'_r \subseteq T_r, k'_1 \leq k_1, k'_2 \leq k_2, i \leq k$, a multiset X containing elements from $V(G)$ and $Y \subseteq V(G)$ such that $|X| \leq 2, |Y| \leq 1, |X \cup Y| \leq 2$ and $X \cap Y = \emptyset$, we define a family $\mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ of subsets of $\binom{V(G)}{2}$ as follows. For any $B \subseteq E(G)$ and $A \subseteq \overline{E(G)}$, $B \cup A \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ if the following conditions are met:

(a) $|E(G[\mathsf{Green}]) \cap B| = k'_1, |B \setminus E(G[\mathsf{Green}])| = k'_2$ and $|B \cup A| = i$.
(b) For any $v \in \mathsf{Green}$, the number of edges in B which are incident with v is exactly equal to $T'_m(v) + T'_g(v) + X(v)$. That is, for all $v \in \mathsf{Green}$, $|B \cap E_G(v)| = T'_m(v) + T'_g(v) + X(v)$.
(c) $|X_{B,f}| = |X|$ and $X_{B,f} \subseteq S(G - B, f) \setminus T_r$.
(d) $G - B$ has at most $k - k' + 1$ connected components.
(e) There is a proper deficiency map $\psi' : (S(G - B, f) \cup T'_g \cup Y) \setminus (T'_r \cup X_{B,f}) \to (S(G - B, f) \cup T'_g \cup Y) \setminus (T'_r \cup X_{B,f})$ such that $A = E_\psi$. Furthermore, for $w \in Y$, if $\psi'(w) = u(j)$ for some j then $T_m(u) = T'_m(u)$.

For $T'_m \subseteq T_m, T'_g \subseteq T_g, T'_r \subseteq T_r, k'_1 \leq k_1, k'_2 \leq k_2, i \leq k$, a multiset X containing elements from $V(G)$ and $Y \subseteq V(G)$, we say that the tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ is a *valid* tuple if the following happens.

1. $|X| \leq 2, |Y| \leq 1, |X \cup Y| \leq 2$ and $X \cap Y = \emptyset$
2. For $w \in Y$, $w(j) \notin T'_r$ for all j.
3. If $u(j) \in T'_g$ then for all $0 < j' < j$, $u(j') \in T'_g$.
4. For any $v \in X$, $T_m(v) = T'_m(v)$.

For the correctness of the algorithm, it is enough to focus on partial solutions defined over valid tuples. Now we prove that in fact $\mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ is "a correct notion of partial solution".

▶ **Lemma 8** (⋆). *Let (G, f, k) be an* Yes *instance of* EECG *with a solution $D \cup A$ such that $D \subseteq E(G), A \subseteq \overline{E(G)}$, $|D \cap E(G[\mathsf{Green}])| = k_1, |D \setminus E(G[\mathsf{Green}])| = k_2$, $k_1 + k_2 = k'$ and $|D \cap E_G(v)| = \Phi(v)$ for all $v \in \mathsf{Green}$. Let ψ be a proper deficiency map over $S(G - D, f)$ such that $E_\psi = A$. Then for each $i \leq k$, there exists $D' \cup A' \subseteq D \cup A$ and a valid tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ such that $D' \cup A' \in \mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ and there is a proper deficiency map ψ' over $R = (S(G - D', f) \cup T'_g \cup Y) \setminus (T'_r \cup X_{D',f})$ with $E_{\psi'} = A'$.*

Our DP algorithm keeps a table entry $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ for each valid tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$. The idea is to store a subset of $\mathcal{Q}(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ in the DP table entry $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ which is sufficient to maintain the correctness of the algorithm. The algorithm computes $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ for all valid tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ and if there exists $D \cup A \in \mathcal{D}[T_m, T_g, \emptyset, k_1, k_2, k, \emptyset, \emptyset]$ such that $D \cup A$ is a solution to EECG, then outputs Yes, otherwise outputs No. In fact one can show the following simple lemma which bounds the number of DP table entries:

▶ **Lemma 9** (⋆). *The number of valid tuples $(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ is at most $2^{\mathcal{O}(k)} n^3$.*

However, the size of family stored at (one table entry) $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ can potentially be $n^{\mathcal{O}(i)}$, thus this algorithm takes time $n^{\mathcal{O}(k)}$. Next we observe that we can prune each table entry to $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ and thus this leads to an FPT algorithm.

Pruning the DP table entry and an FPT algorithm. We need to prune the DP table in a way that we do not change the answer to the given instance (G, f, k). Towards this we show that if some subset we have stored in a DP table entry could lead to a nice deletion set then we do have at least one such set after the pruning operation. Our guessing of k_1, k_2 and Φ allows us to satisfy the properties (i) and (ii) of a nice deletion set. Property (iii) of nice deletion sets implies that D is an independent set in the matroid $M_G(\ell)$, the ℓ-elongation of the co-graphic matroid M_G associated with G, where $\ell = |E(G)| - |V(G)| + k - |D| + 1$. Thus by only considering those D which are independent sets in $M_G(\ell)$ we ensure that property (iii) of nice deletion sets is satisfied. Now consider the property (v) of the nice deletion set, i.e, there exists a proper deficiency map $\psi : S(G - D, f) \to S(G - D, f)$. Our objective is to get a set $D \cup A$ such that there is a proper deficiency map ψ over $S(G - D, f)$ with the property that $E_\psi = A$, along with other properties as well. Let $D_1 \cup A_1, D_2 \cup A_2$ be two partial solutions belonging to the same equivalence class where $D_1, D_2 \subseteq E(G)$ and $A_1, A_2 \subseteq \overline{E(G)}$. Suppose $D' \subseteq E(G)$, $A' \subseteq \overline{E(G)}$, $(D_1 \cup D') \bigcup (A_1 \cup A')$ is a solution and $A_2 \cap A' = \emptyset$. Since $D_1 \cup A_1$, $D_2 \cup A_2$ belongs to same equivalence class and $A_2 \cap A'$ is disjoint, there is a proper deficiency map ψ' over $S(G - (D_2 \cup D'), f)$ such that $E_{\psi'} = A_2 \cup A'$. To take care of the disjointness property between the current addition set and the future addition set while doing the DP, we view the addition set A of the solution as an independent set in a uniform matroid over the universe $\overline{E(G)}$. Let $U_{m',k-k'}$ be a uniform matroid with ground set $\overline{E(G)}$, where $m' = |\overline{E(G)}|$. From the definition of $U_{m',k-k'}$, every set A of size at most $k - k'$ is independent in $U_{m',k-k'}$. We have already explained that we view the

deletion set D as an independent set in $M_G(\ell)$ where $\ell = |E(G)| - |V(G)| + k - k' + 1$. To view the solution set $D \cup A$ as an independent set in a matroid, we consider the direct sum $M = M_G(\ell) \oplus U_{m',k-k'}$ of two matroids. In M, a set I is an independent set if and only if $I \cap E(G)$ is an independent set in $M_G(\ell)$ and $I \cap \overline{E(G)}$ is an independent set in $U_{m',k-k'}$. This ensures that any solution $D \cup A$ is an independent set in M. By viewing any solution of the problem as an independent set in a matroid M (which is linear by Proposition 2), we can use the q-representative families, Theorem 5, to prune the table size to $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$.

However, we still need to ensure that property (iv) of nice deletion set is satisfied. In what follows we explain how we achieve this. The following lemma helps us to satisfy property (iv) partially.

▶ **Lemma 10** ($\star$). *Let F be a connected component in the graph $G[\mathsf{Red}]$ and $D' \subseteq E(G)$. If at least one edge in D' is incident to a vertex in $V(F)$, then for any connected component C in $G - D'$ such that $V(C) \cap V(F) \neq \emptyset$ there is a vertex $v \in V(C) \cap \mathsf{def}(G - D', f)$.*

Now we explain how Lemma 10 is useful in satisfying property (iv) partially. Let $\mathcal{C}$ be the set of connected components in G such that for each vertex v in the component, $d_G(v) = f(v)$.

$$\mathcal{C} = \{C \mid C \text{ is a connected component in } G \wedge \forall v \in V(C), d_G(v) = f(v)\}.$$

Let D_1 and D_2 be deletion sets corresponding to two partial solutions such that for all $C \in \mathcal{C}$, $D_1 \cap E(C) \neq \emptyset$ if and only if $D_2 \cap E(C) \neq \emptyset$. Suppose there is a set $D' \subseteq E(G)$ such that $D_1 \cup D'$ is a nice deletion set. Then we can show that any connected component F in $G - (D_2 \cup D')$ containing only red vertices will have a deficient vertex. Essentially due to Lemma 10, if we partition our partial solutions based on how these partial solutions hit the edges from $\mathcal{C}$ and keep at least one from each equivalence class property (iv) of nice deletion set will be satisfied partially. But this only allows us to take care of connected components containing only red vertices. Now we explain how we can ensure property (iv) for the connected components containing vertices from Green as well.

To achieve this we will prove that corresponding to every deletion set D of a solution, there is a "witness" of $\mathcal{O}(k)$ sized subset of edges whose disjointness from D will ensure property (iv) of nice deletion sets. That is, these witnesses depend on solutions; the witness for solution D could be different from the witness for solution D^*. Even then, these witnesses allow us to satisfy property (iv). In order to avoid this witness being picked in a deletion set D, that is to keep this witness non deletable, we use color coding in our algorithm on top of representative family based pruning of table entries. Towards that we define a weight function w on $E(G)$ as follows.

$$\mathrm{w}((u,v)) = \begin{cases} 0 & \text{if } u, v \in \mathsf{Red} \\ 1 & \text{otherwise} \end{cases}$$

For any subset $S \subseteq E(G)$, $\mathrm{w}(S) = \sum_{e \in S} \mathrm{w}(e)$. The next lemma is crucial for our approach as this not only defines the witness but also gives an upper bound on its size.

▶ **Lemma 11.** *Let $\mathsf{Green} = \{v_1, v_2, \ldots, v_\eta\}, \eta \leq 2k$. Let $D \subseteq E(G)$ such that for any connected component F in $G - D$, $V(F) \cap \mathsf{def}(G - D, f) \neq \emptyset$. Then there exist paths $P_1, \ldots, P_\eta$ such that for all i, P_i is a path in $G - D$ from v_i to a vertex in $\mathsf{def}(G - D, f)$, and $\mathrm{w}(\bigcup_i E(P_i)) \leq 6k$ where $\bigcup_i E(P_i)$ is the set of edges in the paths $P_1, \ldots P_\eta$.*

Proof. We construct $P_1, \ldots, P_\eta$ with the required property. Pick an arbitrary vertex $u_1 \in \mathsf{def}(G - D, f)$ such that v_1 and u_1 are in the same connected component in $G - D$. Let P_1 be a smallest weight path according to weight function w, from v_1 to u_1 in $G - D$. Now

we explain how to construct P_i, given that we have already constructed paths $P_1, \ldots, P_{i-1}$. Pick an arbitrary vertex $u_i \in \mathsf{def}(G - D, f)$ such that v_i and u_i are in the same connected component in $G - D$. Let P be a smallest weight path from v_i to u_i in $G - D$. If P is vertex disjoint from $P_1, \ldots, P_{i-1}$, then we set $P_i = P$. Otherwise, let x be the first vertex in P such that $x \in V(P_1) \cup \ldots \cup V(P_{i-1})$. Let $x \in P_j$ where $j < i$. Let $P = P'P''$ such that P' ends in x and P'' starts at x. Let $P_j = P'_j P''_j$ such that P'_j ends in x and P''_j starts at x. Now we set $P_i = P'P''_j$. Note that P_i is a path in $G - D$ from v_i to a vertex in $\mathsf{def}(G - D, f)$.

Now we claim that $\mathrm{w}(\bigcup_{i=1}^{\eta} E(P_i)) \leq 6k$. Towards the proof we need to count that $|(\bigcup_{i=1}^{\eta} E(P_i)) \cap \mathrm{w}^{-1}(1)| \leq 6k$. We assign each vertex v in $\bigcup_{i=1}^{\eta} V(P_i)$ to the smallest indexed path P_j such that $v \in V(P_j)$. That is, v is assigned to P_j, if $v \in V(P_j)$ and $v \notin (\bigcup_{i=1}^{j-1} V(P_i)$. Note that each vertex in $\bigcup_{i=1}^{\eta} V(P_i)$ is assigned to a unique path. Consider the edge set $A^* \subseteq (\bigcup_{i=1}^{\eta} E(P_i)) \cap \mathrm{w}^{-1}(1)$ as follows. An edge $e = (u, v)$ belongs to A^* if $\mathrm{w}(e) = 1, e \in E(P_j)$, and vertices u and v are assigned to path P_j for some j. Observe that each edge $e \in A^*$ has at least one end point in Green. Since each vertex is assigned to exactly one path, each vertex in a path has degree at most 2 and $|\mathsf{Green}| \leq 2k$, we have that $|A^*| \leq 4k$.

Now we show that there exists sets $\emptyset = B_1 \subseteq B_2 \subseteq \ldots B_\eta$ such that $(\bigcup_{i=1}^{j} E(P_i)) \cap \mathrm{w}^{-1}(1) \subseteq A^* \cup B_j$ and $|B_j| \leq j$. We prove the statement using induction on j. For $j = 1$, we know that $(\bigcup_{i=1}^{j} E(P_i)) \cap \mathrm{w}^{-1}(1) \subseteq A^*$. Thus the statement is true. Now suppose the statement is true for $j - 1$. Consider any path P_j. If the vertices in P_j are disjoint from $\bigcup_{i}^{j-1} V(P_i)$, then all the weight one edges in $E(P_j)$ are counted in A^*. So we can set $B_j = B_{j-1}$ and the statement is true. Otherwise by the construction of P_j, we have that $P_j = P'_j P''_j$ and there exists $r < j$ such that $P_r = P'_r P''_j$. Let (u_1, u_2) be the last edge in P'_j. Note that all the weight one edges in $E(P''_j)$ are counted in $A^* \cup B_{j-1}$ and all the weight one edges in $E(P'_j) \setminus \{(u_1, u_2)\}$ are counted in A^*. In this case we set $B_j = B_{j-1}$ if $\mathrm{w}((u_1, u_2) = 0)$ and $B_j = B_{j-1} \cup \{(u_1, u_2)\}$ otherwise. This implies that $|(\bigcup_{i=1}^{\eta} E(P_i)) \cap \mathrm{w}^{-1}(1)| \leq 6k$. This concludes the proof. ◀

Recall that $E_r = E(G[\mathsf{Red}])$ and $E_g = E(G) \setminus E_r$. Note that in Lemma 11, the weight of each edge in E_g is 1 and the weight of each edge in E_r is 0. By Lemma 11, we have that if D is a nice deletion set, then there exists $E' \subseteq E_g$ of cardinality at most $6k$ such that E' witnesses that each connected component of $G - D$ containing at least one vertex from Green, will also contain a vertex from $\mathsf{def}(G - D, f)$. We call such an edge set E' as ***certificate*** of D. Now we explain how Lemma 11 helps us to satisfy property (iv) of nice deletion sets for components containing at least one vertex from Green. Let $\mathsf{Green} = \{v_1, \ldots, v_\eta\}$ and $D_1 \cup D'$ be a deletion set corresponding to a solution. By Lemma 11 we know that there are paths $P_1, \ldots, P_\eta$ such that the total number of edges from E_g among these paths is bounded by $6k$, and each path P_i is from v_i to a vertex in $\mathsf{def}(G - (D_1 \cup D'), f)$. Suppose we color the edges in E_g with black and orange such that the coloring guarantees that all the edges in $E_g \cap (\bigcup_{i=1}^{\eta} E(P_i))$ are colored black and all the edges in $E_g \cap (D_1 \cup D')$ are colored orange. Assume that we are going to find a nice deletion set which does not contains black color edges. Let D_2 be a deletion set corresponding to a partial solution. Also for a vertex $v_i \in \mathsf{Green}$, there is path from v_i to a vertex in $\mathsf{def}(G - D_1, f)$ in the graph $G - D_1$ which does not contain any orange colored edge if and only if there is path from v_i to a vertex in $\mathsf{def}(G - D_2, f)$ in the graph $G - D_2$ which does not contain any orange colored edge. We can show that any connected component in $G - (D_2 \cup D')$ containing a vertex from Green will contain a vertex from $\mathsf{def}(G - (D_2 \cup D'), f)$. Essentially by Lemma 11 we get the following. Suppose we take all partial solutions corresponding to a DP table entry (or a subset of it) and now we partition these partial solutions based on which all green vertices have found

their deficient vertex currently (there are $2^{|\mathsf{Green}|}$ such partitions), then it is enough to keep a partial solution from each class. Furthermore, suppose $\mathcal{A}$ corresponds to partial solutions with respect to one particular subset of Green and we have kept a set D_1 in $\mathcal{A}$ and deleted rest of the partial solutions from $\mathcal{A}$ (say one of the partial solution we threw out was D_2). Then, if there is D' such that $D_2 \cup D'$ is a solution, then all the connected components in $G - (D_1 \cup D')$ containing at least one green vertex will have a deficient vertex. Just a word of caution that in our actual algorithm in fact we keep a subset of $\mathcal{A}$ of size $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ so that we can also take care of all other properties of a nice deletion set.

We have explained how we will ensure each of the individual properties of a nice deletion set. Now we design a randomized FPT algorithm for the problem. Later we derandomize the algorithm. The algorithm is a DP algorithm in which we have DP table entries indexed exactly in the same way as in the case of the XP algorithm. But instead of keeping $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$, we store a small representative family of $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ which is enough to maintain the correctness of the algorithm. The algorithm uses both color coding and representative family techniques. We have explained that we use color coding to separate the proposed deletion set from its certificate mentioned in Lemma 11. Now our algorithm works with the edge colored graph (edges in E_g are colored black or orange) and output a nice deletion set D, with the property that $D \cap E_g \subseteq E_o$, if there exists such a deletion set. Note that the edges in E_r is uncolored.

Recall that $\mathcal{C}$ is the set of connected components in G such that for each vertex v in the component, $d_G(v) = f(v)$. Now we define a family $\mathcal{J}$ of functions as,

$$\mathcal{J} = \{g : \mathsf{Green} \cup \mathcal{C} \to \{0, 1\}\}.$$

Now we explain how to reduce the size of $\mathcal{D}[T'_g, T'_r, k'_1, k'_2, i, X, Y]$. We say a partial solution $B \in \mathcal{D}[T'_g, T'_r, k'_1, k'_2, i, X, Y]$ is *properly colored*, if $B \cap E_b = \emptyset$. Since our objective is to find out a nice deletion set disjoint from E_b, we delete all partial solutions which contains an edge from E_b. That is, if $B \in \mathcal{D}[T'_g, T'_r, k'_1, k'_2, i, X, Y]$ and $B \cap E_b \neq \emptyset$, then we delete B from $\mathcal{D}[T'_g, T'_r, k'_1, k'_2, i, X, Y]$. So now onwards we assume that for each $B \in \mathcal{D}[T'_g, T'_r, k'_1, k'_2, i, X, Y]$, $B \cap E_b = \emptyset$. Further pruning of the DP table entry $\mathcal{D}[T'_g, T'_r, k'_1, k'_2, i, X, Y]$ is discussed below.

For each valid tuple $(T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y)$ we compute a representative partial solutions for $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ in the increasing order of i and store it instead of $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$. Now we explain how to compute it. First we compute a subfamily $\mathcal{S}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ of $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ using the DP table entries computed for value $i - 1$ and deleting all partial solutions which contain edges from E_b. Now we partition $\mathcal{S}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ according to the refinement of each function in $\mathcal{J}$. That is $\mathcal{S}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y] = \bigcup_{g \in \mathcal{J}} \mathcal{A}_g$ where $\mathcal{A}_g$ is defined as follows. For each $g \in \mathcal{J}$ and $S \cup R \in \mathcal{S}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ where $S \in E(G)$ and $R \in \overline{E(G)}$, $S \cup R \in \mathcal{A}_g$ if the following happens.

- **(i)** For any $v \in \mathsf{Green}$, $g(v) = 1$ if and only if there exists a path from v to a vertex in $\mathsf{def}(G - S, f)$ in $G[E_b \cup (E_r \setminus S)]$ (checking whether there is a witness path that do not use edges in E_o).
- **(ii)** For any $C \in \mathcal{C}$, $g(C) = 1$ if and only if $S \cap E(C) \neq \emptyset$.

Recall that any set $S \cup R \in \mathcal{S}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$ is an independent set of size i in M. Now we compute $\widehat{\mathcal{A}}_g \subseteq^{k-i}_{rep} \mathcal{A}_g$ using Theorem 5. Then we set

$$\mathcal{D}[T'_m, T'_g, \widehat{T'_r, k'_1}, k'_2, i, X, Y] = \bigcup_{g \in \mathcal{J}} \widehat{\mathcal{A}}_g$$

and store it instead of $\mathcal{D}[T'_m, T'_g, T'_r, k'_1, k'_2, i, X, Y]$. We can show the correctness of algorithm even after pruning the DP table entries and our algorithm runs in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ with success probability at least $(1 - \frac{1}{e})$. Our algorithm can be derandomized using $(n, 7k)$-universal sets [16].

References

1. Jin Akiyama and Mikio Kano. Factors and factorizations of graphs – a survey. *Journal of Graph Theory*, 9(1):1–42, 1985.
2. Noga Alon, Daniel Lokshtanov, and Saket Saurabh. Fast FAST. In *Proceedings of the 36th International Colloquium of Automata, Languages and Programming (ICALP)*, volume 5555 of *Lecture Notes in Comput. Sci.*, pages 49–58. Springer, 2009.
3. R. P. Anstee. An algorithmic proof of Tutte's f-factor theorem. *J. Algorithms*, 6(1):112–131, 1985.
4. Rajesh Hemant Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. In *Proceedings of the 53rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 460–469, 2012.
5. Reinhard Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 3rd edition, 2005.
6. Herbert Fleischner. *Eulerian Graphs and Related Topics, Part 1, Volume 1*. Annals of Discrete Mathematics 45. Elsevier, Amsterdam, 1990.
7. Fedor V. Fomin and Yngve Villanger. Subexponential parameterized algorithm for minimum fill-in. *SIAM J. Computing*, 42(6):2197–2216, 2013.
8. Petr A. Golovach. Editing to a connected graph of given degrees. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 8635 of *Lecture Notes in Comput. Sci.*, pages 324–335. Springer, 2014.
9. Ken-ichi Kawarabayashi and Mikkel Thorup. The minimum k-way cut of bounded size is fixed-parameter tractable. In *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 160–169. IEEE, 2011.
10. T Kirkman. On a problem in combinatorics. *Cambridge and Dublin Math. J .*, 2:191–204, 1847.
11. Mekkia Kouider and Preben D. Vestergaard. Connected factors in graphs – a survey. *Graphs and Combinatorics*, 21(1):1–26, 2005.
12. Daniel Lokshtanov, Pranabendu Misra, Fahad Panolan, and Saket Saurabh. Deterministic truncation of linear matroids. In *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 922–934. Springer, 2015.
13. László Lovász and Michael D. Plummer. *Matching theory*. AMS Chelsea Publishing, Providence, RI, 2009.
14. Dániel Marx. A parameterized view on matroid optimization problems. *Theoretical Computers Science*, 410(44):4471–4479, 2009.
15. Luke Mathieson and Stefan Szeider. Editing graphs to satisfy degree constraints: A parameterized approach. *J. Comput. Syst. Sci.*, 78(1):179–191, 2012.
16. Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 182–191. IEEE, 1995.
17. James G. Oxley. *Matroid theory*, volume 21 of *Oxford Graduate Texts in Mathematics*. Oxford university press, 2nd edition, 2010.

18 Julius Petersen. Die Theorie der regulären graphs. *Acta Math.*, 15(1):193–220, 1891. doi:10.1007/BF02392606.

19 Michael D. Plummer. Graph factors and factorization: 1985-2003: A survey. *Discrete Mathematics*, 307(7-8):791–821, 2007.

20 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 887–898. ACM, 2012.

Sub-exponential Approximation Schemes for CSPs: From Dense to Almost Sparse*

Dimitris Fotakis[1], Michael Lampis[2], and Vangelis Th. Paschos[3]

1 **School of Electrical and Computer Engineering, National Technical University of Athens, Greece**
fotakis@cs.ntua.gr

2 **PSL Research University, Université Paris Dauphine, LAMSADE, CNRS UMR7243, France**
michail.lampis@dauphine.fr

3 **PSL Research University, Université Paris Dauphine, LAMSADE, CNRS UMR7243, France**
paschos@lamsade.dauphine.fr

Abstract

It has long been known, since the classical work of (Arora, Karger, Karpinski, JCSS 99), that Max-CUT admits a PTAS on dense graphs, and more generally, Max-k-CSP admits a PTAS on "dense" instances with $\Omega(n^k)$ constraints. In this paper we extend and generalize their exhaustive sampling approach, presenting a framework for $(1-\varepsilon)$-approximating any Max-k-CSP problem in *sub-exponential* time while significantly relaxing the denseness requirement on the input instance.

Specifically, we prove that for any constants $\delta \in (0,1]$ and $\varepsilon > 0$, we can approximate Max-k-CSP problems with $\Omega(n^{k-1+\delta})$ constraints within a factor of $(1-\varepsilon)$ in time $2^{O(n^{1-\delta}\ln n/\varepsilon^3)}$. The framework is quite general and includes classical optimization problems, such as Max-CUT, Max-DICUT, Max-k-SAT, and (with a slight extension) k-Densest Subgraph, as special cases. For Max-CUT in particular (where $k=2$), it gives an approximation scheme that runs in time sub-exponential in n even for "almost-sparse" instances (graphs with $n^{1+\delta}$ edges).

We prove that our results are essentially best possible, assuming the ETH. First, the density requirement cannot be relaxed further: there exists a constant $r < 1$ such that for all $\delta > 0$, Max-k-SAT instances with $O(n^{k-1})$ clauses cannot be approximated within a ratio better than r in time $2^{O(n^{1-\delta})}$. Second, the running time of our algorithm is almost tight *for all densities*. Even for Max-CUT there exists $r < 1$ such that for all $\delta' > \delta > 0$, Max-CUT instances with $n^{1+\delta}$ edges cannot be approximated within a ratio better than r in time $2^{n^{1-\delta'}}$.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases polynomial and subexponential approximation, sampling, randomized rounding

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.37

* This research was supported by the project AlgoNow, co-financed 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, investing in knowledge society through the European Social Fund.

© Dimitris Fotakis, Michael Lampis, and Vangelis Th. Paschos;
licensed under Creative Commons License CC-BY

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 37; pp. 37:1–37:14

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

1 Introduction

The complexity of Constraint Satisfaction Problems (CSPs) has long played a central role in Theoretical Computer Science and it quickly became evident that almost all interesting CSPs are NP-complete [27]. Thus, since approximation algorithms are one of the standard tools for dealing with NP-hard problems, the question of approximating the corresponding optimization problems (Max-CSP) has attracted significant interest over the years [28]. Unfortunately, most CSPs typically resist this approach: not only are they APX-hard [23], but quite often the best polynomial-time approximation ratio we can hope to achieve for them is that guaranteed by a trivial random assignment [21]. This striking behavior is often called *approximation resistance*.

Approximation resistance and other APX-hardness results were originally formulated in the context of *polynomial-time* approximation. It would therefore seem that one conceivable way for working around such barriers could be to consider approximation algorithms running in super-polynomial time, and indeed super-polynomial approximation for NP-hard problems is a topic that has been gaining more attention in the literature recently [10, 7, 6, 11, 12, 13]. Unfortunately, the existence of quasi-linear PCPs with small soundness error, first given in the work of Moshkovitz and Raz [24], established that approximation resistance is a phenomenon that carries over even to *sub-exponential* time approximation, essentially "killing" this approach for CSPs. For instance, we now know that if, for any $\varepsilon > 0$, there exists an algorithm for Max-3-SAT with ratio $7/8 + \varepsilon$ running in time $2^{n^{1-\varepsilon}}$ this would imply the existence of a sub-exponential *exact* algorithm for 3-SAT, disproving the Exponential Time Hypothesis (ETH). It therefore seems that sub-exponential time does not improve the approximability of CSPs, or put another way, for many CSPs obtaining a very good approximation ratio requires almost as much time as solving the problem exactly.

Despite this grim overall picture, many positive approximation results for CSPs have appeared over the years, by taking advantage of the special structure of various classes of instances. One notable line of research in this vein is the work on the approximability of *dense* CSPs, initiated by Arora, Karger and Karpinski [4] and independently by de la Vega [14]. The theme of this set of results is that the problem of maximizing the number of satisfied constraints in a CSP instance with arity k (Max-k-CSP) becomes significantly easier if the instance contains $\Omega(n^k)$ constraints. More precisely, it was shown in [4] that Max-k-CSP admits a *polynomial-time approximation scheme* (PTAS) on dense instances, that is, an algorithm which for any constant $\varepsilon > 0$ can in time polynomial in n produce an assignment that satisfies $(1 - \varepsilon)$OPT constraints. Subsequent work produced a stream of positive [16, 5, 2, 9, 8, 20, 3, 19, 22] (and some negative [15, 1]) results on approximating CSPs which are in general APX-hard, showing that dense instances form an island of tractability where many optimization problems which are normally APX-hard admit a PTAS.

Our contribution. The main goal of this paper is to use the additional power afforded by sub-exponential time to extend this island of tractability as much as possible. To demonstrate the main result, consider a concrete CSP such as Max-3-SAT. As mentioned, we know that sub-exponential time does not in general help us approximate this problem: the best ratio achievable in, say, $2^{\sqrt{n}}$ time is still $7/8$. On the other hand, this problem admits a PTAS on instances with $\Omega(n^3)$ clauses. This density condition is, however, rather strict, so the question we would like to answer is the following: Can we efficiently approximate a larger (and more sparse) class of instances while using sub-exponential time?

In this paper we provide a positive answer to this question, not just for Max-3-SAT, but also for any Max-k-CSP problem. Specifically, we show that for any constants $\delta \in (0, 1]$,

$\varepsilon > 0$ and integer $k \geq 2$, there is an algorithm which achieves a $(1-\varepsilon)$ approximation of Max-k-CSP instances with $\Omega(n^{k-1+\delta})$ constraints in time $2^{O(n^{1-\delta} \ln n/\varepsilon^3)}$. A notable special case of this result is for $k = 2$, where the input instance can be described as a graph. For this case, which contains classical problems such as Max-Cut, our algorithm gives an approximation scheme running in time $2^{O(\frac{n}{\Delta} \ln n/\varepsilon^3)}$ for graphs with average degree Δ. In other words, this is an approximation scheme that runs in time *sub-exponential in* n even for almost sparse instances where the average degree is $\Delta = n^{\delta}$ for some small $\delta > 0$. More generally, our algorithm provides a trade-off between the time available and the density of the instances we can handle. For graph problems ($k = 2$) this trade-off covers the whole spectrum from dense to almost sparse instances, while for general Max-k-CSP, it covers instances where the number of constraints ranges from $\Theta(n^k)$ to $\Theta(n^{k-1})$.

Techniques. The algorithms in this paper are an extension and generalization of the *exhaustive sampling* technique given by Arora, Karger and Karpinski [4], who introduced a framework of smooth polynomial integer programs to give a PTAS for dense Max-k-CSP. The basic idea of that work can most simply be summarized for Max-Cut. This problem can be recast as the problem of maximizing a quadratic function over n boolean variables. This is of course a hard problem, but suppose that we could somehow "guess" for each vertex how many of its neighbors belong in each side of the cut. This would make the quadratic problem linear, and thus much easier. The main intuition now is that, if the graph is dense, we can take a sample of $O(\log n)$ vertices and guess their partition in the optimal solution. Because every non-sample vertex will have "many" neighbors in this sample, we can with high confidence say that we can estimate the fraction of neighbors on each side for all vertices. The work of de la Vega [14] uses exactly this algorithm for Max-Cut, greedily deciding the vertices outside the sample. The work of [4] on the other hand pushed this idea to its logical conclusion, showing that it can be applied to degree-k polynomial optimization problems, by recursively turning them into linear programs whose coefficients are estimated from the sample. The linear programs are then relaxed to produce fractional solutions, which can be rounded back into an integer solution to the original problem.

On a very high level, the approach we follow in this paper retraces the steps of [4]: we formulate Max-k-CSP as a degree-k polynomial maximization problem; we then recursively decompose the degree-k polynomial problem into lower-degree polynomial optimization problems, estimating the coefficients by using a sample of variables for which we try all assignments; the result of this process is an integer linear program, for which we obtain a fractional solution in polynomial time; we then perform randomized rounding to obtain an integer solution that we can use for the original problem.

The first major difference between our approach and [4] is of course that we need to use a larger sample. This becomes evident if one considers Max-Cut on graphs with average degree Δ. In order to get the sampling scheme to work we must be able to guarantee that each vertex outside the sample has "many" neighbors inside the sample, so we can safely estimate how many of them end up on each side of the cut. For this, we need a sample of size at least $n \log n/\Delta$. Indeed, we use a sample of roughly this size, and exhausting all assignments to the sample is what dominates the running time of our algorithm. As we argue later, not only is the sample size we use essentially tight, but more generally the running time of our algorithm is essentially optimal (under the ETH).

Nevertheless, using a larger sample is not in itself sufficient to extend the scheme of [4] to non-dense instances. As observed in [4] "to achieve a multiplicative approximation for dense instances it suffices to achieve an additive approximation for the nonlinear integer

programming problem". In other words, one of the basic ingredients of the analysis of [4] is that additive approximation errors of the order εn^k can be swept under the rug, because we know that in a dense instance the optimal solution has value $\Omega(n^k)$. This is *not* true in our case, and we are therefore forced to give a more refined analysis of the error of our scheme, independently bounding the error introduced in the first step (coefficient estimation) and the last (randomized rounding).

A further, more serious complication arises when considering Max-k-CSP for $k > 2$. Indeed, the idea of using a larger sample size to handle non-dense instances of Max-Cut was already considered in [17], which for $k = 2$ gives an algorithm of similar performance to the one we present in this paper. The main obstacle to extending such an algorithm to the general case of $k > 2$ is that the scheme of [4] recursively decomposes dense instances into lower-order polynomials which roughly retain the same "denseness". This property seems much harder to transfer to the non-dense case, because intuitively if we start from a non-dense instance the decomposition could end up producing some dense and some sparse sub-problems. Here we present a scheme that approximates Max-k-CSP with $\Omega(n^{k-1+\delta})$ constraints, but does not seem to extend to instances with fewer than n^{k-1} constraints. As we will see, there seems to be a fundamental complexity-theoretic justification explaining exactly why this decomposition method cannot be extended further.

Hardness. What makes the results of this paper more interesting is that we can establish that in many ways they are essentially best possible, if one assumes the ETH. In particular, there are at least two ways in which one may try to improve on these results further: one would be to improve the running time of our algorithm, while another would be to extend the algorithm to the range of densities it cannot currently handle. In Section 6 we show that both of these approaches would face significant barriers. Our starting point is the fact that (under ETH) it takes exponential time to approximate Max-Cut arbitrarily well on sparse instances, which is a consequence of the existence of quasi-linear PCPs. By manipulating such Max-Cut instances, we are able to show that for *any* average degree $\Delta = n^\delta$ with $\delta < 1$ the time needed to approximate Max-Cut arbitrarily well almost matches the performance of our algorithm. Furthermore, starting from sparse Max-Cut instances, we can produce instances of Max-k-SAT with $O(n^{k-1})$ clauses while preserving hardness of approximation. This gives a complexity-theoretic justification for our difficulties in decomposing Max-k-CSP instances with less than n^{k-1} constraints. We note that the hardness results we present here refute (subject to the ETH) a conjecture made in [17] that a better time-density trade-off can be achieved for Max-Cut.

2 Notation and Preliminaries

An n-variate degree-d polynomial $p(\vec{x})$ is *β-smooth* [4], for some constant $\beta \geq 1$, if for every $\ell \in \{0, \ldots, d\}$, the absolute value of each coefficient of each degree-ℓ monomial in the expansion of $p(\vec{x})$ is at most $\beta n^{d-\ell}$. An n-variate degree-d β-smooth polynomial $p(\vec{x})$ is *δ-bounded*, for some constant $\delta \in (0, 1]$, if for every ℓ, the sum, over all degree-ℓ monomials in $p(\vec{x})$, of the absolute values of their coefficients is $O(\beta n^{d-1+\delta})$. Therefore, for any n-variate degree-d β-smooth δ-bounded polynomial $p(\vec{x})$ and any $\vec{x} \in \{0,1\}^n$, $|p(\vec{x})| = O(d\beta n^{d-1+\delta})$.

Our algorithms for Max-Cut, Max-k-SAT, and Max-k-CSP are obtained by reducing to the following problem: Given an n-variate d-degree β-smooth δ-bounded polynomial $p(\vec{x})$, we seek a binary vector $\vec{x}^* \in \{0,1\}^n$ that maximizes p.

As in [4, Lemma 3.1], our general approach is motivated by the fact that any n-variate

d-degree β-smooth polynomial $p(\vec{x})$ can be naturally decomposed into a collection of n polynomials $p_j(\vec{x})$. Each of them has degree $d-1$ and at most n variables and is β-smooth.

▶ **Lemma 1** ([4]). *Let $p(\vec{x})$ be any n-variate degree-d β-smooth polynomial. Then, there exist a constant c and degree-$(d-1)$ β-smooth polynomials $p_j(\vec{x})$ such that $p(\vec{x}) = c + \sum_{j=1}^{n} x_j p_j(\vec{x})$.*

Let $G(V, E)$ be a (simple) graph with n vertices and m edges. For each vertex $i \in V$, $N(i)$ denotes i's neighborhood in G, i.e., $N(i) = \{j \in V : \{i, j\} \in E\}$. We let $\deg(i) = |N(i)|$ be the degree of i in G and $\Delta = 2|E|/n$ denote the average degree of G. We say that a graph G is δ-*almost sparse*, for some constant $\delta \in (0, 1]$, if $m = \Omega(n^{1+\delta})$ (and thus, $\Delta = \Omega(n^{\delta})$).

In Max-Cut, we seek a partitioning of the vertices of G into two sets S_0 and S_1 so that the number of edges with endpoints in S_0 and S_1 is maximized. If G has m edges, the number of edges in the optimal cut is at least $m/2$.

In k-Densest Subgraph, given an undirected graph $G(V, E)$, we seek a subset C of k vertices so that the induced subgraph $G[C]$ has a maximum number of edges.

An instance of (boolean) Max-k-CSP with n variables consists of m boolean constraints $f_1, \ldots, f_m$, where each $f_j : \{0, 1\}^k \to \{0, 1\}$ depends on k variables and is satisfiable, i.e., f_j evaluates to 1 for some truth assignment. We seek a truth assignment to the variables that maximizes the number of satisfied constraints. Max-k-SAT is a special case of Max-k-CSP where each constraint f_j is a disjunction of k literals. An averaging argument implies that the optimal assignment of a Max-k-CSP (resp. Max-k-SAT) instance with m constraints satisfies at least $2^{-k}m$ (resp. $(1-2^{-k})m$) of them. We say that an instance of Max-k-CSP is δ-*almost sparse*, for some constant $\delta \in (0, 1]$, if the number of constraints is $m = \Omega(n^{k-1+\delta})$.

Using standard arithmetization techniques (see e.g., [4, Sec. 4.3]), we can reduce any instance of Max-k-CSP with n variables to an n-variate degree-k polynomial $p(\vec{x})$ so that the optimal truth assignment for Max-k-CSP corresponds to a maximizer $\vec{x}^* \in \{0, 1\}$ of $p(\vec{x})$ and the value of the optimal Max-k-CSP solution is equal to $p(\vec{x}^*)$. Such a polynomial $p(\vec{x})$ is β-smooth, for an appropriate constant β that may depend on k, and has at least m and at most $4^k m$ monomials. Moreover, if the instance of Max-k-CSP has $m = \Theta(n^{k-1+\delta})$ constraints, then $p(\vec{x})$ is δ-bounded and its maximizer $\vec{x}^*$ has $p(\vec{x}^*) = \Omega(n^{k-1+\delta})$.

Since each k-tuple of variables can appear in at most 2^k different constraints, $p(\vec{x})$ is β-smooth, for $\beta \in [1, 4^k]$, and has at least m and at most $4^k m$ monomials. Moreover, if the instance of Max-k-CSP has $m = \Theta(n^{k-1+\delta})$ constraints, then $p(\vec{x})$ is δ-bounded and its maximizer $\vec{x}^*$ has $p(\vec{x}^*) = \Omega(n^{k-1+\delta})$.

An algorithm has *approximation ratio* $\rho \in (0, 1]$ (or is ρ-*approximate*) if for all instances, the value of its solution is at least ρ times the value of the optimal solution. For graphs with n vertices or CSPs with n variables, we say that an event E happens with high probability (or whp.), if E happens with probability at least $1 - 1/n^c$, for some constant $c \geq 1$. For brevity and clarity, we sometimes write $\alpha \in (1 \pm \epsilon_1)\beta \pm \epsilon_2\gamma$, for some constants $\epsilon_1, \epsilon_2 > 0$, to denote that $(1-\epsilon_1)\beta - \epsilon_2\gamma \leq \alpha \leq (1+\epsilon_1)\beta + \epsilon_2\gamma$.

3 Approximating Max-CUT in Almost Sparse Graphs

In this section, we apply our approach to Max-Cut, which serves as a convenient example and allows us to present the intuition and the main ideas.

The Max-Cut problem in a graph $G(V, E)$ is equivalent to maximizing, over all binary vectors $\vec{x} \in \{0, 1\}^n$, the following n-variate degree-2 2-smooth polynomial

$$p(\vec{x}) = \sum_{\{i,j\} \in E} \left(x_i(1 - x_j) + x_j(1 - x_i)\right).$$

Setting a variable x_i to 0 indicates that the corresponding vertex i is assigned to the left side of the cut, i.e., to S_0, and setting x_i to 1 indicates that vertex i is assigned to the right side of the cut, i.e., to S_1. We assume that G is δ-almost sparse and thus, has $m = \Omega(n^{1+\delta})$ edges and average degree $\Delta = \Omega(n^{\delta})$. Moreover, if $m = \Theta(n^{1+\delta})$, $p(\vec{x})$ is δ-bounded, since for each edge $\{i,j\} \in E$, the monomial $x_i x_j$ appears with coefficient -2 in the expansion of p, and for each vertex $i \in V$, the monomial x_i appears with coefficient $\deg(i)$ in the expansion of p. Therefore, for $\ell \in \{1,2\}$, the sum of the absolute values of the coefficients of all monomials of degree ℓ is at most $2m = O(n^{1+\delta})$.

Next, we extend and generalize the approach of [4] and show how to $(1-\varepsilon)$-approximate the optimal cut, for any constant $\varepsilon > 0$, in time $2^{O(n \ln n/(\Delta\varepsilon^3))}$ (see Theorem 5). The running time is subexponential in n, if G is δ-almost sparse.

3.1 Outline and Main Ideas

Applying Lemma 1, we can write the smooth polynomial $p(\vec{x})$ as

$$p(\vec{x}) = \sum_{j \in V} x_j(\deg(j) - p_j(\vec{x}))\,, \tag{1}$$

where $p_j(\vec{x}) = \sum_{i \in N(j)} x_i$ is a degree-1 1-smooth polynomial that indicates how many neighbors of vertex j are in S_1 in the solution corresponding to $\vec{x}$. The key observation, due to [4], is that if we have a good estimation ρ_j of the value of each p_j at the optimal solution $\vec{x}^*$, then approximate maximization of $p(\vec{x})$ can be reduced to the solution of the following Integer Linear Program:

$$\max \sum_{j \in V} y_j(\deg(j) - \rho_j) \tag{IP}$$

$$\text{s.t.} \quad (1-\epsilon_1)\rho_j - \epsilon_2\Delta \le \sum_{i \in N(j)} y_i \le (1+\epsilon_1)\rho_j + \epsilon_2\Delta \quad \forall j \in V$$

$$y_j \in \{0,1\} \qquad \forall j \in V$$

The constants $\epsilon_1, \epsilon_2 > 0$ and the estimations $\rho_j \ge 0$ are computed so that the optimal solution $\vec{x}^*$ is a feasible solution to (IP). We always assume wlog. that $0 \le \sum_{i \in N(j)} y_i \le \deg(j)$, i.e., we let the lhs of the j-th constraint be $\max\{(1-\epsilon_1)\rho_j - \epsilon_2\Delta, 0\}$ and the rhs be $\min\{(1+\epsilon_1)\rho_j + \epsilon_2\Delta, \deg(j)\}$. Clearly, if $\vec{x}^*$ is a feasible solution to (IP), it remains a feasible solution after this modification. We let (LP) denote the Linear Programming relaxation of (IP), where each $y_j \in [0,1]$.

The first important observation is that for any $\epsilon_1, \epsilon_2 > 0$, we can compute estimations ρ_j, by exhaustive sampling, so that $\vec{x}^*$ is a feasible solution to (IP) with high probability (see Lemma 2). The second important observation is that the objective value of any feasible solution $\vec{y}$ to (LP) is close to $p(\vec{y})$ (see Lemma 3). Namely, for any feasible solution $\vec{y}$, $\sum_{j \in V} y_j(\deg(j) - \rho_j) \approx p(\vec{y})$.

Based on these observations, the approximation algorithm performs the following steps:

1. We guess a sequence of estimations $\rho_1, \ldots, \rho_n$, by exhaustive sampling, so that $\vec{x}^*$ is a feasible solution to the resulting (IP) (see Section 3.2 for the details).
2. We formulate (IP) and find an optimal fractional solution $\vec{y}^*$ to (LP).
3. We obtain an integral solution $\vec{z}$ by applying randomized rounding to $\vec{y}^*$ (and the method of conditional probabilities, as in [26, 25]).

To see that this procedure indeed provides a good approximation to $p(\vec{x}^*)$, we observe that:

$$p(\vec{z}) \approx \sum_{j \in V} z_j(\deg(j) - \rho_j) \approx \sum_{j \in V} y_j^*(\deg(j) - \rho_j) \geq \sum_{j \in V} x_j^*(\deg(j) - \rho_j) \approx p(\vec{x}^*)\,. \qquad (2)$$

The first approximation holds because $\vec{z}$ is an (almost) feasible solution to (IP) (see Lemma 4), the second approximation holds because the objective value of $\vec{z}$ is a good approximation to the objective value of $\vec{y}^*$, due to randomized rounding, the inequality holds because $\vec{x}^*$ is a feasible solution to (LP) and the final approximation holds because $\vec{x}^*$ is a feasible solution to (IP).

In Sections 3.3 and 3.4, we make the notion of approximation precise so that $p(\vec{z}) \geq (1-\varepsilon)p(\vec{x}^*)$. As for the running time, it is dominated by the time required for the exhaustive-sampling step. Since we do not know $\vec{x}^*$, we need to run the steps (2) and (3) above for every sequence of estimations produced by exhaustive sampling. So, the outcome of the approximation scheme is the best of the integral solutions $\vec{z}$ produced in step (3) over all executions of the algorithm. In Section 3.2, we show that a sample of size $O(n \ln n/\Delta)$ suffices for the computation of estimations ρ_j so that $\vec{x}^*$ is a feasible solution to (IP) with high probability. If G is δ-almost sparse, the sample size is sublinear in n and the running time is subexponential in n.

3.2 Obtaining Estimations ρ_j by Exhaustive Sampling

To obtain good estimations ρ_j of the values $p_j(\vec{x}^*) = \sum_{i \in N(j)} x_i^*$, i.e., of the number of j's neighbors in S_1 in the optimal cut, we take a random sample $R \subseteq V$ of size $\Theta(n \ln n/\Delta)$ and try exhaustively all possible assignments of the vertices in R to S_0 and S_1. If $\Delta = \Omega(n^\delta)$, we have $2^{O(n \ln n/\Delta)} = 2^{O(n^{1-\delta} \ln n)}$ different assignments. For each assignment, described by a 0/1 vector $\vec{x}$ restricted to R, we compute an estimation $\rho_j = (n/|R|) \sum_{i \in N(j) \cap R} x_i$, for each vertex $j \in V$, and run the steps (2) and (3) of the algorithm above. Since we try all possible assignments, one of them agrees with $\vec{x}^*$ on all vertices of R. So, for this assignment, the estimations computed are $\rho_j = (n/|R|) \sum_{i \in N(j) \cap R} x_i^*$. The following shows that for these estimations, we have that $p_j(\vec{x}^*) \approx \rho_j$ with high probability.

▶ **Lemma 2.** *Let $\vec{x}$ be any binary vector. For all $\alpha_1, \alpha_2 > 0$, we let $\gamma = \Theta(1/(\alpha_1^2 \alpha_2))$ and let R be a multiset of $r = \gamma n \ln n/\Delta$ vertices chosen uniformly at random with replacement from V. For any vertex j, if $\rho_j = (n/r) \sum_{i \in N(j) \cap R} x_i$ and $\hat{\rho}_j = \sum_{i \in N(j)} x_i$, with probability at least $1 - 2/n^3$,*

$$(1-\alpha_1)\hat{\rho}_j - (1-\alpha_1)\alpha_2\Delta \leq \rho_j \leq (1+\alpha_1)\hat{\rho}_j + (1+\alpha_1)\alpha_2\Delta\,. \qquad (3)$$

We note that $\rho_j \geq 0$ and always assume that $\rho_j \leq \deg(j)$, since if ρ_j satisfies (3), $\min\{\rho_j, \deg(j)\}$ also satisfies (3). For all $\epsilon_1, \epsilon_2 > 0$, setting $\alpha_1 = \frac{\epsilon_1}{1+\epsilon_1}$ and $\alpha_2 = \epsilon_2$ in Lemma 2, and taking the union bound over all vertices, we obtain that for $\gamma = \Theta(1/(\epsilon_1^2 \epsilon_2))$, with probability at least $1 - 2/n^2$, the following holds for all vertices $j \in V$:

$$(1-\epsilon_1)\rho_j - \epsilon_2\Delta \leq \hat{\rho}_j \leq (1+\epsilon_1)\rho_j + \epsilon_2\Delta\,. \qquad (4)$$

Therefore, with probability at least $1 - 2/n^2$, the optimal cut $\vec{x}^*$ is a feasible solution to (IP) with the estimations ρ_j obtained by restricting $\vec{x}^*$ to the vertices in R.

3.3 The Cut Value of Feasible Solutions

We next show that the objective value of any feasible solution $\vec{y}$ to (LP) is close to $p(\vec{y})$. Therefore, assuming that $\vec{x}^*$ is feasible, any good approximation to (IP) is a good approximation to the optimal cut.

▶ **Lemma 3.** *Let $\rho_1, \ldots, \rho_n$ be non-negative numbers and $\vec{y}$ be any feasible solution to (LP). Then,*

$$p(\vec{y}) \in \sum_{j \in V} y_j(\deg(j) - \rho_j) \pm 2(\epsilon_1 + \epsilon_2)m\,. \tag{5}$$

3.4 Randomized Rounding of the Fractional Optimum

As a last step, we show how to round the fractional optimum $\vec{y}^* = (y_1^*, \ldots, y_n^*)$ of (LP) to an integral solution $\vec{z} = (z_1, \ldots, z_n)$ that almost satisfies the constraints of (IP).

To this end, we use randomized rounding, as in [26]. In particular, we set independently each z_j to 1, with probability y_j^*, and to 0, with probability $1 - y_j^*$. By Chernoff bounds [1], we obtain that with probability at least $1 - 2/n^8$, for each vertex j,

$$(1-\epsilon_1)\rho_j - \epsilon_2\Delta - 2\sqrt{\deg(j)\ln(n)} \leq \sum_{i \in N(j)} z_i \leq (1+\epsilon_1)\rho_j + \epsilon_2\Delta + 2\sqrt{\deg(j)\ln(n)}\,. \tag{6}$$

Specifically, the inequality above follows from the Chernoff bound in footnote 1, with $k = \deg(j)$ and $t = 2\sqrt{\deg(j)\ln(n)}$, since $\mathbb{E}[\sum_{i \in N(j)} z_j] = \sum_{i \in N(j)} y_j^* \in (1 \pm \epsilon_1)\rho_j \pm \epsilon_2\Delta$. By the union bound, (6) is satisfied with probability at least $1 - 2/n^7$ for all vertices j.

By linearity of expectation, $\mathbb{E}[\sum_{j \in V} z_j(\deg(j) - \rho_j)] = \sum_{j \in V} y_j^*(\deg(j) - \rho_j)$. Moreover, since the probability that $\vec{z}$ does not satisfy (6) for some vertex j is at most $2/n^7$ and since the objective value of (IP) is at most n^2, the expected value of a rounded solution $\vec{z}$ that satisfies (6) for all vertices j is least $\sum_{j \in V} y_j^*(\deg(j) - \rho_j) - 1$ (assuming that $n \geq 2$). Using the method of conditional expectations, as in [25], we can find in (deterministic) polynomial time an integral solution $\vec{z}$ that satisfies (6) for all vertices j and has $\sum_{j \in V} z_j(\deg(j) - \rho_j) \geq \sum_{j \in V} y_j^*(\deg(j) - \rho_j) - 1$. Next, we sometimes abuse the notation and refer to such an integral solution $\vec{z}$ (computed deterministically) as the integral solution obtained from $\vec{y}^*$ by randomized rounding.

The following is similar to Lemma 3 and shows that the objective value $p(\vec{z})$ of the rounded solution $\vec{z}$ is close to the optimal value of (LP).

▶ **Lemma 4.** *Let $\vec{y}^*$ be the optimal solution of (LP) and let $\vec{z}$ be the integral solution obtained from $\vec{y}^*$ by randomized rounding (and the method of conditional expectations). Then,*

$$p(\vec{z}) \in \sum_{j \in V} y_j^*(\deg(j) - \rho_j) \pm 3(\epsilon_1 + \epsilon_2)m\,. \tag{7}$$

3.5 Putting Everything Together

Therefore, for any $\varepsilon > 0$, if G is δ-almost sparse and $\Delta = n^\delta$, the algorithm described in Section 3.1, with sample size $\Theta(n \ln n/(\varepsilon^3\Delta))$, computes estimations ρ_j such that the

[1] We use the following standard Chernoff bound (see e.g., [18, Theorem 1.1]): Let $Y_1, \ldots, Y_k$ independent random variables in $[0,1]$ and let $Y = \sum_{j=1}^k Y_j$. Then for all $t > 0$, $\mathbb{P}[|Y - \mathbb{E}[Y]| > t] \leq 2\exp(-2t^2/k)$.

optimal cut $\vec{x}^*$ is a feasible solution to (IP) whp. Hence, by the analysis above, the algorithm approximates the value of the optimal cut $p(\vec{x}^*)$ within an additive term of $O(\varepsilon m)$. Specifically, setting $\epsilon_1 = \epsilon_2 = \varepsilon/16$, the value of the cut $\vec{z}$ produced by the algorithm satisfies the following with probability at least $1 - 2/n^2$:

$$p(\vec{z}) \geq \sum_{j \in V} y_j^*(\deg(j) - \rho_j) - \frac{3\varepsilon m}{8} \geq \sum_{j \in V} x_j^*(\deg(j) - \rho_j) - \frac{3\varepsilon m}{8} \geq p(\vec{x}^*) - \frac{\varepsilon m}{2} \geq (1-\varepsilon)p(\vec{x}^*)\,.$$

The first inequality follows from Lemma 4, the second inequality holds because $\vec{y}^*$ is the optimal solution to (LP) and $\vec{x}^*$ is feasible for (LP), the third inequality follows from Lemma 3 and the fourth inequality holds because the optimal cut has at least $m/2$ edges.

▶ **Theorem 5.** *Let $G(V, E)$ be a δ-almost sparse graph with n vertices. Then, for any $\varepsilon > 0$, we can compute, in time $2^{O(n^{1-\delta} \ln n/\varepsilon^3)}$ and with probability at least $1 - 2/n^2$, a cut $\vec{z}$ of G with value $p(\vec{z}) \geq (1-\varepsilon)p(\vec{x}^*)$, where $\vec{x}^*$ is the optimal cut.*

4 Approximate Maximization of Smooth Polynomials

Generalizing the ideas applied to Max-Cut, we arrive at the main algorithmic result of the paper: an algorithm to approximately optimize β-smooth δ-bounded polynomials $p(\vec{x})$ of degree d over all binary vectors $\vec{x} \in \{0,1\}^n$. The intuition and the main ideas are quite similar to those in Section 3, but the details are significantly more involved because we are forced to recursively decompose degree d polynomials to eventually obtain a linear program.

▶ **Theorem 6.** *Let $p(\vec{x})$ be an n-variate degree-d β-smooth δ-bounded polynomial. Then, for any $\varepsilon > 0$, we can compute, in time $2^{O(d^7\beta^3 n^{1-\delta} \ln n/\varepsilon^3)}$ and with probability at least $1 - 8/n^2$, a binary vector $\vec{z}$ so that $p(\vec{z}) \geq p(\vec{x}^*) - \varepsilon n^{d-1+\delta}$, where $\vec{x}^*$ is the maximizer of $p(\vec{x})$.*

Max-k-CSP. Using Theorem 6 it is a straightforward observation that for any Max-k-CSP problem (for constant k) we can obtain an algorithm which, given a Max-k-CSP instance with $\Omega(n^{k-1+\delta})$ constraints for some $\delta > 0$, for any $\varepsilon > 0$ returns an assignment that satisfies $(1-\varepsilon)$OPT constraints in time $2^{O(n^{1-\delta} \ln n/\varepsilon^3)}$. This follows from Theorem 6 using two observations: first, the standard arithmetization of Max-k-CSP described in Section 2 produces a degree-k β-smooth δ-bounded polynomial for β depending only on k. Second, the optimal solution of such an instance satisfies at least $\Omega(n^{k-1+\delta})$ constraints, therefore the additive error given in Theorem 6 is $O(\varepsilon\text{OPT})$. This algorithm for Max-k-CSP contains as special cases algorithm for various standard problems such as Max-Cut, Max-DICUT and Max-k-SAT.

5 Approximating the k-Densest Subgraph in Almost Sparse Graphs

In this section, we present an extension of the algorithms we have presented which can be used to approximate k-Densest Subgraph in δ-almost sparse graphs. This is a problem also handled in [4], but only for the case where $k = \Omega(n)$. Smaller values of k cannot be handled by the scheme of [4] for dense graphs because when $k = o(n)$ the optimal solution has objective value much smaller than the additive error of εn^2 inherent in their scheme.

Here we obtain a sub-exponential time approximation scheme that works on graphs with $\Omega(n^{1+\delta})$ edges *for all* k by judiciously combining two approaches: when k is relatively large, we use a sampling approach similar to Max-Cut; when k is small, we can resort to the naïve algorithm that tries all $\binom{n}{k}$ possible solutions. We select (with some foresight) the

threshold between the two algorithms to be $k = \Omega(n^{1-\delta/3})$, so that in the end we obtain an approximation scheme with running time of $2^{O(n^{1-\delta/3}\ln n)}$, that is, slightly slower than the approximation scheme for Max-Cut. It is clear that the brute-force algorithm achieves this running time for $k = O(n^{1-\delta/3})$, so in the remainder we focus on the case of large k.

The k-Densest Subgraph problem in a graph $G(V, E)$ is equivalent to maximizing, over all vectors $\vec{x} \in \{0,1\}^n$, the n-variate degree-2 1-smooth polynomial $p(\vec{x}) = \sum_{\{i,j\}\in E} x_i x_j$, under the linear constraint $\sum_{j\in V} x_j = k$. Setting a variable x_i to 1 indicates that the vertex i is included in the set C that induces a dense subgraph $G[C]$ of k vertices. We assume that G is δ-almost sparse i.e. $m = \Omega(n^{1+\delta})$ edges. As usual, $\vec{x}$ denotes the optimal solution.

The algorithm follows the same general approach and the same basic steps as the algorithm for Max-Cut in Section 3. In the following, we highlight only the differences.

Obtaining Estimations by Exhaustive Sampling. We first observe that if G is δ-almost sparse and $k = \Omega(n^{1-\delta/3})$, a random subset of k vertices contains $\Omega(n^{1+\delta/3})$ edges in expectation. We thus assume that the optimal solution induces at least $\Omega(n^{1+\delta/3})$ edges.

Working as in Section 3.2, we use exhaustive sampling and obtain for each vertex $j \in V$, an estimation ρ_j of j's neighbors in the optimal dense subgraph, i.e., ρ_j is an estimation of $\hat{\rho}_j = \sum_{i\in N(j)} x^*_i$. For the analysis, we apply Lemma 2 with $n^{\delta/3}$, instead of Δ, or in other words, we use a sample of size $\Theta(n^{1-\delta/3}\ln n)$. The reason is that we can only tolerate an additive error of $\varepsilon n^{1+\delta/3}$, by the lower bound on the optimal solution observed in the previous paragraph. Then, the running time due to exhaustive sampling is $2^{O(n^{1-\delta/3}\ln n)}$.

By Lemma 2 and the discussion following it in Section 3.2, we obtain that for all $\epsilon_1, \epsilon_2 > 0$, if we use a sample of the size $\Theta(n^{1-\delta/3}\ln n/(\epsilon_1^2\epsilon_2))$, with probability at least $1 - 2/n^2$, the following holds for all estimations ρ_j and all vertices $j \in V$:

$$(1-\epsilon_1)\rho_j - \epsilon_2 n^{\delta/3} \le \hat{\rho}_j \le (1+\epsilon_1)\rho_j + \epsilon_2 n^{\delta/3} \tag{8}$$

Linearizing the Polynomial. Applying Lemma 1, we can write the polynomial $p(\vec{x})$ as $p(\vec{x}) = \sum_{j\in V} x_j p_j(\vec{x})$, where $p_j(\vec{x}) = \sum_{i\in N(j)} x_i$ is a degree-1 1-smooth polynomial that indicates how many neighbors of vertex j are in C in the solution corresponding to $\vec{x}$. Then, using the estimations ρ_j of $\sum_{i\in N(j)} x^*_i$, obtained by exhaustive sampling, we have that approximate maximization of $p(\vec{x})$ can be reduced to the solution of the following ILP:

$$\begin{aligned} &\max \sum_{j\in V} y_j\rho_j && \text{(IP}'\text{)}\\ \text{s.t.}\quad &(1-\epsilon_1)\rho_j - \epsilon_2 n^{\delta/3} \le \sum_{i\in N(j)} y_i \le (1+\epsilon_1)\rho_j + \epsilon_2 n^{\delta/3} \quad \forall j \in V\\ &\sum_{i\in V} y_i = k \end{aligned}$$

By (8), if the sample size is $|R| = \Theta(n^{1-\delta/3}\ln n/(\epsilon_1^2\epsilon_2))$, with probability at least $1-2/n^2$, the densest subgraph $\vec{x}^*$ is a feasible solution to (IP$'$) with the estimations ρ_j obtained by restricting $\vec{x}^*$ to the vertices in R. In the following, we let (LP$'$) denote the Linear Programming relaxation of (IP$'$), where each $y_j \in [0,1]$.

The Number of Edges in Feasible Solutions. We next show that the objective value of any feasible solution $\vec{y}$ to (LP$'$) is close to $p(\vec{y})$. Therefore, assuming that $\vec{x}^*$ is feasible, any good approximation to (IP$'$) is a good approximation to the densest subgraph.

▶ **Lemma 7.** *Let $\rho_1, \ldots, \rho_n$ be non-negative numbers and $\vec{y}$ be any feasible solution to (LP′). Then,*

$$p(\vec{y}) \in (1 \pm \epsilon_1) \sum_{j \in V} y_j \rho_j \pm \epsilon_2 n^{1+\delta/3} . \tag{9}$$

Randomized Rounding of the Fractional Optimum. As a last step, we show how to round the fractional optimum $\vec{y}^* = (y_1^*, \ldots, y_n^*)$ of (LP′) to an integral solution $\vec{z} = (z_1, \ldots, z_n)$ that almost satisfies the constraints of (IP′). We use randomized rounding, as for Max-CUT.

▶ **Lemma 8.** *Let $\vec{y}^*$ be the optimal solution of (LP′) and let $\vec{z}$ be the integral solution obtained from $\vec{y}^*$ by randomized rounding (and the method of conditional expectations). Then,*

$$p(\vec{z}) \in (1 \pm \epsilon_1)^2 \sum_{j \in V} y_j^* \rho_j \pm 3\epsilon_2 n^{1+\delta/3} . \tag{10}$$

We thus arrive to the main theorem of this section.

▶ **Theorem 9.** *Let $G(V, E)$ be a δ-almost sparse graph with n vertices. Then, for any integer $k \geq 1$ and for any $\varepsilon > 0$, we can compute, in time $2^{O(n^{1-\delta/3} \ln n/\varepsilon^3)}$ and with probability at least $1 - 2/n^2$, an induced subgraph $\vec{z}$ of G with k vertices whose number of edges satisfies $p(\vec{z}) \geq (1-\varepsilon)p(\vec{x}^*)$, where $\vec{x}^*$ is the number of edges in the k-Densest Subgraph of G.*

6 Lower Bounds

We now give some lower bound arguments showing that the schemes we have presented are, in some senses, likely to be almost optimal. Our complexity assumption will be the ETH, which states that no algorithm can solve instances of 3-SAT of size n in time $2^{o(n)}$.

There are two natural ways in which one may hope to improve or extend the algorithms we have presented so far: relaxing the density requirement or decreasing the running time. First, recall that the algorithm we have given for Max-k-CSP works in the density range between n^k and n^{k-1}. Here, we give a reduction establishing that it's unlikely that this can be improved. Our starting point is the following (known) inapproximability result.

▶ **Theorem 10.** *There exist $c, s \in (0,1)$ with $c > s$ such that for all $\epsilon > 0$ we have: if there exists an algorithm which, given an n-vertex 5-regular instance of Max-CUT, can distinguish between the case where a solution cuts at least a c fraction of the edges and the case where all solutions cut at most an s fraction of the edges in time $2^{n^{1-\epsilon}}$ then the ETH fails.*

▶ **Theorem 11.** *There exists $r > 1$ such that for all $\epsilon > 0$ and all (fixed) integers $k \geq 3$ we have the following: if there exists an algorithm which r-approximates Max-k-SAT on instances with $\Omega(n^{k-1})$ clauses in time $2^{n^{1-\epsilon}}$ then the ETH fails.*

Proof. We reduce a Max-CUT instance from Theorem 10 to Max-2-SAT: the set of variables is the set of vertices; for each edge (u, v) we include the clauses $(u \vee v)$ and $(\neg u \vee \neg v)$. The new instance has n variables and $5n$ clauses and there exist constants c, s such that either some assignment satisfies $5cn$ clauses or all assignments satisfy at most $5sn$ of them.

Fix k and add to the instance $(k-2)n$ new variables $x_{(i,j)}$, $i \in \{1, \ldots, k-2\}$, $j \in \{1, \ldots, n\}$. We perform the following transformation: for each clause $(l_1 \vee l_2)$ and for each tuple $(i_1, i_2, \ldots, i_{k-2}) \in \{1, \ldots, n\}^{k-2}$ we construct 2^{k-2} new clauses of size k. The first two literals of these clauses are l_1, l_2. The rest consist of the variables $x_{(1,i_1)}, x_{(2,i_2)}, \ldots, x_{(k,i_{k-2})}$, but in each clause a different set of variables is negated. In other words, to construct a clause

of the new instance we select a clause of the original instance, one variable from each of the groups of n new variables, and a subset of these variables to be negated.

First, observe that the new instance has $5n^{k-1}2^k$ clauses and $(k-1)n$ variables, which satisfies the density conditions. Consider an assignment of the original formula. Any satisfied clause has now been replaced by $n^{k-2}2^k$ satisfied clauses, while for an unsatisfied clause any assignment to the new variables satisfies exactly $n^{k-2}(2^k-1)$ clauses. Thus, for fixed k, there exist constants s', c' such that either a c' fraction of the clauses of the new instance is satisfiable or at most a s' fraction is. If we had an approximation algorithm with ratio better than c'/s' running in time $2^{N^{1-\epsilon}}$, where N is the number of variables of the new instance, we could use it to decide the original instance in time that would disprove the ETH. ◀

A second possible avenue for improvement may be to consider potential speedups of our algorithms. We give an almost tight answer to such questions via the following theorem.

▶ **Theorem 12.** *There exists $r > 1$ such that for all $\epsilon > 0$ we have the following: if there exists an algorithm which, for some $\Delta = o(n)$, r-approximates* Max-*CUT on n-vertex Δ-regular graphs in time $2^{(n/\Delta)^{1-\epsilon}}$ then the ETH fails.*

Proof. Without loss of generality we prove the theorem for the case when the degree is a multiple of 10. Consider an instance $G(V, E)$ of Max-CUT as given by Theorem 10. Let $n = |V|$ and suppose that the desired degree is $d = 10\Delta$, where Δ is a function of n. We construct a graph G' as follows: for each vertex $u \in V$ we introduce Δ new vertices $u_1, \ldots, u_\Delta$ as well as 5Δ "consistency" vertices $c_1^u, \ldots, c_{5\Delta}^u$. For every edge $(u, v) \in E$ we add all edges (u_i, v_j) for $i, j \in \{1, \ldots, \Delta\}$. Also, for every $u \in V$ we add all edges (u_i, c_j^u), for $i \in \{1, \ldots, \Delta\}$ and $j \in \{1, \ldots, 5\Delta\}$. This completes the construction.

The graph we have constructed is 10Δ-regular and is made up of $6\Delta n$ vertices. Consider an optimal cut and observe that, for a given $u \in V$ all the vertices c_i^u can be assumed to be on the same side of the cut, since they all have the same neighbors. Furthermore, for a given $u \in V$, all vertices u_i can be assumed to be on the same side of the cut, namely on the side opposite that of c_i^u, since the vertices c_i^u are a majority of the neighborhood of each u_i. With this observation it is easy to construct a one-to-one correspondence between cuts in G and locally optimal cuts in G'.

Consider a cut that cuts $c|E|$ edges of G. If we set all u_i of G' on the same side as u is in G we cut $c|E|\Delta^2$ edges of the form (u_i, v_j). Furthermore, by placing the c_i^u on the opposite side of u_i we cut $5\Delta^2|V|$ edges. Thus the max cut of G' is at least $c|E|\Delta^2 + 5\Delta^2|V|$. Using the observations on locally optimal cuts of G' we can conclude that if G' has a cut with $s|E|\Delta^2 + 5\Delta^2|V|$ edges, then G has a cut with $s|E|$ edges. Having $2|E| = 5|V|$ (since G is 5-regular) we get a constant ratio r between the size of the cut of G' in the two cases.

Suppose now that we have an approximation algorithm with ratio better than r which, given an N-vertex d-regular graph runs in time $2^{(N/d)^{1-\epsilon}}$. Giving our constructed instance as input to this algorithm would allow to decide the original instance in time $2^{n^{1-\epsilon}}$. ◀

References

1 Nir Ailon and Noga Alon. Hardness of fully dense problems. *Inf. Comput.*, 205(8):1117–1129, 2007. doi:10.1016/j.ic.2007.02.006.

2 Noga Alon, Wenceslas Fernandez de la Vega, Ravi Kannan, and Marek Karpinski. Random sampling and approximation of max-csps. *J. Comput. Syst. Sci.*, 67(2):212–243, 2003. doi:10.1016/S0022-0000(03)00008-4.

3 A. Arora, A. Frieze, and H. Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. *Mathematical Programming, Series A*, 92:1–36, 2002.

4 A. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. *Journal of Computer and System Sciences*, 58:193–210, 1999.

5 Cristina Bazgan, Wenceslas Fernandez de la Vega, and Marek Karpinski. Polynomial time approximation schemes for dense instances of minimum constraint satisfaction. *Random Struct. Algorithms*, 23(1):73–91, 2003. doi:10.1002/rsa.10072.

6 Nicolas Bourgeois, Federico Della Croce, Bruno Escoffier, and Vangelis Th. Paschos. Fast algorithms for min independent dominating set. *Discrete Applied Mathematics*, 161(4-5):558–572, 2013. doi:10.1016/j.dam.2012.01.003.

7 Nicolas Bourgeois, Bruno Escoffier, and Vangelis Th. Paschos. Approximation of max independent set, min vertex cover and related problems by moderately exponential algorithms. *Discrete Applied Mathematics*, 159(17):1954–1970, 2011. doi:10.1016/j.dam.2011.07.009.

8 Jean Cardinal, Marek Karpinski, Richard Schmied, and Claus Viehmann. Approximating subdense instances of covering problems. *Electronic Notes in Discrete Mathematics*, 37:297–302, 2011. doi:10.1016/j.endm.2011.05.051.

9 Jean Cardinal, Marek Karpinski, Richard Schmied, and Claus Viehmann. Approximating vertex cover in dense hypergraphs. *J. Discrete Algorithms*, 13:67–77, 2012. doi:10.1016/j.jda.2012.01.003.

10 Parinya Chalermsook, Bundit Laekhanukit, and Danupon Nanongkai. Independent set, induced matching, and pricing: Connections and tight (subexponential time) approximation hardnesses. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 370–379. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.47.

11 Marek Cygan, Lukasz Kowalik, and Mateusz Wykurz. Exponential-time approximation of weighted set cover. *Inf. Process. Lett.*, 109(16):957–961, 2009. doi:10.1016/j.ipl.2009.05.003.

12 Marek Cygan and Marcin Pilipczuk. Exact and approximate bandwidth. *Theor. Comput. Sci.*, 411(40-42):3701–3713, 2010. doi:10.1016/j.tcs.2010.06.018.

13 Marek Cygan, Marcin Pilipczuk, and Jakub Onufry Wojtaszczyk. Capacitated domination faster than $O(2^n)$. *Inf. Process. Lett.*, 111(23-24):1099–1103, 2011. doi:10.1016/j.ipl.2011.09.004.

14 Wenceslas Fernandez de la Vega. MAX-CUT has a randomized approximation scheme in dense graphs. *Random Struct. Algorithms*, 8(3):187–198, 1996. doi:10.1002/(SICI)1098-2418(199605)8:3<187::AID-RSA3>3.0.CO;2-U.

15 Wenceslas Fernandez de la Vega and Marek Karpinski. On the approximation hardness of dense TSP and other path problems. *Inf. Process. Lett.*, 70(2):53–55, 1999. doi:10.1016/S0020-0190(99)00048-4.

16 Wenceslas Fernandez de la Vega and Marek Karpinski. Polynomial time approximation of dense weighted instances of MAX-CUT. *Random Struct. Algorithms*, 16(4):314–332, 2000. doi:10.1002/1098-2418(200007)16:4<314::AID-RSA2>3.0.CO;2-E.

17 Wenceslas Fernandez de la Vega and Marek Karpinski. Approximation complexity of nondense instances of MAX-CUT. *Electronic Colloquium on Computational Complexity (ECCC)*, 13(101), 2006. URL: http://eccc.hpi-web.de/eccc-reports/2006/TR06-101/index.html.

18 D.P. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

19 Martin E. Dyer, Alan M. Frieze, and Mark Jerrum. Approximately counting hamilton paths and cycles in dense graphs. *SIAM J. Comput.*, 27(5):1262–1272, 1998. doi:10.1137/S009753979426112X.

20 Alan M. Frieze and Ravi Kannan. The regularity lemma and approximation schemes for dense problems. In *37th Annual Symposium on Foundations of Computer Science, FOCS'96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 12–20. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548459.

21 Johan Hastad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001. doi:10.1145/502090.502098.

22 Tomokazu Imamura and Kazuo Iwama. Approximating vertex cover on dense graphs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 582–589. SIAM, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070513.

23 Sanjeev Khanna, Madhu Sudan, and David P. Williamson. A complete classification of the approximability of maximization problems derived from boolean constraint satisfaction. In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 11–20. ACM, 1997. doi:10.1145/258533.258538.

24 Dana Moshkovitz and Ran Raz. Two-query PCP with subconstant error. *J. ACM*, 57(5), 2010. doi:10.1145/1754399.1754402.

25 P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143, 1988.

26 P. Raghavan and C.D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–474, 1987.

27 Thomas J. Schaefer. The complexity of satisfiability problems. In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 216–226. ACM, 1978. doi:10.1145/800133.804350.

28 Luca Trevisan. Inapproximability of combinatorial optimization problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 2004. URL: http://eccc.hpi-web.de/eccc-reports/2004/TR04-065/index.html.

The Complexity of the Hamilton Cycle Problem in Hypergraphs of High Minimum Codegree*

Frederik Garbe[1] **and Richard Mycroft**[2]

1 **School of Mathematics, University of Birmingham, Birmingham B15 2TT, United Kingdom**
fxg472@bham.ac.uk

2 **School of Mathematics, University of Birmingham, Birmingham B15 2TT, United Kingdom**
r.mycroft@bham.ac.uk

Abstract

We consider the complexity of the Hamilton cycle decision problem when restricted to k-uniform hypergraphs H of high minimum codegree $\delta(H)$. We show that for tight Hamilton cycles this problem is NP-hard even when restricted to k-uniform hypergraphs H with $\delta(H) \geq \frac{n}{2} - C$, where n is the order of H and C is a constant which depends only on k. This answers a question raised by Karpiński, Ruciński and Szymańska. Additionally we give a polynomial-time algorithm which, for a sufficiently small constant $\varepsilon > 0$, determines whether or not a 4-uniform hypergraph H on n vertices with $\delta(H) \geq \frac{n}{2} - \varepsilon n$ contains a Hamilton 2-cycle. This demonstrates that some looser Hamilton cycles exhibit interestingly different behaviour compared to tight Hamilton cycles. A key part of the proof is a precise characterisation of all 4-uniform hypergraphs H on n vertices with $\delta(H) \geq \frac{n}{2} - \varepsilon n$ which do not contain a Hamilton 2-cycle; this may be of independent interest. As an additional corollary of this characterisation, we obtain an exact Dirac-type bound for the existence of a Hamilton 2-cycle in a large 4-uniform hypergraph.

1998 ACM Subject Classification G.2.2 Graph Theory – Hypergraphs, Graph Algorithms

Keywords and phrases Hamilton cycles, hypergraphs, graph algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.38

1 Introduction

The study of Hamilton cycles in graphs has been a topic of great significance in graph theory, and continues to be well-studied. For example, the Hamilton cycle decision problem (given a graph, determine whether it contains a Hamilton cycle) was one of Karp's celebrated 21 NP-complete problems [9], whilst one very well-known classic result is Dirac's theorem [4], which states that any graph on $n \geq 3$ vertices with minimum degree at least $\frac{n}{2}$ contains a Hamilton cycle.

The problem of generalising these results to the hypergraph setting has been a highly-active area of research over the past several years (see, for example, the recent surveys by Kühn and Osthus [15], Rödl and Ruciński [16] and Zhao [21]). To describe these developments we require the following standard definitions. A *k-uniform hypergraph*, or *k-graph* H consists of a set of vertices $V(H)$ and a set of edges $E(H)$, where each edge consists of k vertices. So a 2-graph is a (simple) graph. We say that a k-graph C is an *ℓ-cycle* if its vertices can be cyclically ordered in such a way that each edge of C consists of k consecutive vertices,

* RM supported by EPSRC Grant EP/M011771/1.

© Frederik Garbe and Richard Mycroft;
licensed under Creative Commons License CC-BY

and so that each edge intersects the subsequent edge in ℓ vertices. This naturally generalises the notion of a cycle in a graph, and is the most commonly-used definition of a hypergraph cycle. However, various other definitions have also been considered, such as a Berge cycle [1]. Note in particular that each edge of an ℓ-cycle k-graph C has $k-\ell$ vertices which were not contained in the previous edge, so the number of vertices of C must be divisible by $k-\ell$. We say that a k-graph H on n vertices contains a *Hamilton ℓ-cycle* if it contains an n-vertex ℓ-cycle as a subgraph; as before, this is only possible if $k-\ell$ divides n. We refer to $(k-1)$-cycles as *tight cycles*, and in the same way refer to *tight Hamilton cycles*. Given a k-graph H and a set $S \subseteq V(H)$, the *degree* of S, denoted $\deg_H(S)$ (or $\deg(S)$ when H is clear from the context), is the the number of edges of H which contain S as a subset. The *minimum codegree* of H, denoted $\delta(H)$, is the minimum of $\deg(S)$ taken over all sets of $k-1$ vertices of H, and the *maximum codegree* of H, denoted $\Delta(H)$, is the maximum of $\deg(S)$ taken over all sets of $k-1$ vertices of H. In the graph case the maximum and minimum codegree are simply the maximum and minimum degree respectively.

An elementary reduction from the graph case demonstrates that for any $k \geq 3$ and $1 \leq \ell \leq k$ the Hamilton ℓ-cycle decision problem (given a k-graph H, determine whether it contains a Hamilton ℓ-cycle) is also NP-complete. For this reason, many authors have asked for conditions on H which render this problem tractable, or which guarantee the existence of a Hamilton ℓ-cycle in H. In particular, since a Hamilton cycle in H cannot exist if H has an isolated vertex, it is natural to study minimum degree conditions on H.

1.1 Dirac-Type Results

The following theorem, whose various cases were proved by Rödl, Ruciński and Szemerédi [17, 18], Kühn and Osthus [14], Keevash, Kühn, Osthus and Mycroft [12], Hàn and Schacht [6], and Kühn, Osthus and Mycroft [13], is an approximate hypergraph analogue of Dirac's theorem; for any k and ℓ it gives the asymptotically best-possible minimum codegree condition which guarantees the existence of a Hamilton ℓ-cycle in a k-graph.

▶ **Theorem 1** ([6, 12, 13, 14, 17, 18]). *For any $k \geq 3$, $1 \leq \ell \leq k-1$ and $\eta > 0$, there exists n_0 such that if $n \geq n_0$ is divisible by $k-\ell$ and H is a k-graph on n vertices with*

$$\delta(H) \geq \begin{cases} \left(\frac{1}{2}+\eta\right)n & \text{if } k-\ell \text{ divides } k, \\ \left(\frac{1}{\lceil \frac{k}{k-\ell} \rceil (k-\ell)}+\eta\right)n & \text{otherwise,} \end{cases}$$

then H contains a Hamilton ℓ-cycle.

Simple constructions show that for any k and ℓ this minimum codegree condition is best possible up to the ηn error term. More recently the exact threshold (for large n) has been determined in some cases: for $k=3, \ell=2$ by Rödl, Ruciński and Szemerédi [19], for $k=3, \ell=1$ by Czygrinow and Molla [2], and for $k \geq 3$ and $\ell < k/2$ by Han and Zhao [8]. As part of our work on the question of tractability (described in more detail in the next section), we successfully characterised all 4-graphs H with $\delta(H) \geq \frac{n}{2} - \varepsilon n$ which do not contain a Hamilton cycle. As a straightforward consequence of this, we add to the aforementioned results the exact Dirac-type statement for the previously-open case $k=4, \ell=2$.

▶ **Theorem 2.** *There exists n_0 such that if $n \geq n_0$ is even and H is a 4-graph on n vertices with*

$$\delta(H) \geq \begin{cases} \frac{n}{2}-2 & \text{if } n \text{ is divisible by } 8, \\ \frac{n}{2}-1 & \text{otherwise,} \end{cases}$$

then H contains a Hamilton 2-cycle. Moreover, this condition is best-possible for any even $n \geq n_0$.

1.2 Tractability of the Restricted Hamilton Cycle Decision Problem

We now turn to the primary focus of this paper: minimum degree conditions which render the Hamilton cycle decision problem tractable. In the graph case, Dahlhaus, Hajnal and Karpiński [3] showed that for any fixed $\varepsilon > 0$ this problem remains NP-complete when restricted to graphs with minimum degree at least $(1-\varepsilon)\frac{n}{2}$. More recently, Karpiński, Ruciński and Szymańska [10] showed that for any $k \geq 3$ and any fixed $\varepsilon > 0$ the tight Hamilton cycle decision problem remains NP-complete when restricted to k-graphs with minimum codegree $(1-\varepsilon)\frac{n}{k}$. They noted that, combined with Theorem 1, this left a 'hardness gap' of $[\frac{n}{k}, \frac{n}{2}]$ for which the hardness of the problem remained unknown. We answer this question with the following theorem.

▶ **Theorem 3.** *For any $k \geq 3$ there exists C such that the tight Hamilton cycle decision problem remains NP-complete when restricted to k-graphs H with $\delta(H) \geq \frac{n}{2} - C$ (where $n = |V(H)|$).*

Assuming that P $\neq$ NP, Theorems 1 and 3 together imply that the minimum codegree threshold at which the tight Hamilton cycle decision problem becomes tractable is asymptotically equal to the minimum codegree threshold for the existence of a tight Hamilton cycle, mirroring the situation in the graph case. Interestingly, we can demonstrate that the Hamilton 2-cycle problem exhibits significantly different behaviour; our next theorem shows that there is a linear-size gap between the threshold at which the problem becomes tractable and at which the existence of a cycle is guaranteed.

▶ **Theorem 4.** *There exist a constant $\varepsilon > 0$ and an algorithm which, given a 4-graph H on n vertices with $\delta(H) \geq \frac{n}{2} - \varepsilon n$, determines in time $O(n^{25})$ whether H contains a Hamilton 2-cycle.*

A slight adaptation of the argument of Karpiński, Ruciński and Szymańska [10] mentioned above shows that for any fixed $\varepsilon > 0$ the Hamilton 2-cycle problem remains NP-complete when restricted to 4-graphs with minimum codegree at least $(1-\varepsilon)\frac{n}{4}$.

A key result in our proof of Theorem 4, which may be of independent interest, is Theorem 6, which (for sufficiently small ε and large n) precisely characterises all 4-graphs on n vertices which satisfy $\delta(H) \geq \frac{n}{2} - \varepsilon n$ but which do not contain a Hamilton 2-cycle. We prove this result using recently developed techniques of extremal graph theory, in particular the so-called 'absorbing method' of Rödl, Ruciński and Szemerédi [17]. Establishing this characterisation is the principal difficulty in the proof of Theorem 4, as then the algorithm for Theorem 4 simply checks whether this characterisation is satisfied. Likewise, Theorem 2 follows from Theorem 6 by a case analysis.

1.3 Discussion

In the light of Theorem 4, it would be very interesting to know which other values of k and ℓ also have the property that there is a linear-size gap between the minimum codegree threshold which renders the k-graph Hamilton ℓ-cycle problem tractable and the minimum codegree threshold under which the problem becomes trivial. Theorem 3 shows that this is not the case when $\ell = k-1$, whilst a slight adaptation to the arguments of Karpiński, Ruciński and Szymańska [10] demonstrates that this is also not true if $k-\ell$ does not divide k (in which case the lower degree threshold of Theorem 1 applies); all other cases remain open.

We also note that Theorem 3 demonstrates an interesting difference between the perfect matching problem and tight Hamilton cycle problem in k-graphs. Indeed, while the unrestricted versions of both problems are NP-complete, Keevash, Knox and Mycroft [11] and Han [7]

showed that the perfect matching problem can be solved in polynomial time in k-graphs H with $\delta(H) \geq n/k$; complementing a previous result of Szymanska [20], who showed that for any $\varepsilon > 0$ the problem remains NP-complete under the restriction $\delta(H) \geq (\frac{1}{k} - \varepsilon)n$. So, assuming P $\neq$ NP, for any $\frac{1}{k} \leq \alpha < \frac{1}{2}$ the two problems lie in distinct complexity classes when restricted to k-graphs with minimum codegree $\delta(H) \geq \alpha n$.

Finally, whilst the constant ε in Theorem 4 is quite small, we conjecture that Theorem 6 (the characterisation of 4-graphs H with $\delta(H) \geq \frac{n}{2} - \varepsilon n$ and no Hamilton 2-cycle) is in fact valid under the weaker condition that $\delta(H) > \frac{n}{3}$. If true, this would imply that Theorem 4 would also hold under this weaker codegree assumption.

1.4 Notation

Given a set V, we write $\binom{V}{k}$ for the set of subsets of V of size k. Also, we write $x \ll y$ ("x is sufficiently smaller than y") to mean that for any $y > 0$ there exists $x_0 > 0$ such that for any $x \leq x_0$ the subsequent statement holds. Similar statements with more variables are defined accordingly.

2 Hamilton 2-Cycles

In this section we outline the proof of Theorem 4. The key to the proof is Theorem 6, which precisely characterises all large 4-graphs H with $\delta(H) \geq (\frac{1}{2} - \varepsilon)n$ which do not contain a Hamilton 2-cycle. This is presented in Section 2.1. Having established this characterisation, it is fairly straightforward to exhibit a polynomial-time algorithm which tests whether a 4-graph has this property, as shown in Section 2.2. Instead, the difficult part of the proof is to prove Theorem 6; we outline how this is done in Section 2.3. Finally, in Section 2.4 we present the short deduction of Theorem 2 from Theorem 6.

2.1 A Characterisation of Dense 4-graphs with no Hamilton 2-Cycle

For 4-graphs H, our characterisation considers partitions of $V(H)$ into two parts A and B. Whenever we refer to, for example, 'a partition (A, B) of $V(H)$', this should be interpreted as meaning a partition of $V(H)$ into two non-empty parts A and B. Given such a partition of $V(H)$, we say that an edge $e \in E(H)$ is *odd* if $|e \cap A|$ is odd, and *even* if $|e \cap A|$ is even. We write H_{even} for the subgraph of H consisting only of even edges of H, and similarly write H_{odd} for the subgraph of H consisting only of odd edges of H. Also, we say that a pair $\{x, y\}$ of distinct vertices of H is a *split* pair if $x \in A$ and $y \in B$ or vice versa, and that $\{x, y\}$ is an *equal* pair if $x, y \in A$ or $x, y \in B$.

We define an ℓ-path in a k-graph analogously to an ℓ-cycle: a k-graph is an *ℓ-path* if its vertices can be linearly ordered $v_1, \ldots, v_n$ such that every edge consists of k consecutive vertices and successive edges intersect in precisely ℓ vertices. As for cycles we refer to $(k-1)$-paths as *tight paths*. The *length* of an ℓ-path is the number of edges. Given a 4-graph H, we define the *total 2-pathlength* of H to be the maximum sum of lengths of vertex-disjoint 2-paths in H. For example, H having total 2-pathlength 3 could be achieved by 3 disjoint edges (i.e. 2-paths of length 1) in H, or a 2-path of length 3 in H, or two vertex-disjoint 2-paths in H, one of length 1 and one of length 2. Using these definitions we can now give the central definition of our characterisation.

▶ **Definition 5.** Let H be a 4-graph on n vertices, where n is even. We say that a partition (A, B) of $V(H)$ is *even-good* if at least one of the following statements holds:

(i) $|A|$ is even or $|A| = |B|$.
(ii) H contains odd edges e and e' such that either $e \cap e' = \emptyset$ or $e \cap e'$ is a split pair.
(iii) $|A| = |B| + 2$ and H contains odd edges e and e' with $e \cap e' \in \binom{A}{2}$.
(iv) $|B| = |A| + 2$ and H contains odd edges e and e' with $e \cap e' \in \binom{B}{2}$.

Now let $m \in \{0, 2, 4, 6\}$ and $d \in \{0, 2\}$ be such that $m \equiv n \mod 8$ and $d \equiv |A| - |B| \mod 4$. Then we say that (A, B) is *odd-good* if at least one of the following statements holds.

(v) $(m, d) \in \{(0, 0), (4, 2)\}$.
(vi) $(m, d) \in \{(2, 2), (6, 0)\}$ and H contains an even edge.
(vii) $(m, d) \in \{(4, 0), (0, 2)\}$ and H_{even} has total 2-pathlength at least two.
(viii) $(m, d) \in \{(6, 2), (2, 0)\}$ and either there is an edge $e \in E(H)$ with $|e \cap A| = |e \cap B| = 2$ or H_{even} has total 2-pathlength at least three.

A key observation is that if (A, B) is a partition of $V(H)$ which is not even-good, then there exists a set X of at most four vertices of H such that every odd edge of H intersects X. Indeed, if H contains an odd edge e, then we may take $X = e$, and otherwise we may take $X = \emptyset$. Similarly, by choosing X to be the vertices of at most two disjoint even edges, or of a 2-path of length two in H_{even}, we find that if (A, B) is a partition of $V(H)$ which is not odd-good, then there exists a set X of at most 8 vertices of H such that every even edge of H intersects X.

We now give our characterisation of 4-graphs of high minimum codegree with no Hamilton 2-cycle. Recall for this that any 2-cycle 4-graph has an even number of vertices.

▶ **Theorem 6.** *There exist $\varepsilon, n_0 > 0$ such that the following statement holds for any even $n \geq n_0$. Let H be a 4-graph on n vertices with $\delta(H) \geq (\frac{1}{2} - \varepsilon)n$. Then H admits a Hamilton 2-cycle if and only if every partition (A, B) of $V(H)$ is both even-good and odd-good.*

2.2 The Algorithm

Our polynomial-time algorithm for determining the existence of a Hamilton 2-cycle in a 4-graph of high codegree makes use of a special case of a result of Keevash, Knox and Mycroft [11]. This result allows us to efficiently list all partitions (A, B) of $V(H)$ with no odd edges, or all partitions with no even edges.

▶ **Lemma 7** ([11], special case of Lemma 2.2)**.** *Let H be a 4-graph on n vertices with $\delta(H) > \frac{n}{3}$, and let $x \in \{\text{even}, \text{odd}\}$. Then there are at most 64 partitions (A, B) of $V(H)$ for which no edge of H has parity x with respect to (A, B). Moreover, there exists an algorithm* ListPartitions(H, x) *with running time $O(n^5)$ which, given H and x, returns all such partitions.*

We now present an algorithm, Procedure GoodPartition(H, x), which determines whether or not there exists an even-good/odd-good partition (A, B) for a 4-graph H. Note that, given a 4-graph H and a partition (A, B) of $V(H)$, the truth of the statements '(A, B) is odd-good' and '(A, B) is even-good' depend only on the values of n and $|A|$ and whether or not H_{odd} or H_{even} contain certain subgraphs with at most 12 vertices. It follows that the validity of these statements (and therefore the condition of the 'if' statement in Procedure GoodPartition) can be tested in time $O(n^{12})$.

▶ **Proposition 8.** *Let H be a 4-graph on n vertices with $\delta(H) > \frac{n}{3}$, where n is even, and fix a parity $x \in \{\text{even}, \text{odd}\}$. Then Procedure GoodPartition(H, x) will correctly determine whether there exists a partition (A, B) of $V(H)$ which is not x-good, with running time $O(n^{25})$.*

Procedure GoodPartition(H, x)

Data: A 4-graph H with vertex set V and a parity $x \in \{\text{even}, \text{odd}\}$.
Result: Determines if there is a partition (A, B) of V which is not x-good.

```
for each set X ⊆ V(H) with |X| = 8 do
    Let V' = V \ X and H' = H[V'].
    Run Procedure ListPartitions(H', x) to obtain all partitions (A', B') of V' with no
     edges not of parity x.
    for each such partition (A', B') do
        for each partition (A, B) of V with A' ⊆ A and B' ⊆ B do
            if (A, B) is not x-good then
                State '(A, B) is not x-good', and terminate.
State 'Every partition is x-good', and terminate.
```

Proof. We first establish correctness of the algorithm; for this, fix H and x as in the proposition statement. Clearly, if every partition (A, B) of $V := V(H)$ is x-good, then GoodPartition(H, x) will output this fact. So suppose that some partition (A, B) of V is not x-good. As noted following Definition 5, we may then choose a set X of at most 8 vertices of H which is intersected by every edge of H which does not have parity x. This means that when GoodPartition(H, x) considers this set X, ListPartitions will return the partition (A', B') where $A' = A \setminus X$ and $B' = B \setminus X$, and at this point GoodPartition(H, even) will return that (A, B) is not x-good, as required.

Finally we consider the running time of the algorithm. For this note that there are $\binom{n}{8}$ choices for X in the outside 'for loop', and for each of these Procedure ListPartitions(H', x) runs in time $O(n^5)$. The inside 'for loops' then range over sets of size at most 64 (by Lemma 7) and $2^8 = 256$ respectively. Finally, as noted above we may test whether a partition (A, B) is x-good in time $O(n^{12})$; together these bounds combine to give the claimed running time. ◀

Proof of Theorem 4. Let n_0 be sufficiently large and $\varepsilon > 0$ sufficiently small for Theorem 6 to apply. Given a 4-graph H on n vertices with $\delta(H) \geq (\frac{1}{2} - \varepsilon)n$, we apply the following algorithm. Firstly, if n is odd, then there can be no Hamilton 2-cycle in H, so we output this fact and terminate. Secondly, if $n < n_0$, then we use a brute-force approach, testing each of the at most $n_0!$ orderings of $V(H)$ in turn to determine whether it yields a Hamilton 2-cycle in H. We then output the appropriate answer and terminate. Finally, if $n \geq n_0$ is even, then we first run Procedure GoodPartition(H, even), and then run Procedure GoodPartition(H, odd). If either of these procedures yields a partition (A, B) of $V(H)$ which is not even-good or which is not odd-good, then we return that there is no Hamilton 2-cycle in H, otherwise we return that there is such a cycle. Note that in the first two cases this algorithm runs in constant time, whilst in the final case it runs in time $O(n^{25})$ by Proposition 8. Moreover Theorem 6 ensures that this algorithm will always output the correct answer. ◀

2.3 Proof of Theorem 6

We begin by establishing the forward implication of Theorem 6, expressed in the following proposition. In fact, the minimum codegree condition on H is not required for this direction.

▶ **Proposition 9.** *If H is a 4-graph which contains a Hamilton 2-cycle, then every partition (A, B) of $V(H)$ is both even-good and odd-good.*

Proof. Let n be the order of H, let $C = (v_1, v_2, \dots, v_n)$ be a Hamilton 2-cycle in H and let (A, B) be a partition of $V(H)$. Write $P_i = \{v_{2i-1}, v_{2i}\}$ for each $1 \leq i \leq \frac{n}{2}$, so the edges of C are $e_i := P_i \cup P_{i+1}$ for $1 \leq i \leq \frac{n}{2}$ (with addition taken modulo $\frac{n}{2}$). The key observation is that e_i is even if P_i and P_{i+1} are both split pairs or both equal pairs, and odd otherwise.

We first show that (A, B) is even-good. This holds by (ii) if H contains two disjoint odd edges, so we may assume without loss of generality that all edges of H other than e_1 and $e_{n/2}$ are even. It follows that the pairs $P_2, P_3, \dots, P_{n/2}$ are either all split pairs or all equal pairs. In the former case, if P_1 is a split pair then $|A| = |B|$, so (i) holds, whilst if $P_1 \subseteq A$ then (iii) holds, and if $P_1 \subseteq B$ then (iv) holds. In the latter case, if P_1 is an equal pair then $|A|$ is even, so (i) holds, whilst if P_1 is a split pair then (ii) holds. So in all cases we find that (A, B) is even-good.

To show that (A, B) is odd-good, suppose first that 4 does not divide n, and note that by our key observation the number of even edges in C must then be odd. If C contains three or more even edges or an edge with precisely two vertices in A, then (A, B) is odd-good by (vi) and (viii), so we may assume without loss of generality that $e_{n/2}$ is the unique even edge in C and that $e_{n/2} \subseteq A$ or $e_{n/2} \subseteq B$. It follows that $P_1, P_3, \dots, P_{n/2}$ are equal pairs and the remaining pairs are split, so $|A| - |B| \equiv 2\lceil \frac{n}{4} \rceil \mod 4$. We must therefore have $(m, d) \in \{(2, 2), (6, 0)\}$, and (A, B) is odd-good by (vi). On the other hand, if 4 divides n, then by our key observation the number of even edges in C is even. If this number is at least two then (A, B) is odd-good by (v) and (vii). If instead every edge of C is odd, then exactly $\frac{n}{4}$ of the pairs P_i are equal pairs, so $|A| - |B| \equiv \frac{n}{2} \mod 4$, and C is odd-good by (v). ◀

To prove Theorem 6 it therefore suffices to prove the backwards implication. Our approach for this is motivated by the observation that if H is a 4-graph and (A, B) is a partition of $V(H)$ which is not odd-good, then H must have very few even edges. Likewise, if (A, B) is not even-good, then H has very few odd edges. We therefore consider three cases for H: two 'near-extremal' cases, in which $V(H)$ admits a partition (A, B) with few even edges or with few odd edges, and a 'non-extremal' case, in which there is no such partition. In the 'non-extremal case' we proceed by the so-called 'absorbing' method, introduced by Rödl, Ruciński and Szemerédi [19], in which we rely heavily on the fact that H is not 'near-extremal'. On the other hand, in the 'near-extremal' cases we have significant information about the structure of H (specifically that there is a partition of $V(H)$ with few even/odd edges). Making essential use of this structural information, we proceed by *ad hoc* methods to construct a Hamilton 2-cycle in H.

The following definition formalises our two notions of 'near-extremal'.

▶ **Definition 10.** Let $c_1, c_2 > 0$ and let H be a 4-graph on n vertices.

(a) We say that H is c_1-*even-extremal* if there exists a partition (A, B) of $V(H)$ such that $(\frac{1}{2} - c_1)n \leq |A| \leq (\frac{1}{2} + c_1)n$ and H contains at most $c_1\binom{n}{4}$ odd edges.

(b) We say that H is c_2-*odd-extremal*, if there exists a partition (A, B) of $V(H)$ such that $(\frac{1}{2} - c_2)n \leq |A| \leq (\frac{1}{2} + c_2)n$ and H contains at most $c_2\binom{n}{4}$ even edges.

2.3.1 Non-Extremal 4-Graphs

As described above, in the case when H is not near-extremal, we proceed by the 'absorbing' method of Rödl, Ruciński and Szemerédi [19]. To do this we establish three key lemmas. The first of these is a 'connecting lemma', which shows that since H is not even-extremal, we can find a constant-length 2-path connecting any two disjoint pairs of vertices. For this, we say that the *ends* of a 2-path 4-graph $(v_1, \dots, v_n)$ are the pairs $\{v_1, v_2\}$ and $\{v_{n-1}, v_n\}$.

▶ **Lemma 11** (Connecting lemma). *Suppose that $\frac{1}{n} \ll \varepsilon \ll c$ and that H is a 4-graph on n vertices with $\delta(H) \geq (\frac{1}{2} - \varepsilon)n$ which is not c-even-extremal. Then for every two disjoint pairs $\{a_1, a_2\}, \{b_1, b_2\} \in \binom{V}{2}$ there is a 2-path of length at most 3 whose ends are $\{a_1, a_2\}$ and $\{b_1, b_2\}$.*

Loosely speaking, our proof of Lemma 11 supposes that we have pairs $\{a_1, a_2\}$ and $\{b_1, b_2\}$ for which no such 2-path exists. It follows that there is no pair $\{x, y\} \in \binom{V(H)}{2}$ for which $\{a_1, a_2, x, y\}$ and $\{b_1, b_2, x, y\}$ are both edges of H. Combined with the minimum codegree condition of H this yields significant structural information on H, which we use to deduce that H must be c-even-extremal and so prove the lemma.

The second key lemma is an 'absorbing lemma', which shows that since H is neither even-extremal nor odd-extremal, we can find a short 2-path in H which can 'absorb' most small collections of pairs of H.

▶ **Lemma 12** (Absorbing lemma). *Suppose that $\frac{1}{n} \ll \varepsilon \ll \rho \ll \beta \ll \lambda \ll c, \mu$. Let H be a 4-graph on n vertices with $\delta(H) \geq \frac{n}{2} - \varepsilon n$ which is neither c-even-extremal nor c-odd-extremal. Then there is a 2-path P in H and a graph G on $V(H)$ with the following properties.*

- **(i)** *P has at most μn vertices.*
- **(ii)** *Every vertex of $V(H) \setminus V(P)$ lies in at least $(1 - \lambda)n$ edges of G.*
- **(iii)** *For any $q \leq \rho n$ and any q disjoint edges $e_1, \dots, e_q$ of G which do not intersect P there is a 2-path P^* in H with the same ends as P such that $V(P^*) = V(P) \cup \bigcup_{j=1}^{q} e_j$.*

Loosely speaking, to prove Lemma 12, we first show that provided H is not c-odd-extremal, for almost every pair $\{x, y\} \in \binom{V}{2}$ there are many 2-paths Q of length 3 which can 'absorb' $\{x, y\}$, in the sense that there is a 2-path Q^* with vertex set $V(Q) \cup \{x, y\}$ and with the same ends as Q. We take G to be the graph of such pairs. We then randomly select a linear number of 2-paths of length 3 and use Lemma 11 to connect these 2-paths into a single short 2-path P (this is where we require that H is not c-even-extremal). Next we extend P to include the small number of vertices which lie in fewer than $(1 - \lambda)n$ edges of G, so that (ii) holds. Finally, we show that given any set of edges $e_1, \dots, e_q$ of G as in (iii), we can match these edges to the randomly chosen paths Q, and absorb each edge into the corresponding path to obtain P^*.

Our final key lemma is a 'path cover lemma', which states that we can cover almost all vertices of H by a constant number of vertex-disjoint 2-paths. In fact, we do not actually need the requirement that H is not near-extremal, and can simply cite a result of Kühn, Mycroft and Osthus [13].

▶ **Lemma 13** (Path cover lemma [13]). *Suppose that $\frac{1}{n} \ll \frac{1}{D} \ll \gamma \ll \eta$ and that H is a 4-graph on n vertices with $\delta(H) \geq (\frac{1}{4} + \eta)n$. Then H contains a set of at most D vertex-disjoint 2-paths covering all but at most γn vertices of H.*

For non-extremal 4-graphs H, combining these three lemmas proves the reverse implication of Theorem 6, which we express in the following lemma.

▶ **Lemma 14.** *Suppose that $\frac{1}{n} \ll \varepsilon \ll c$ and that n is even, and let H be a 4-graph of order n with $\delta(H) \geq (\frac{1}{2} - \varepsilon)n$. If H is neither c-odd-extremal nor c-even-extremal, then H contains a Hamilton 2-cycle.*

Proof sketch. Introduce constants with $1/n \ll 1/D, \varepsilon \ll \gamma \ll \rho \ll \beta \ll \lambda \ll c, \mu \ll 1$, and apply Lemma 12 to obtain an absorbing 2-path P_0 in H and a graph G on $V(H)$ with the stated properties. Let $V := V(H)$ and $U := V(P_0)$, and now choose uniformly at random a set $R \subseteq V \setminus U$ of size ρn. Next, apply Lemma 13 (with, say, $\eta = 1/10$) to obtain at most D

vertex-disjoint 2-paths $P_1, \dots, P_q$ in $H[V \setminus (U \cup R)]$ covering all but at most γn vertices. By q applications of Lemma 11 we can find vertex-disjoint 2-paths $Q_0, Q_1, \dots, Q_q$, each of length at most 3, such that Q_0 connects the end of P_0 to the start of P_1, Q_1 connects the end of P_1 to the start of P_2, and so forth, with Q_q connecting the end of P_q to the start of P_0. Moreover, all vertices of Q_i except those in the end of P_i or the start of P_{i+1} should be taken from R. (The random choice of R ensures that the conditions of Lemma 11 are satisfied for each application.) This yields a 2-cycle $C = P_0Q_0P_1Q_1P_2 \dots P_qQ_q$ in H covering all vertices except the at most γn vertices not covered by $P_1, \dots, P_q$ and between $\rho n - 3D$ and ρn unused vertices of R. So $X := V \setminus V(C)$ has size $\rho n - 3D \leq |X| \leq \rho n + \gamma n$. Furthermore, $|X|$ is even since n and $|V(C)|$ are both even, and our random choice of R ensures that every vertex $x \in X$ has $\deg_{G[X]}(x) \geq |X|/2$. So there is a perfect matching $e_1, \dots, e_{|X|/2}$ in $G[X]$; since $|X|/2 \leq \rho n$ we may 'absorb' X into P_0 to obtain a 2-path P^*. Replacing P_0 by P^* in C gives a Hamilton 2-cycle in H. ◀

2.3.2 Extremal 4-Graphs

Having dealt with the 'non-extremal' case, it remains to deal with the two 'near-extremal' cases by proving the following two lemmas via an extremal case.

▶ **Lemma 15.** *Suppose that $\frac{1}{n} \ll \varepsilon, c \ll 1$ and that n is even, and let H be a 4-graph of order n with $\delta(H) \geq (\frac{1}{2} - \varepsilon)n$. If H is c-even-extremal and every partition of $V(H)$ into two parts A and B is even-good, then H contains a Hamilton 2-cycle.*

▶ **Lemma 16.** *Suppose that $\frac{1}{n} \ll \varepsilon, c \ll 1$ and that n is even, and let H be a 4-graph of order n with $\delta(H) \geq (\frac{1}{2} - \varepsilon)n$. If H is c-odd-extremal and every partition of $V(H)$ into two parts A and B is odd-good, then H contains a Hamilton 2-cycle.*

Proof sketch. As is typical of this type of argument, each lemma is proved by a long and detailed extremal case analysis, and so we limit ourselves here to a brief outline of the argument for Lemma 15 (the outline for Lemma 16 is similar with 'even' and 'odd' reversed). Let (A', B') be a partition of $V(H)$ witnessing that H is c-even-extremal. We first observe that the bound on $\delta(H)$ implies that H has density at least $(\frac{1}{2} - \varepsilon)$. Combined with the fact that H has few odd edges, this implies that almost every set $S \subseteq V(H)$ for which $|A' \cap S|$ is even is an edge of H. However, it is possible that a small number of vertices may lie in very few even edges, so we begin by 'tidying up' the partition: we move a few vertices of H from one side to the other to ensure that, for instance, every vertex of H lies in many even edges. Let (A, B) be the tidied partition. By assumption this partition (A, B) is even-good, and this fact yields some structure in H with respect to this partition (precisely what structure depends on the values of n and $|A|$). For example, we might obtain two disjoint odd edges in H. We then form a short 2-path P from the given structure to satisfy the desired parity conditions, and then (using even edges only) extend P to a Hamilton 2-cycle in H. ◀

Proof of Theorem 6. Fix a constant c small enough for Lemmas 15 and 16. Having done so, choose ε sufficiently small for us to apply Lemma 14 with this choice of c, and n_0 sufficiently large that we may apply Lemmas 14, 15 and 16 with these choices of c and ε and any even $n \geq n_0$. Let H be a 4-graph on n vertices with $\delta(H) \geq (\frac{1}{2} - \varepsilon)n$, and suppose that every partition (A, B) of $V(H)$ is both even-good and odd-good. If H is either c-even-extremal or c-odd-extremal then H contains a Hamilton 2-cycle by Lemma 15 or 16 respectively. On the other hand, if H is neither c-odd-extremal nor c-even-extremal then H contains a Hamilton 2-cycle by Lemma 14. This completes the proof of the backwards implication of Theorem 6; the proof of the forwards implication was Proposition 9. ◀

2.4 Proof of Theorem 2

To conclude this section, we show how Theorem 2 can be deduced from Theorem 6. We begin by justifying the claim that the degree bound of Theorem 2 is best-possible. To see this, fix an even integer $n \geq 6$, and construct a 4-graph H^* as follows. Let A and B be disjoint sets with $|A \cup B| = n$ such that $|A| = \frac{n}{2} - 1$ if 8 divides n and $|A| = \frac{n}{2}$ otherwise. Then the vertex set of H^* is $A \cup B$, and the edges of H^* are all sets $e \in \binom{A \cup B}{4}$ such that $|e \cap A|$ is odd. Then it is easily checked that $\delta(H^*) = \frac{n}{2} - 3$ if 8 divides n and $\frac{n}{2} - 2$ otherwise. Moreover, since H^* has no even edges, our choice of size of A implies that the partition (A, B) of $V(H^*)$ is not odd-good. By Theorem 6 we conclude that there is no Hamilton 2-cycle in H^*.

Proof of Theorem 2. Choose ε, n_0 as in Theorem 6. Let $n \geq n_0$ be even and large enough that $\frac{n}{2} - 2 \geq (\frac{1}{2} - \varepsilon)n$, and let H be a 4-graph on n vertices which satisfies the minimum codegree condition of Theorem 2. Also let (A, B) be a partition of $V(H)$, and assume without loss of generality that $|A| \leq \frac{n}{2}$. By Theorem 6 it suffices to prove that (A, B) is even-good and odd-good. For this, note that if 8 divides n and $|A| = \frac{n}{2}$ then (A, B) is even-good by (i) and odd-good by (v). So we may assume that if 8 divides n then $|A| \leq \frac{n}{2} - 1$ and $\delta(H) \geq \frac{n}{2} - 2$, whilst otherwise we have $|A| \leq \frac{n}{2}$ and $\delta(H) \geq \frac{n}{2} - 1$. Either way, we must have $\delta(H) \geq |A| - 1$. Also, for any distinct $x, y, z \in V(H)$, let $N_B(x, y, z)$ denote the set of vertices $w \in B$ such that $\{x, y, z, w\} \in E(H)$.

To see that (A, B) must be even-good, arbitrarily choose vertices $x_1, x_2, y_1, y_2, z_1, z_2 \in A$. Then $|N_B(x_1, y_1, z_1)|, |N_B(x_2, y_2, z_2)| \geq \delta(H) - (|A| - 3) \geq 2$, so we may choose distinct $w_1, w_2 \in B$ with $w_1 \in N_B(x_1, y_1, z_1)$ and $w_2 \in N_B(x_2, y_2, z_2)$. The sets $\{x_1, y_1, z_1, w_1\}$ and $\{x_2, y_2, z_2, w_2\}$ are then disjoint odd edges of H, so (A, B) is even-good by (ii).

We next show that that (A, B) is also odd-good. For this, arbitrarily choose distinct vertices $a_1, a_2, \ldots, a_9, a'_1, \ldots, a'_9 \in A$ and $b_1, \ldots, b_9 \in B$. For any $1 \leq i, j \leq 9$ we have $|N_B(a_i, a'_i, b_j)| \geq \delta(H) - (|A| - 2) \geq 1$, so there must be $b^i_j \in B$ such that $\{a_i, a'_i, b_j, b^i_j\}$ is an (even) edge of H. If for each $1 \leq j \leq 9$ the vertices b^i_j for $1 \leq i \leq 9$ are all distinct, then there is no set $X \subseteq V(H)$ with $|X| \leq 8$ which intersects every even edge of H. However, as observed immediately after Definition 5, such a set X must exist if (A, B) is not odd-good. We may therefore assume that $b^{i'}_j = b^i_j$ for some $1 \leq i, i', j \leq 9$ with $i \neq i'$. It follows that $\{a_i, a'_i, b_j, b^i_j\}$ is an even edge of H with exactly two vertices in A, whilst $(a_i, a'_i, b_j, b^i_j, a_{i'}, a'_{i'})$ is a 2-path of length 2 in H_{even}. So (A, B) is odd-good by (v), (vi), (vii) or (viii), according to the value of n modulo 8. ◀

3 Tight Hamilton Cycles

Our aim in this section is to explain the principal ideas of the proof of Theorem 3, which proceeds by a series of reductions. We begin with a full proof of the case $k = 3$, in which case we proceed from a theorem of Garey, Johnson and Stockmeyer [5], who proved that the Hamilton cycle problem remains NP-complete when restricted to subcubic graphs (we say that a graph G is *subcubic* if G has maximum degree $\Delta(G) \leq 3$). The following proposition is an immediate corollary of that theorem.

▶ **Proposition 17** ([5]). *The problem of determining whether a subcubic graph admits a Hamilton path is NP-complete.*

The next lemma is the $k = 3$ case of Theorem 3, which holds with $C = 9$.

▶ **Lemma 18.** *The 3-graph tight Hamilton cycle decision problem is NP-complete even when restricted to 3-graphs H on m vertices with $\delta(H) \geq \frac{m}{2} - 9$.*

Proof. Let G be a subcubic graph on n vertices, and write $X := V(G)$. Assume for simplicity that n is even (a very similar argument handles the case where n is odd). Fix disjoint sets A and B with $|A| = \frac{3n}{2}$ and $|B| = \frac{3n}{2} + 1$ such that $X \subseteq A$, and define a 3-graph H with vertex set $A \cup B$ whose edges are

- **(i)** all sets $e \in \binom{A\cup B}{3}$ with $|A \cap e| \leq 1$,
- **(ii)** all sets $e \in \binom{A\cup B}{3}$ with $|A \cap e| = 2$ and $A \cap e \in E(G)$ (note in particular that this requires that $A \cap e \subseteq X$), and
- **(iii)** all sets $e \in \binom{A}{3}$ for which no $e' \in E(G)$ satisfies $e' \subseteq e$.

Observe first that H has $m := 3n + 1$ vertices and minimum codegree $\delta(H) \geq \frac{m}{2} - 9$. To see this, let x and y be distinct vertices of H. If either $x \in B$ or $y \in B$ then $\{x, y, z\}$ is an edge of H for any $z \in B \setminus \{x, y\}$, so $\deg_H(\{x, y\}) \geq |B| - 2 = \frac{3n}{2} - 1$. Exactly the same applies if $x, y \in A$ and $xy \in E(G)$. Finally, if $x, y \in A$ and $xy \notin E(G)$, then $\{x, y, z\}$ is an edge of H for any $z \in A \setminus \{x, y\}$ except for those z such that $xz \in E(G)$ or $yz \in E(G)$. So $\deg_H(\{x, y\}) \geq |A| - 2 - \deg_G(x) - \deg_G(y)$; since G is subcubic this gives $\deg_H(\{x, y\}) \geq \frac{3n}{2} - 8 \geq \frac{m}{2} - 9$, as claimed.

We claim that H contains a tight Hamilton cycle if and only if G contains a Hamilton path. To see this, first suppose that G contains a Hamilton path $(x_1, \cdots, x_n)$. Enumerate the vertices of $A \setminus X$ and B as $a_1, a_2, \ldots, a_{n/2}$ and $b_1, b_2, \ldots, b_{3n/2+1}$ respectively. Then

$$\left(x_1, x_2, b_1, x_3, x_4, b_2, \cdots, x_{n-1}, x_n, b_{\frac{n}{2}}, b_{\frac{n}{2}+1}, a_1, b_{\frac{n}{2}+2}, b_{\frac{n}{2}+3}, a_2, \cdots, a_{\frac{n}{2}}, b_{\frac{3n}{2}}, b_{\frac{3n}{2}+1}\right)$$

is a tight Hamilton cycle in H.

Now suppose instead that H contains a tight Hamilton cycle C. Note that our construction of H ensures that there are no edges $e, e' \in E(H)$ with $|e \cap A| = 3$, $|e' \cap A| = 2$ and $|e \cap e'| = 2$. Since every edge of C intersects the subsequent edge of C in precisely two vertices, and $B \neq \emptyset$, it follows that C cannot contain any edge e with $|e \cap A| = 3$. So there are at least $\frac{n}{2}$ vertices $a \in X$ which are succeeded in C by a vertex of B. Now let A_1 be the set of vertices of X for which the subsequent vertex of A on C is in X and A_2 be the set of vertices of X for which the subsequent vertex of A on C is in $A \setminus X$. Also let $A_3 := A \setminus X$, so A is the disjoint union of A_1, A_2 and A_3. By construction of H, any vertex of $A \setminus X$ must be preceded in C by two vertices of B and succeeded in C by two vertices of B; it follows that any vertex of $A_2 \cup A_3$ is succeeded in C by two vertices of B, and so we obtain

$$|B| \geq (\tfrac{n}{2} - |A_2|) + 2(|A_2| + |A_3|) = \tfrac{n}{2} + |A_2| + 2|A_3| = \tfrac{3n}{2} + |A_2|.$$

Since $A \setminus X$ is non-empty, we must have $|A_2| \geq 1$. Combined with the fact that $|B| = \frac{3n}{2} + 1$ this implies that $|A_2| = 1$, and all inequalities are in fact equalities. So precisely one vertex of X is succeeded in C by two vertices of B, $\frac{n}{2} - 1$ vertices of X are succeeded by one vertex of B, and the remaining $\frac{n}{2}$ vertices of X are succeeded by a vertex of A (which must therefore be in X). This implies that C contains a tight Hamilton path of the form $(x_1, x_2, b_1, x_3, x_4, b_2, \ldots, b_{n/2-1}, x_{n-1}, x_n)$, where $X = \{x_1, \ldots, x_n\}$ and $b_i \in B$ for $1 \leq i \leq \frac{n}{2} - 1$. By our construction of H it follows that $(x_1, x_2, \ldots, x_n)$ is a Hamilton path in G.

Altogether, this shows that any instance of the Hamilton cycle problem for subcubic graphs can be reduced to a single instance of the problem of finding a tight Hamilton cycle in a 3-graph on m vertices with $\delta(H) \geq \frac{m}{2} - 9$, where $m = 3n + 1$. Together with Proposition 17, this proves the lemma. ◀

We conclude by outlining the steps we use to prove Theorem 3 in full generality, using the following notation. For a function $f(n)$, we write $\mathrm{HC}(k, f(n))$ (respectively $\mathrm{HP}(k, f(n))$)

to denote the k-graph tight Hamilton cycle (respectively Hamilton path) decision problem restricted to k-graphs H on n vertices with minimum codegree $\delta(H) \geq f(n)$. On the other hand, for an integer D, we write $\overline{\mathrm{HC}}(k, D)$ (respectively $\overline{\mathrm{HP}}(k, D)$) to denote the k-graph tight Hamilton cycle (respectively Hamilton path) decision problem restricted to k-graphs H with maximum codegree $\delta(H) \leq D$. So, for example, Proposition 17 states that $\overline{\mathrm{HP}}(2, 3)$ is NP-complete, whilst Lemma 18 states that $\mathrm{HC}(3, \frac{n}{2} - 9)$ is NP-complete. We prove Theorem 3 by exhibiting the following polynomial-time reductions.

- **(i)** For any $k \geq 2$ and D we give polynomial-time reductions from $\overline{\mathrm{HC}}(k, D)$ to $\overline{\mathrm{HP}}(k, D)$ and from $\overline{\mathrm{HP}}(k, D)$ and $\overline{\mathrm{HC}}(k, D)$. These reductions are elementary and permit us the convenience of treating the tight Hamilton cycle and tight Hamilton path problems in graphs of low maximum codegree as being interchangeable.
- **(ii)** For any $k \geq 2$ we give polynomial-time reductions from $\overline{\mathrm{HC}}(k, D)$ to $\overline{\mathrm{HC}}(2k-1, 2D)$ and from $\overline{\mathrm{HC}}(k, D)$ to $\overline{\mathrm{HC}}(2k, D)$. In each case, given a k-graph H on a vertex set V, we take copies H_1 and H_2 of H with disjoint vertex sets V_1 and V_2. For the former reduction we define a $(2k-1)$-graph H^* on $V_1 \cup V_2$ whose edges are those $(2k-1)$-tuples which consist of an edge e_1 from H_1 and the copies in H_2 of $k-1$ vertices of e_1, or the same with the roles of H_1 and H_2 reversed. Likewise, for the latter reduction we define a $2k$-graph H^* on $V_1 \cup V_2$ whose edges are those $2k$-tuples $e_1 \cup e_2$ where e_1 is an edge of H_1, e_2 is an edge of H_2, and e_2 contains the copies of at least $k-1$ vertices of e_1. In either case it is not too hard to show that H^* contains a tight Hamilton cycle if and only if H does, and that $\Delta(H^*) \leq 2\Delta(H)$ in one case and $\Delta(H^*) \leq \Delta(H)$ in the other.
- **(iii)** Finally, for any $k \geq 2$ we present a polynomial-time reduction from $\overline{\mathrm{HP}}(k, D)$ to $\mathrm{HC}(2k-1, \lfloor \frac{n}{2} \rfloor - k(D+1))$ and from $\overline{\mathrm{HC}}(k, D)$ to $\mathrm{HC}(2k, \frac{n}{2} - k(D+1))$. These are similar to the reduction given in the proof of Lemma 18, except that G is now a k-graph with $\Delta(G) \leq D$, and H is a $(2k-1)$-graph or $2k$-graph (according to which reduction we are presenting).

By induction on k, with Proposition 17 as the base case, the reductions of (i) and (ii) combine to prove the following theorem, which can be seen as a generalisation to k-graphs of the aforementioned theorem of Garey, Johnson and Stockmeyer.

▶ **Theorem 19.** *For every $k \geq 2$ there exists D such that $\overline{\mathrm{HC}}(k, D)$ and $\overline{\mathrm{HP}}(k, D)$ are NP-complete.*

Theorem 3 follows immediately from Theorem 19 and the reductions of (iii).

References

1. J.-C. Bermond, A. Germa, M.-C. Heydemann, and D. Sotteau. Hypergraphes hamiltoniens. In *Problèmes combinatoires et théorie des graphes (Colloq. Internat. CNRS, Univ. Orsay, Orsay, 1976)*, volume 260 of *Colloq. Internat. CNRS*, pages 39–43. CNRS, Paris, 1978.
2. Andrzej Czygrinow and Theodore Molla. Tight codegree condition for the existence of loose Hamilton cycles in 3-graphs. *SIAM Journal on Discrete Mathematics*, 28(1):67–76, 2014. doi:10.1137/120890417.
3. Elias Dahlhaus, Peter Hajnal, and Marek Karpiński. On the parallel complexity of Hamiltonian cycle and matching problem on dense graphs. *Journal of Algorithms*, 15:367–384, 1993.
4. G. A. Dirac. Some theorems on abstract graphs. *Proceedings of the London Mathematical Society*, 2:69–81, 1952.
5. M.R. Garey, D.S. Johnson, and L.Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

6 Hiep Hàn and Mathias Schacht. Dirac-type results for loose Hamilton cycles in uniform hypergraphs. *Journal of Combinatorial Theory, Series B*, 100(3):332–346, 2010. doi:10.1016/j.jctb.2009.10.002.

7 Jie Han. Decision problem for perfect matchings in dense k-uniform hypergraphs. arXiv:1409.5931.

8 Jie Han and Yi Zhao. Minimum codegree threshold for Hamilton ℓ-cycles in k-uniform hypergraphs. *Journal of Combinatorial Theory, Series A*, 132(0):194–223, 2015. doi:10.1016/j.jcta.2015.01.004.

9 Richard Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, New York: Plenum:85–103, 1972.

10 Marek Karpiński, Andrzej Ruciński, and Edyta Szymańska. Computational complexity of the Hamiltonian cycle problem in dense hypergraphs. In *LATIN 2010: Theoretical Informatics, 9th Latin American Symposium, Oaxaca, Mexico, April 19-23, 2010. Proceedings*, pages 662–673, 2010.

11 Peter Keevash, Fiachra Knox, and Richard Mycroft. Polynomial-time perfect matchings in dense hypergraphs. *Advances in Mathematics*, 269:265–334, 2014.

12 Peter Keevash, Daniela Kühn, Richard Mycroft, and Deryk Osthus. Loose Hamilton cycles in hypergraphs. *Discrete Mathematics*, 311(7):544–559, 2011. doi:10.1016/j.disc.2010.11.013.

13 Daniela Kühn, Richard Mycroft, and Deryk Osthus. Hamilton ℓ-cycles in uniform hypergraphs. *Journal of Combinatorial Theory, Series A*, 117:910–927, 2010.

14 Daniela Kühn and Deryk Osthus. Loose Hamilton cycles in 3-uniform hypergraphs of high minimum degree. *Journal of Combinatorial Theory, Series B*, 96(6):767–821, 2006. doi:10.1016/j.jctb.2006.02.004.

15 Daniela Kühn and Deryk Osthus. Hamilton cycles in graphs and hypergraphs: an extremal perspective. *Proceedings of the International Congress of Mathematicians 2014, Seoul, Korea*, 4:381–406, 2014.

16 Vojtěch Rödl and Andrzej Ruciński. Dirac-type questions for hypergraphs – a survey (or more problems for Endre to solve). In Imre Bárány, József Solymosi, and Gáábor Sági, editors, *An Irregular Mind*, volume 21 of *Bolyai Society Mathematical Studies*, pages 561–590. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-14444-8_16.

17 Vojtěch Rödl, Andrzej Ruciński, and Endre Szemerédi. A Dirac-type theorem for 3-uniform hypergraphs. *Combinatorics, Probability and Computing*, 15(1-2):229–251, 2006. doi:10.1017/S0963548305007042.

18 Vojtěch Rödl, Andrzej Ruciński, and Endre Szemerédi. An approximate Dirac-type theorem for k-uniform hypergraphs. *Combinatorica*, 28(2):229–260, 2008. doi:10.1007/s00493-008-2295-z.

19 Vojtěch Rödl, Andrzej Ruciński, and Endre Szemerédi. Dirac-type conditions for Hamiltonian paths and cycles in 3-uniform hypergraphs. *Advances in Mathematics*, 227:1225–1299, 2011.

20 Edyta Szymańska. The complexity of almost perfect matchings and other packing problems in uniform hypergraphs with high codegree. *European Journal of Combinatorics*, 34(3):632–646, 2013. doi:10.1016/j.ejc.2011.12.009.

21 Yi Zhao. Recent advances on Dirac-type problems for hypergraphs. arXiv:1508.06170.

Efficiently Finding All Maximal α-gapped Repeats

Paweł Gawrychowski[1], Tomohiro I[2], Shunsuke Inenaga[3], Dominik Köppl[4], and Florin Manea[5]

1 **Institute of Informatics, University of Warsaw, Poland**
gawry@mimuw.edu.pl
2 **Department of Computer Science, TU Dortmund, Germany**
tomohiro.i@cs.tu-dortmund.de
3 **Department of Informatics, Kyushu University, Japan**
inenaga@inf.kyushu-u.ac.jp
4 **Department of Computer Science, TU Dortmund, Germany**
dominik.koeppl@cs.tu-dortmund.de
5 **Department of Computer Science, Kiel University, Germany**
flm@informatik.uni-kiel.de

Abstract

For $\alpha \geq 1$, an α-gapped repeat in a word w is a factor uvu of w such that $|uv| \leq \alpha|u|$; the two occurrences of a factor u in such a repeat are called *arms*. Such a repeat is called *maximal* if its arms cannot be extended simultaneously with the same symbol to the right nor to the left. We show that the number of all maximal α-gapped repeats occurring in words of length n is upper bounded by $18\alpha n$, allowing us to construct an algorithm finding all maximal α-gapped repeats of a word on an integer alphabet of size $n^{\mathcal{O}(1)}$ in $\mathcal{O}(\alpha n)$ time. This result is optimal as there are words that have $\Theta(\alpha n)$ maximal α-gapped repeats. Our techniques can be extended to get comparable results in the case of α-gapped palindromes, i.e., factors $uvu^{\intercal}$ with $|uv| \leq \alpha|u|$.

1998 ACM Subject Classification G.2.1 Combinatorics, I.1.2 Algorithms

Keywords and phrases combinatorics on words, counting algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.39

1 Introduction

Gapped repeats and palindromes are repetitive structures occurring in words that were investigated extensively within theoretical computer science (see, e.g., [11, 3, 14, 15, 16, 4, 6, 5, 10, 7, 17] and the references therein) with motivation coming especially from the analysis of DNA and RNA structures, modelling different types of tandem and interspersed repeats as well as hairpin structures; such structures are important in analysing the structural and functional information of the genetic sequences (see, e.g., [11, 3, 15]).

Besides introducing the definitions of (maximal) α-gapped repeats and palindromes, both papers [15, 16] lead to combinatorial and algorithmic problems that extend the classical results obtained for squares and palindromes. In fact, problems like how many maximal α-gapped repeats or palindromes can a word of length n contain, how efficiently can we compute the set of maximal α-gapped repeats or palindromes in a word, how efficiently can we compute the α-gapped repeat or palindrome with the longest arm, were already investigated [15, 3, 16, 10, 17, 5]. In this article we obtain the following results:

- The number of all maximal α-gapped repeats in a word of length n is at most $18\alpha n$.
- We can compute the list of all α-gapped repeats in $\mathcal{O}(\alpha n)$ time for *integer* alphabets.

© Paweł Gawrychowski, Tomohiro I, Shunsuke Inenaga, Dominik Köppl, and Florin Manea;
licensed under Creative Commons License CC-BY

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 39; pp. 39:1–39:14

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

Our techniques can be extended to show that the number of all maximal α-gapped palindromes in a word of length n is upper bounded by $28\alpha n + 7n$; they can be found in $\mathcal{O}(\alpha n)$ time. As there are words of length n that contain $\Theta(\alpha n)$ maximal α-gapped repeats (see [16]), it follows that our obtained bounds on the number of all maximal α-gapped repeats are asymptotically tight, and that we cannot hope for algorithms finding all α-gapped repeats faster in the worst case.

Our results improve those of [16] (as well as those existing in the literature before [16]). There, the authors present an algorithm computing all maximal α-gapped repeats in $\mathcal{O}(\alpha^2 n + \mathrm{occ}(n))$ time for integer alphabets, where $\mathrm{occ}(n)$ is the number of all maximal α-gapped repeats occurring in a word of length n. Further, they proved that $\mathrm{occ}(n) = \mathcal{O}(\alpha^2 n)$.

An alternative proof for the upper bound $\mathrm{occ}(n) = \mathcal{O}(\alpha n)$ was given in the very recent paper [5]. However, compared to that paper, we present a more direct proof of the $\mathcal{O}(\alpha n)$ upper bound as well as a concrete evaluation of the constant hidden by the $\mathcal{O}$-denotation.

The algorithms given in [5, 17] compute all maximal α-gapped repeats of a word in $\mathcal{O}(\alpha n + \mathrm{occ}(n))$. In the light of the upper bound $\mathcal{O}(\alpha n)$ on $\mathrm{occ}(n)$, it follows that these algorithms work in $\mathcal{O}(\alpha n)$ time, but only for constant alphabets. Extending the approach in [10], we devise an algorithm for the same problem with integer alphabets. The algorithm requires a deeper analysis than the one developed in [10] for finding the longest α-gapped repeat, and uses essentially different techniques and data structures than the ones in [5, 17].

A related problem is the computation of all factors with an exponent less than 2 that are maximal wrt. their exponents. This problem was recently investigated in [1].

2 Combinatorics on Words

Let Σ be a finite alphabet; Σ^* denotes the set of all finite words over Σ. The ***length*** of a word $w \in \Sigma^*$ is denoted by $|w|$.

For $v = xuy$ with $x, u, y \in \Sigma^*$, we call x, u and y a ***prefix***, ***factor***, and ***suffix*** of v, respectively. We denote by $w[i]$ the symbol occurring at position i in w, and by $w[i, j]$ the factor of w starting at position i and ending at position j, consisting of the catenation of the symbols $w[i], \ldots, w[j]$, where $1 \leq i \leq j \leq n$; we say that $w[i, j]$ is empty if $i > j$. By $w^\intercal$ we denote the ***mirror image*** of w. A ***period*** of a word w over Σ is a positive integer p such that $w[i] = w[j]$ for all i and j with $i \equiv j \pmod{p}$; a word that has period p is also called ***p-periodic***. Let $per(w)$ be the smallest period of w. A word w with $per(w) \leq \frac{|w|}{2}$ is called ***periodic***; otherwise, w is called ***aperiodic***. It is worth noting that the length of the overlap between two consecutive occurrences of an aperiodic factor v in w is upper bounded by $\frac{|v|}{2}$.

By $\mathcal{I} = [b, e]$ we represent the set of consecutive integers from b to e, for $b \leq e$, and call $\mathcal{I}$ an ***interval***. For an interval $\mathcal{I}$, we use the notations $\mathsf{b}(\mathcal{I})$ and $\mathsf{e}(\mathcal{I})$ to denote the beginning and end of $\mathcal{I}$; i.e., $\mathcal{I} = [\mathsf{b}(\mathcal{I}), \mathsf{e}(\mathcal{I})]$. We write $|\mathcal{I}|$ to denote the length of $\mathcal{I}$; i.e., $|\mathcal{I}| = \mathsf{e}(\mathcal{I}) - \mathsf{b}(\mathcal{I}) + 1$. A ***subword*** u of a word w is a pair $(s, [b, e])$ consisting of a factor s of w and an interval $[b, e]$ in w such that $s = w[b, e]$. While a factor is identified only by a sequence of letters, a subword is also identified by its position in the word. So subwords are always unique, while a word may contain multiple occurrences of the same factor. For two subwords u and $\overline{u}$ of a word w, we write $u = \overline{u}$ if they start at the same position in w and have the same length. We write $u \equiv \overline{u}$ if the factors identifying these subwords are the same. We implicitly use subwords both like factors of w and as intervals contained in $[1, |w|]$, e.g., we write $u \subseteq \overline{u}$ if two subwords $u = (s, [b, e]), \overline{u} = (\overline{s}, [\overline{b}, \overline{e}])$ of w satisfy $[b, e] \subseteq [\overline{b}, \overline{e}]$, i.e., $\mathsf{b}(\overline{u}) \leq \mathsf{b}(u) \leq \mathsf{e}(u) \leq \mathsf{e}(\overline{u})$. Two subwords u and $\overline{u}$ of the same word w are called ***consecutive***, iff $\mathsf{e}(u) + 1 = \mathsf{b}(\overline{u})$.

For a word w, we call a triple of consecutive subwords u_λ, v, u_ρ a ***gapped repeat*** with ***period*** $|u_\lambda v|$ and ***gap*** $|v|$ iff $u_\rho \equiv u_\lambda$. A triple of consecutive subwords u_λ, v, u_ρ is called a ***gapped palindrome*** with gap $|v|$ iff $u_\rho \equiv u_\lambda{}^\intercal$. The subwords u_λ and u_ρ are called left and right ***arm***, respectively. For $\alpha \geq 1$, the gapped repeat (palindrome) u_λ, v, u_ρ is called α***-gapped*** iff $|u_\lambda| + |v| \leq \alpha\,|u_\lambda|$. Further, it is called ***maximal*** iff its arms cannot be extended simultaneously to the right nor to the left. Let $\mathcal{G}_\alpha(w)$ (respectively, $\mathcal{G}^\intercal_\alpha(w)$) denote the set of maximal α-gapped repeats (palindromes) in w. The representation of a maximal gapped repeat (palindrome) by the subword $z := w[u_\lambda]w[v]w[u_\rho]$ is not unique – the same subword z can be composed of gapped repeats (palindromes) with different periods (different gaps). Instead, a maximal gapped repeat (palindrome) is uniquely determined by its left arm u_λ and its period (gap). By fixing w, we thus can map u_λ, v, u_ρ injectively to the pair of integers $(\mathsf{e}(u_\lambda), |u_\lambda v|)$ in case of gapped repeats, or to $(\mathsf{e}(u_\lambda), |v|)$ in case of gapped palindromes.

A ***repetition*** in a word w is a periodic factor; a ***run*** is a maximal repetition; the ***exponent*** of a run is the number of times the period fits in that run. For a word w, let $E(w)$ denote the number of runs and the sum of the exponents of runs in w, respectively. The exponent of a run r is denoted by $\exp(r)$. We use the following results from literature:

▶ **Lemma 1** ([2]). *For a word w, $E(w) < 3\,|w|$, and the number of runs is less than $|w|$.*

▶ **Corollary 2** ([5, Conclusions]). *The number of maximal 1-gapped repeats is less than n.*

▶ **Observation 3.** *The mirror image of a gapped repeat (palindrome) is a gapped repeat (palindrome) with the same period. Hence, there exist the bijections $\mathcal{G}_\alpha(w) \sim \mathcal{G}_\alpha(w^\intercal)$ and $\mathcal{G}^\intercal_\alpha(w) \sim \mathcal{G}^\intercal_\alpha(w^\intercal)$.*

2.1 Point Analysis

A pair of positive integers is called a ***point***. We use points to bound the cardinality of a subset of gapped repeats and gapped palindromes by injectively mapping a gapped repeat (palindrome) to a point as stated above. To this end, we show that some vicinity of any point generated by a member of this subset does not contain any point that is generated by another member. This vicinity is given by

▶ **Definition 4.** For any $\gamma \in (0, 1]$, we say that a point (x, y) γ***-covers*** a point (x', y') iff $x - \gamma y \leq x' \leq x$ and $y - \gamma y \leq y' \leq y$.

It is crucial that the γ factor is always multiplied with the y-coordinates. In other words, the number of γ-covers of a point $(\cdot, y)$ correlates with γ and the value y. The main property of this definition is given by

▶ **Lemma 5.** *For any $\gamma \in (0, 1]$, let $S \subset [1, n]^2 \subset \mathbb{N}^2$ be a set of points such that no two distinct points in S γ-cover the same point. Then $|S| < 3n/\gamma$.*

Proof. We estimate the maximal number of points that can be placed in $[1, n]^2 \subset \mathbb{N}^2$ such that their covered points are disjoint. First, the number of points $(\cdot, y) \in [1, n]^2$ with $y < 1/\gamma$ is less than n/γ. Second, if a point $(\cdot, y)$ satisfies $2^l/\gamma \leq y < 2^{l+1}/\gamma$ for some integer $l \geq 0$, the point $(\cdot, y)$ γ-covers at least $2^l \times 2^l$ points, or to put it differently, this point γ-covers at least 2^l points $(\cdot, y')$ with $y - 2^l \leq y' \leq y$. In other words, there are at most $n/(2^l\gamma)$ points in S with $2^l/\gamma \leq y < 2^{l+1}/\gamma$. Hence, $|S| < n/\gamma + \sum_{l=0}^{\infty} n/(2^l\gamma) = 3n/\gamma$. ◀

Kolpakov et al. [16] split the set of maximal α-gapped repeats into three subsets, and studied the maximal size of each subset. They analysed maximal α-gapped repeats by

partitioning them into three subsets: those whose arms are contained in one or two runs, those whose arms contain a periodic prefix or suffix larger than half of the size of the arms, and those belonging to neither of the two subsets.

They showed that the first two subsets contain at most $\mathcal{O}(\alpha n)$ elements. The point analysis is used as a tool for studying the last subset. By mapping a gapped repeat to a point consisting of the end position of its left arm and its period, they showed that the points created by two different maximal α-gapped repeats cannot $\frac{1}{4\alpha}$-cover the same point. By this property, they bounded the size of the last subset by $\mathcal{O}(\alpha^2 n)$. Lemma 5 immediately improves this bound of $\mathcal{O}(\alpha^2 n)$ to $\mathcal{O}(\alpha n)$. Consequently, it shows that the number of maximal α-gapped repeats of a word of length n is $\mathcal{O}(\alpha n)$.

2.2 Upper Bound for the Number of Maximal α-gapped Repeats

We optimize the proof technique from [16] and improve the upper bound of the number of maximal α-gapped repeats in a word of length n from $\mathcal{O}(\alpha n)$ to $18\alpha n$. Unlike [16, 5], we partition the maximal α-gapped repeats differently. We categorize a gapped repeat depending on whether their left arm contains a periodic prefix or not. The two subsets are treated differently. For the ones having a periodic prefix, we think about the number of runs covering this prefix. The other category is analysed by using the results of Section 2.1. We begin with a formal definition of both subsets and analyse the former subset.

Let $0 < \beta < 1$. A gapped repeat $\sigma = u_\lambda, v, u_\rho$ belongs to $\beta\mathcal{P}_\alpha(w)$ iff u_λ contains a periodic prefix of length at least $\beta\,|u_\lambda|$. We call σ ***periodic***. Otherwise $\sigma \in \overline{\beta\mathcal{P}}_\alpha(w)$, where $\overline{\beta\mathcal{P}}_\alpha(w) := \mathcal{G}_\alpha(w) \setminus \beta\mathcal{P}_\alpha(w)$; we call σ ***aperiodic***.

▶ **Lemma 6.** *Let w be a word, $\alpha > 1$ and $0 < \beta < 1$ two real numbers. Then $|\beta\mathcal{P}_\alpha(w)|$ is at most $2\alpha E(w)/\beta$.*

Proof. Let $\sigma = (u_\lambda, v, u_\rho) \in \beta\mathcal{P}_\alpha(w)$. By definition, the left arm u_λ has a periodic prefix s_λ of length at least $\beta\,|u_\lambda|$. Let r_λ denote the run that generates s_λ, i.e., $s_\lambda \subseteq r_\lambda$ and they both have the common shortest period p. By the definition of gapped repeats, there is a right copy s_ρ of s_λ contained in u_ρ with $s_\rho = w[\mathsf{b}(s_\lambda) + |u_\lambda v|\,, \mathsf{e}(s_\lambda) + |u_\lambda v|] \equiv s_\lambda$.

Let r_ρ be a run generating s_ρ (it is possible that r_ρ and r_λ are identical). By definition, r_ρ has the same period p as r_λ. In the following, we will see that σ is uniquely determined by r_λ and the period $q := |u_\lambda v|$, if σ is a periodic gapped repeat. We will fix r_λ and pose the question how many maximal periodic gapped repeats can be generated by r_λ.

Since σ is maximal, $\mathsf{b}(u_\lambda) = \mathsf{b}(r_\lambda)$ or $\mathsf{b}(u_\rho) = \mathsf{b}(r_\rho)$ must hold; otherwise we could extend σ to the left. We analyse the case $\mathsf{b}(s_\lambda) = \mathsf{b}(r_\lambda)$, the other is treated exactly in the same way by symmetry. The gapped repeat σ is identified by r_λ and the period q. We fix r_λ and count the number of possible values for the period q. Given two different gapped repeats σ_1 and σ_2 with respective periods q_1 and q_2 such that the left arms of both are generated by r_λ, the difference between q_1 and q_2 must be at least p.

Since $|u_\lambda| \le |s_\lambda|\,/\beta$ and σ is α-gapped, $1 \le q \le |s_\lambda|\,\alpha/\beta \le |r_\lambda|\,\alpha/\beta$. Then the number of possible periods q is bounded by $|r_\lambda|\,\alpha/(\beta p) = \exp(r_\lambda)\alpha/\beta$. Therefore the number of maximal α-gapped repeats is bounded by $\alpha E(w)/\beta$ for the case $\mathsf{b}(u_\lambda) = \mathsf{b}(r_\lambda)$. Summing up we get the bound $2\alpha E(w)/\beta$. ◀

Remembering the results of Section 2.1, we map gapped repeats to their respective points. By using the period as the y-coordinate, one can show Lemma 7.

▶ **Lemma 7.** *Given a word w, and two real numbers $\alpha > 1$ and $2/3 \le \beta < 1$. The points mapped by two different maximal gapped repeats in $\overline{\beta\mathcal{P}}_\alpha(w)$ cannot $\frac{1-\beta}{\alpha}$-cover the same point.*

Proof. Let $\sigma = u_\lambda, v, u_\rho$ and $\overline{\sigma} = \overline{u_\lambda}, \overline{v}, \overline{u_\rho}$ be two different maximal gapped repeats in $\overline{\beta\mathcal{P}_\alpha(w)}$. Set $u := |u_\lambda| = |u_\rho|$, $\overline{u} := |\overline{u_\lambda}| = |\overline{u_\rho}|$, $q := |u_\lambda v|$ and $\overline{q} := |\overline{u_\lambda v}|$. We map the maximal gapped repeats σ and $\overline{\sigma}$ to the points $(\mathsf{e}(u_\lambda), q)$ and $(\mathsf{e}(\overline{u_\lambda}), \overline{q})$, respectively. Assume, for the sake of contradiction, that both points $\frac{1-\beta}{\alpha}$-cover the same point (x, y).

Let $z := |\mathsf{e}(u_\lambda) - \mathsf{e}(\overline{u_\lambda})|$ be the difference of the endings of both left arms, and $s_\lambda := w[[\mathsf{b}(u_\lambda), \mathsf{e}(u_\lambda)] \cap [\mathsf{b}(\overline{u_\lambda}), \mathsf{e}(\overline{u_\lambda})]]$ be the overlap of u_λ and $\overline{u_\lambda}$. Let $s := |s_\lambda|$, and let s_ρ (resp. $\overline{s_\rho}$) be the right copy of s_λ based on σ (resp. $\overline{\sigma}$).

Sub-Claim: The overlap s_λ is not empty, and $s_\rho \neq \overline{s_\rho}$

Sub-Proof. Assume for this sub-proof that $\mathsf{e}(u_\lambda) < \mathsf{e}(\overline{u_\lambda})$ (otherwise exchange σ with $\overline{\sigma}$, or yield the contradiction $\sigma = \overline{\sigma}$). By combining the $(1-\beta)/\alpha$-cover property with the fact that $\overline{\sigma}$ is α-gapped, we yield $\mathsf{e}(\overline{u_\lambda}) - \overline{u} \leq \mathsf{e}(\overline{u_\lambda}) - \overline{q}/\alpha \leq \mathsf{e}(\overline{u_\lambda}) - \overline{q}(1-\beta)/\alpha \leq x \leq \mathsf{e}(u_\lambda) < \mathsf{e}(\overline{u_\lambda})$. So the subword $w[\mathsf{e}(u_\lambda)]$ is contained in $\overline{u_\lambda}$. If $s_\rho = \overline{s_\rho}$, then we get a contradiction to the maximality of σ: By the above inequality, $w[\mathsf{e}(u_\lambda) + 1]$ is contained in $\overline{u_\lambda}$, too. Since $\overline{\sigma}$ is a gapped repeat, the character $w[\mathsf{e}(u_\lambda) + 1]$ occurs in $\overline{u_\rho}$, exactly at $w[\mathsf{e}(u_\rho) + 1]$. ◀

So $q \neq \overline{q}$. Without loss of generality let $q < \overline{q}$. Then

$$\overline{q} - \frac{\overline{q}(1-\beta)}{\alpha} \leq y \leq q \leq \overline{q}. \tag{1}$$

$$\text{So the difference of both periods is } 0 < \delta := \overline{q} - q \leq \overline{q}(1-\beta)/\alpha \leq \overline{u}(1-\beta). \tag{2}$$

$$\text{Eq. (1) also yields that } u \geq q/\alpha \geq \frac{\overline{q}}{\alpha}(1 - \frac{1-\beta}{\alpha}) \geq \overline{q}\beta/\alpha. \tag{3}$$

Since $s_\rho = [\mathsf{b}(s_\lambda) + q, \mathsf{e}(s_\lambda) + q]$ and $\overline{s_\rho} = [\mathsf{b}(s_\lambda) + \overline{q}, \mathsf{e}(s_\lambda) + \overline{q}]$, we have $\mathsf{b}(\overline{s_\rho}) - \mathsf{b}(s_\rho) = \delta$.

By case analysis, we show that u_λ or $\overline{u_\lambda}$ has a periodic prefix, which leads to the contradiction that σ or $\overline{\sigma}$ are in $\beta\mathcal{P}_\alpha(w)$.

1. Case: $\mathsf{e}(u_\lambda) \leq \mathsf{e}(\overline{u_\lambda})$. Since $\mathsf{e}(\overline{u_\lambda}) - \overline{q}(1-\beta)/\alpha \leq x \leq \mathsf{e}(u_\lambda) \leq \mathsf{e}(\overline{u_\lambda})$,

$$z = \mathsf{e}(\overline{u_\lambda}) - \mathsf{e}(u_\lambda) \leq \overline{q}(1-\beta)/\alpha \leq \overline{u}(1-\beta). \tag{4}$$

1a. Sub-Case: $\mathsf{b}(u_\lambda) \leq \mathsf{b}(\overline{u_\lambda})$. By Eq. (4), we get $s = \overline{u} - z \geq \overline{u}\beta$. It follows from Eq. (2) and $2/3 \leq \beta < 1$ that $s/\delta \geq \overline{u}\beta/\overline{u}(1-\beta) = \beta/(1-\beta) \geq 2$, which means that s_ρ and $\overline{s_\rho}$ overlap at least half of their common length, so s_λ is periodic. Since s_λ is a prefix of $\overline{u_\lambda}$ of length $s \geq \overline{u}\beta$, $\overline{\sigma}$ is in $\beta\mathcal{P}_\alpha(w)$, a contradiction.

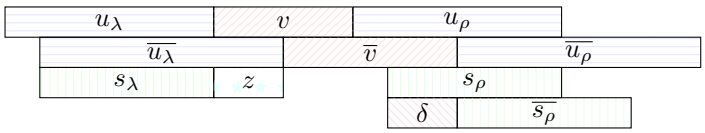


Figure 1 Sub-Case 1a.

1b. Sub-Case: $\mathsf{b}(u_\lambda) > \mathsf{b}(\overline{u_\lambda})$. We conclude that $s_\lambda = u_\lambda$. It follows from Eqs. (2) and (3) and $2/3 \leq \beta < 1$ that $s/\delta \geq \overline{q}\alpha\beta/(\overline{q}\alpha(1-\beta)) = \beta/(1-\beta) \geq 2$, which means that $s_\lambda = u_\lambda$ is periodic. Hence σ is in $\beta\mathcal{P}_\alpha(w)$, a contradiction.

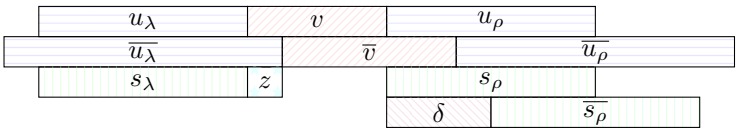


Figure 2 Sub-Case 1b.

2. Case: $\mathsf{e}(u_\lambda) > \mathsf{e}(\overline{u_\lambda})$. Since $\mathsf{e}(u_\lambda) - q(1-\beta)/\alpha \le x \le \mathsf{e}(\overline{u_\lambda}) \le \mathsf{e}(u_\lambda)$,

$$z = \mathsf{e}(u_\lambda) - \mathsf{e}(\overline{u_\lambda}) \le q(1-\beta)/\alpha \le \overline{q}(1-\beta)/\alpha \le \overline{u}(1-\beta). \tag{5}$$

2a. Sub-Case: $\mathsf{b}(u_\lambda) \le \mathsf{b}(\overline{u_\lambda})$. We conclude that $s_\lambda = \overline{u_\lambda}$. It follows from Eq. (2) and $2/3 \le \beta < 1$ that $s/\delta \ge \overline{u}/(\overline{u}(1-\beta)) = 1/(1-\beta) \ge 3 > 2$, which means that $s_\lambda = \overline{u_\lambda}$ is periodic. Hence $\overline{\sigma}$ is in $\beta\mathcal{P}_\alpha(w)$, a contradiction.

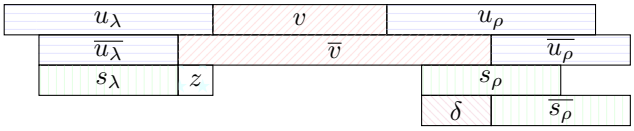


Figure 3 Sub-Case 2a.

2b. Sub-Case: $\mathsf{b}(u_\lambda) > \mathsf{b}(\overline{u_\lambda})$. By Eq. (5) we get $z \le q(1-\beta)/\alpha \le u(1-\beta)$ and hence $s = u - z \ge u\beta$. If $\delta \le s/2$, s_ρ and $\overline{s_\rho}$ overlap at least half of their common length, which leads to the contradiction that u_λ has a periodic prefix s_λ of length at least $u\beta$. Otherwise, let us assume that $s/2 < \delta$. By Eqs. (2) and (3) we get $u/\delta \ge \overline{q}\alpha\beta/(\overline{q}\alpha(1-\beta)) = \beta/(1-\beta) \ge 2$ with $2/3 \le \beta < 1$. Hence, δ is upper bounded by $u/2$; so u_ρ has a periodic prefix of length at least 2δ (since $2\delta > s \ge u\beta$), a contradiction.

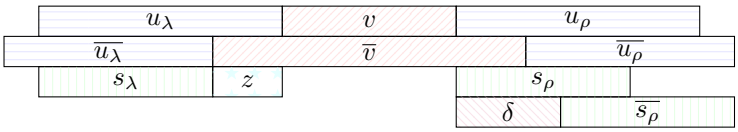


Figure 4 Sub-Case 2b.

◀

The next lemma follows immediately from Lemmas 5 and 7.

▶ **Lemma 8.** *For $\alpha > 1$, $2/3 \le \beta < 1$ and a word w of length n, $\left|\overline{\beta\mathcal{P}}_\alpha(w)\right| < 3\alpha n/(1-\beta)$.*

▶ **Theorem 9.** *Given a word w of length n, and a real number $\alpha > 1$. Then $|\mathcal{G}_\alpha(w)| < 18\alpha n$.*

Proof. Combining the results of Lemmas 6 and 8, $|\mathcal{G}_\alpha(w)| = |\beta\mathcal{P}_\alpha(w)| + \left|\overline{\beta\mathcal{P}}_\alpha(w)\right| < 2\alpha E(w)/\beta + 3\alpha n/(1-\beta)$ for $2/3 \le \beta < 1$. Applying Lemma 1, the term is upper bounded by $6\alpha n/\beta + 3\alpha n/(1-\beta)$. The number is minimal for $\beta = 2/3$, yielding the bound $18\alpha n$. ◀

With Corollary 2 we obtain the result of Theorem 9 for $\alpha \ge 1$.

We can bound the number of maximal α-gapped palindromes by similar proofs to $28\alpha n + 7n$. This bound solves an open problem in [15], where Kolpakov and Kucherov

conjectured that the number of α-gapped palindromes with $\alpha \geq 2$ in a string is linear. We briefly explain the main differences and similarities needed to understand the relationship between gapped repeats and palindromes. Let σ be a maximal α-gapped repeat (or α-gapped palindrome). If the gapped repeat (palindrome) has a periodic prefix s_λ generated by some run, the right arm has a periodic prefix (suffix) s_ρ generated by a run of the same period. Since σ is maximal, both runs have to obey constraints that are similar in both cases, considering whether σ is a gapped repeat or a gapped palindrome. So it is easy to change the proof of Lemma 6 in order to work with palindromes. Like with aperiodic gapped repeats, we can apply the point analysis to the aperiodic α-gapped palindromes, too. Our main idea is to map a gapped palindrome u_λ, v, u_ρ injectively to the pair of integers $(\mathsf{e}(u_\lambda), |v|)$, exchanging the period with the size of the gap. Details are provided in the full version of the paper [9].

3 Finding All Maximal α-gapped Repeats

The computational model we use to design and analyse our algorithms is the standard unit-cost RAM with logarithmic word size, which is generally used in the analysis of algorithms. In the upcoming algorithmic problems, we assume that the words we process are sequences of integers. In general, if the input word has length n then we assume its letters are in $\{1, \ldots, n\}$, so each letter fits in a single memory-word. This is a common assumption in stringology (see, e.g., the discussion in [12]). For a word w, $|w| = n$, we build in $\mathcal{O}(n)$ time the suffix array as well as data structures allowing us to retrieve in constant time the length of the longest common prefix of any two suffixes $w[i, n]$ and $w[j, n]$ of w, denoted $LCP_w(i, j)$ (the subscript w is omitted when there is no danger of confusion). In what follows, such structures are called *LCP* data structures (see, e.g., [12, 11]). We begin with a simple lemma.

▶ **Lemma 10.** *Given a word w, $|w| = n$, we can process it in $\mathcal{O}(n)$ time such that, for each $i, p \leq n$, we can return in $\mathcal{O}(1)$ time the longest factor of period p starting at position i in w.*

Let w be a word and v be a factor of w with $per(v) = p$. Further, let z be a subword of length $\ell|v|$ of w. An occurrence of v in z is a subword $(v, [i, i + |v| - 1])$ of z; we say that v ***occurs*** at position i in z. For an easier presentation of our algorithm, we distinguish between two types of occurrences of v in z. On the one hand, we have the so-called ***single occurrences***. If v is aperiodic, then all its occurrences in z are single occurrences; there are $\mathcal{O}(\ell)$ such occurrences (see, e.g., [13]). If v is periodic, then a subword $(v, [i, i + |v| - 1])$ of z starting at position i in z is a single occurrence if v occurs neither at position $i - p$ nor at position $i + p$ in z. On the other hand, we have ***occurrences of v within a run*** of z, whose period is $p = per(v)$. That is, the subword $(v, [i, i + |v| - 1])$ starting at position i in z is an occurrence of v within a run if v occurs either at $i - p$ or at $i + p$. We say that $(v, [i, i + |v| - 1])$ is the ***first occurrence*** of v in a run of period p of z if v does not occur at $i - p$ but occurs at $i + p$. Note that there are $\mathcal{O}(\ell)$ runs containing occurrences of v in z, or, equivalently, $\mathcal{O}(\ell)$ first occurrences of v in runs of period p.

Consequently, the occurrences of v in z can be succinctly represented as follows. For the single occurrences we just store their starting position. The occurrences of v in a run r can be represented by the starting position of the first occurrence of v in r, together with the period of v, since the starting positions of the occurrences of v in r form an arithmetic progression of period p.

In our approach, basic factors (i.e., factors of length 2^k, for $k \geq 1$) of the input word are important. For some integer $c \geq 2$, the occurrences of the basic factor $w[i, i + 2^k - 1]$ in a subword of length $c2^k$ can be represented in a compact manner: $\mathcal{O}(c)$ positions of the single

occurrences of $w[i, i + 2^k - 1]$ and $\mathcal{O}(c)$ first occurrences of $w[i, i + 2^k - 1]$ in runs, together with the period of $w[i, i + 2^k - 1]$. We need the next lemma (see [10, 13]).

▶ **Lemma 11.** *Given a word w of length n and an integer $c \geq 2$, we can process w in time $\mathcal{O}(n \log n)$ such that given any basic factor $y = w[i, i + 2^k - 1]$ and any subword of w $(z, [j, j + c2^k - 1])$, with $k \geq 0$, we can compute in $\mathcal{O}(\log \log n + c)$ time the representation of all the (single and within runs) occurrences of y in z.*

We now focus on short basic factors of words. The constant 16 occurring in the following considerations can be replaced by any other constant; we just use it here so that we can apply these results directly in the main proofs of this section.

Given a word v and some integer $\beta \geq 16$ with $|v| = \beta \log n$, as well as a basic factor $y = v[i2^k + 1, (i + 1)2^k]$, with $i, k \geq 0$ and $i2^k + 1 > (\beta - 16) \log n$ (so occurring in the suffix of length $16 \log n$ of v), the occurrences of y in v can be represented as $\mathcal{O}(\beta)$ bit-sets, each containing $\mathcal{O}(\log n)$ bits, the 1-bits marking the starting positions of the occurrences of y in v. The next result can be shown using tools developed in [8] (see also [10]).

▶ **Lemma 12.** *Given a word v and an integer $\beta > 16$, with $|v| = \beta \log n$, we can process v in $\mathcal{O}(\beta \log n)$ time such that given any basic factor $y = v[i2^k + 1, (i + 1)2^k]$ with $i, k \geq 0$ and $i2^k + 1 > (\beta - 16) \log n$, we can find in $\mathcal{O}(\beta)$ time the $\mathcal{O}(\beta)$ bit-sets, each storing $\mathcal{O}(\log n)$ bits, characterizing all the occurrences of y in v.*

In the context of the previous lemma, once the occurrences of y in v are computed, given a subword z of v of length $|z| = c|y|$, for some $c \geq 1$, we can obtain in $\mathcal{O}(c)$ time both the single occurrences of y in z and the occurrences of y within runs of z. We just have to select (by bitwise operations on the bit-sets encoding the factors of v that overlap z) the positions where y occurs (so the positions of the 1-bits in those bit-sets). For each two consecutive such occurrences of y we detect whether they are part of a run in v and then skip over all the occurrences of y from that run (and the corresponding parts of the bit-sets) before looking again for 1-bits in the bit-sets; for the positions that form a run we store the first occurrence of y and its period, while for the single occurrences we store the position of that occurrence.

Now we can begin the presentation of the algorithm finding all the maximal α-gapped repeats of a word. We first show how to find maximal repeats with short arms.

▶ **Lemma 13.** *Given a word w and $\alpha \geq 1$, we can find all the maximal α-gapped repeats u_λ, u', u_ρ occurring in w, with $|u_\rho| \leq 16 \log n$, in time $\mathcal{O}(\alpha n)$.*

Proof. If a maximal α-gapped repeat u_λ, u', u_ρ (where we denote by u the underlying factor of both arms) has $|u| \leq 16 \log n$, we get that u_ρ must be completely contained in a subword $(w', [m \log n + 1, (m + 17) \log n])$, for some m with $\frac{n}{\log n} - 17 \geq m \geq 0$. By fixing the interval where u_ρ may occur (that is, fix m), we also fix the place where u_λ may occur. Indeed, the entire subword u_λ, u', u_ρ is completely contained in the factor $x_m = (w'', [(m - 16\alpha) \log n + 1, (m + 17) \log n])$ (or, in the factor $x_m = (w'', [1, (m + 17) \log n])$ if $(m - 16\alpha) \log n + 1 < 1$).

Hence, we look for maximal α-gapped repeats u_λ, u', u_ρ completely contained in x_m with u_ρ completely contained in the suffix of length $16 \log n$ of x_m; then we repeat this process for all m. To begin with, we process x_m as in Lemma 12, and construct *LCP*-structures for it.

Now, once we fixed the subword x_m of w where we search the maximal α-gapped repeats, we try to fix also their length. That is, we find all maximal α-gapped repeats u_λ, u', u_ρ with $2^{k+1} \leq |u| \leq 2^{k+2}$ completely contained in x_m with u_ρ completely contained in the suffix of length $16 \log n$ of x_m; we execute this process for all $0 \leq k \leq \log(16 \log n)$. Note that all the

maximal α-gapped repeats with arms shorter than 2 (occurring anywhere in the word w) can be trivially found in $\mathcal{O}(\alpha n)$ time.

Since we can find occurrences of basic factors in x_m efficiently, we try to build a maximal gapped repeat by extending gapped repeats whose arms contain a basic factor. To this end, we analyse some *subwords of* x_m: If $2^{k+1} \leq |u| \leq 2^{k+2}$ then u_ρ contains at least one subword $(y, [j2^k + 1, (j+1)2^k])$ starting within its first 2^k positions. A copy of the factor y occurs also within the first 2^k positions of u_λ (with the same offset with respect to the starting position of u_λ as the offset of the occurrence of y with respect to the starting position of u_ρ). So, finding the respective copy of y from u_λ helps us discover the place where u_λ actually occurs. Indeed, assume that we identified the copy of y from u_λ, and assume that this copy is $(y, [\ell + 1, \ell + |y|])$; we try to build u_λ and u_ρ around these two occurrences of y, respectively. Hence, in order to identify u_λ and u_ρ we compute the longest factor p of x_m that ends both at $j2^k$ and at ℓ and the longest factor s that starts both at $(j+1)2^k + 1$ and at $\ell + |y| + 1$. Now, if $\ell + |y| + |s| \leq j2^k - |p|$ then u_λ is obtained by concatenating p and s around $x_m[\ell + 1, \ell + |y|]$ while u_ρ is obtained by concatenating p and s to the left and, respectively, right of $x_m[j2^k + 1, (j+1)2^k]$; otherwise, the two occurrences of y do not determine a maximal repeat. Moreover, the repeat we determined is a valid solution of our problem only if its right arm contains position $j2^k + 1$ of x_m within its first 2^k positions.

Now we explain how to determine efficiently the copy of y around which we try to build u_λ. As $|u| < 2^{k+2}$ and $|y| = 2^k$ we get that the copy of y that corresponds to u_λ should be completely contained in the subword of x_m of length $\alpha 2^{k+2}$ ending at position $j2^k$. As said above, we already processed x_m to construct the data structures from Lemma 12. Therefore, we can obtain in $\mathcal{O}(\alpha)$ time a representation of all the occurrences of y inside the factor of length $\alpha 2^{k+2}$ ending at position $j2^k$. These occurrences can be single occurrences and occurrences within runs. There are $\mathcal{O}(\alpha)$ single occurrences, and we can process each of them individually, as explained, to find the maximal α-gapped repeat they determine together with the occurrence of y from u_ρ. However, it is not efficient to do the same for the occurrences of y within runs. For these (which are also $\mathcal{O}(\alpha)$ many) we proceed as follows.

Assume we have a repetition of y's inside the factor of x_m of length $\alpha 2^{k+2}$ ending at position $j2^k$. Let ℓ be the starting position of the first occurrence of y in this repetition and let p be the period of y. Now, using Lemma 10 we can determine the maximal p-periodic subword (a run of period p) r_λ of x_m containing this repetition of y-occurrences. Similarly, we can determine the maximal p-periodic subword (a run of period p) r_ρ that contains the occurrence of y from u_ρ (i.e., $x_m[j2^k + 1, (j+1)2^k]$). To determine efficiently the α-gapped repeats that contain $x_m[j2^k + 1, (j+1)2^k]$ in the right arm and a corresponding occurrence of y from r_λ in the left arm we analyse several cases.

Assume u_ρ starts at a position of r_ρ, other than its first one. Then u_λ should also start at the first position of r_λ (or we could extend both arms to the left, a contradiction to the maximality of the repeat). If u_ρ ends at a position to the right of r_ρ, then u_λ also ends at a position to the right of r_λ, and, moreover, the suffix of u_λ occurring after the end of r_λ and the suffix of u_ρ occurring after the end of r_ρ are equal, and can be computed by a longest common prefix query on x_m. This means that u_λ can be determined exactly (we know where it starts and where it ends) so u_ρ can also be determined exactly (we know where it ends), and we can check if the obtained repeat is indeed a maximal α-gapped repeat, and the arms fulfil the required length conditions (i.e., their length is between 2^{k+1} and 2^{k+2}, the right arm contains position $j2^k + 1$ of x_m within its first 2^k positions). If u_ρ ends exactly at the same position as r_ρ, then u_ρ is periodic of period p. We compute the longest p-periodic prefix u' of r_λ which is also a suffix of r_ρ. Since u_λ is longer than p, the α-gapped repeats under consideration have

the left arm $u_\lambda := r_\lambda[1..|u'| - pi]$ and the right arm $u_\rho := r_\rho[|r_\rho| - (|u'| - pi) + 1..|r_\rho|]$ for $i \geq 0$ such that the gap $v := (w, [\mathsf{e}(u_\lambda) + 1, \mathsf{b}(u_\rho) - 1])$ respects the condition $|u_\lambda v| \leq \alpha\,|u_\lambda|$. Clearly, we can output in $\mathcal{O}(1)$ time each such repeat.

The final case is when u_ρ ends at a position of r_ρ, other than its last position. In that case, we get that $u_\lambda = r_\lambda$ (or, otherwise, we could extend both arms to the right). Essentially, this means that we know exactly where u_λ is located and its length (and we continue only if this length is between 2^{k+1} and 2^{k+2}); so u_λ denotes a factor $z^h z'$ for some z of length p. Now, looking at the run r_ρ, we can get easily the position of the first occurrence of z in that run, and the position of its last occurrence. If the first occurrence is ℓ', then the occurrences of z have their starting positions ℓ', $\ell' + p, \ldots, \ell' + tp$ for some t. As we know the length of u_λ and the fact that u_λ, u', u_ρ is α-gapped, we can determine in constant time the values $0 \leq i \leq t$ such that u_ρ may start at position $\ell' + ip$, the repeat we obtain is α-gapped, and u_ρ contains position $j2^k + 1$ of x_m within its first 2^k positions. If u_ρ is a prefix of r_ρ we also have to check that we cannot extend simultaneously u_ρ and u_λ to the left; if u_ρ is a suffix of r_λ we have to check that we cannot extend simultaneously u_ρ and u_λ to the right. Then we can return the maximal α-gapped repeats we constructed.

The cases when u_ρ starts at the first position of r_ρ or when it starts at a position to the left of r_ρ can be treated similarly, and as efficiently.

This concludes our algorithm. Its correctness follows from the explanations above. Moreover, we can ensure that our algorithm finds and outputs each maximal repeat exactly once; this clearly holds when we analyse the repeats of x_m for each m separately. However, when moving from x_m to x_{m+1} we must also check that the right arm of each repeat we find is not completely contained in x_m (so, already found). This condition can be easily imposed in our search: when constructing the arms determined by a single occurrence of y, we check the containment condition separately; when constructing a repeat determined by a run of y-occurrences, we have to impose the condition that the right arm extends out of x_m when searching the starting positions of the possible arms.

Next, we compute the complexity of the algorithm. Once we fix m, k, and j, our process takes $\mathcal{O}(\alpha + N_{j,m,k})$ time, where $N_{j,m,k}$ is the number of maximal α-gapped repeats determined for the fixed m, j, k. So, the time complexity of the algorithm is:
$\mathcal{O}(n + \sum_{0 \leq m \leq n/\log n}(16\alpha \log n + \sum_{0 \leq k \leq \log(16 \log n)}(\sum_{j \leq 16 \log n/2^k}(\alpha + N_{j,m,k})))) = \mathcal{O}(\alpha n)$,
as the total number of maximal α-gapped repeats is $\mathcal{O}(\alpha n)$ and we need $\mathcal{O}(|x_m|)$ preprocessing time for each x_m and $\mathcal{O}(n)$ preprocessing time for w. ◀

Next, we find all maximal α-gapped repeats with longer arms.

▶ **Lemma 14.** *Given a word w and $\alpha \geq 1$, we can find all the maximal α-gapped repeats u_λ, u', u_ρ occurring in w, with $|u_\rho| > 16 \log n$, in time $\mathcal{O}(\alpha n)$.*

Proof. The general approach in proving this lemma is similar to that used in the proof of the previous result. Essentially, when identifying a new maximal α-gapped repeat, we try to fix the place and length of the right arm u_ρ of the respective repeat, which restricts the place where the left arm u_λ occurs. This allows us to fix some long enough subword of w as being part of the right arm, detect its occurrences that are possibly contained in the left arm, and, finally, to efficiently identify the actual repeat. The main difference is that we cannot use the result of Lemma 12, as we have to deal with repeats with arms longer than $16 \log n$. Instead, we will use the structures constructed in Lemma 11. However, to get the stated complexity, we cannot apply this lemma directly to the word w, but rather to an encoded variant of w.

Thus, the first step of the algorithm is to construct a word w', of length $\frac{n}{\log n}$, whose symbols, called ***blocks***, encode $\log n$ consecutive symbols of w grouped together. That is, the

first block of the new word corresponds to $w[1, \log n]$, the second one to $w[\log n + 1, 2 \log n]$, and so on. Hence, we have two versions of the word w: the original one, and the one where it is split in blocks. It is not hard to see that the blocks can be encoded into numbers between 1 and n in linear time. Indeed, we build the suffix array and LCP-data structures for w, and then we cluster together the suffixes of the suffix array that share a common prefix of length at least $\log n$. Then, all the suffixes of a cluster are given the same number (between 1 and n), and a block is given the number of the suffix starting with the respective block.

We can now construct in $\mathcal{O}(n)$ time the suffix arrays and LCP-data structures for both w and w', as well as the data structures of Lemma 11 for the word w'.

Now, we guess the length of the arms of the repeat. We try to find the maximal α-gapped repeats u_λ, u', u_ρ of w with $2^{k+1} \log n \leq |u_\lambda| \leq 2^{k+2} \log n$, $k \leq \log \frac{n}{\log n} - 2$. We fix k and split again the word w, this time in factors of length $2^k \log n$, called ***k*-blocks**. Assume that each split is exact (padding the word with some new symbols ensures this).

Now, if a maximal α-gapped repeat u_λ, u', u_ρ with $2^{k+1} \log n \leq |u_\lambda| \leq 2^{k+2} \log n$ exists, then it contains an occurrence of a k-block within its first $2^k \log n$ positions. So, let z be a k-block and assume that it is the first k-block occurring in u_ρ (in this way fixing a range where u_ρ may occur). Obviously, if u_ρ contains z, then u_λ also contains an occurrence of z; however, this occurrence is not necessarily starting at a position $j \log n + 1$ for some $j \geq 0$ (so, it is not necessarily a sequence of blocks). But, at least one of the factors of length $2^{k-1} \log n$ starting within the first $\log n$ positions of z (which are not necessarily sequences of blocks) must correspond, in fact, to a sequence of blocks from the left arm u_λ. So, let us fix now a factor y of length $2^{k-1} \log n$ that starts within the first $\log n$ positions of z (we try all of them in the algorithm, one by one). As said, the respective occurrence of y from u_ρ is not necessarily a sequence of blocks (so it cannot be mapped directly to a factor of w'). But, we look for an occurrence of y starting at one of the $\alpha 2^{k+2} \log n$ positions to the left of z, corresponding to a sequence of blocks, and assume that the respective occurrence is exactly the occurrence of y from u_λ.

By binary searching the suffix array of w' (using LCP-queries on w to compare the factors of $\log n$ symbols of y and the blocks of w', at each step of the search) we try to detect a factor of w' that encodes a word equal to y. Assume that we can find such a sequence y' of 2^{k-1} blocks of w' (otherwise, y cannot correspond to a sequence of blocks from u_λ, so we should try other factors of z instead). Using Lemma 11 for w', we get in $\mathcal{O}(\log \log |w'| + \alpha)$ time a representation of the occurrences of y' in the range of $\alpha 2^{k+2}$ blocks of w' occurring before the blocks of z; this range corresponds to an interval of w with a length of $\alpha 2^{k+2} \log n$.

Further, we process these occurrences of y' just like in the previous lemma. Namely, the occurrences of y' in that range are either single occurrences or occurrences within runs. Looking at their corresponding factors from w, we note that each of these factors fixes a possible left arm u_λ; this arm, together with the corresponding arm u_ρ can be constructed just like before. In the case of single occurrences (which are at most $\mathcal{O}(\alpha)$, again), we try to extend both the respective occurrence and the occurrence of y from u_ρ both to the left and, respectively, to the right, simultaneously, and see if we can obtain in this way the arms of a valid maximal α-gapped repeat. Note that we must check also that the length of the arm of the repeat is between 2^{k+1} and 2^{k+2}, and that z is the first k-block of the right arm. As before, complications occur when the occurrences of y' are within runs. In this case, the run of occurrences of y' does not necessarily give us the period of y, but a multiple of this period that can be expressed also as a multiple of $\log n$ (or, in other words, the minimum period of y is a multiple of the block-length). This, however, does not cause any problems, as the factor y from u_ρ should always correspond to a block sequence from u_λ, so definitely to one of the factors encoded in the run of occurrences of y'.

Therefore, by determining the maximal factor that contains y and has the same period as the repetition of y'-occurrences (with the period measured within w), we can perform a very similar analysis to the corresponding one from the case when we searched maximal α-gapped repeats with arms shorter than $16 \log n$.

It remains to prove that each maximal gapped repeat is counted only once. Essentially, the reason for this is that for two separate factors y_1 and y_2 (of length $2^{k-1} \log n$) occurring in the first $\log n$ symbols of z we cannot get occurrences of the corresponding factors y_1' and y_2' that define the same repeat; in that case, the distance between y_1' and y_2' should be at least one block, so the distance between y_1 and y_2 should be at least $\log n$, a contradiction. Similarly, if we have a factor y occurring in the first $\log n$ symbols of some k-block z_1 such that this factor determines an α-gapped maximal repeat, then the same maximal repeat cannot be determined by a factor of another k-block, since z_1 is the first k-block of u_ρ.

The correctness of the algorithm described above follows easily from the explanations given in the proofs of the last two lemmas. Let us evaluate its complexity. The preprocessing phase (construction of w' and of all the needed data structures) takes $\mathcal{O}(n)$ time. Further, we can choose k (and implicitly an interval for the length of the arms of the repeats) such that $k \leq \log \frac{n}{\log n} - 2$. After choosing k, we can choose a k-block z in $\frac{n}{2^k \log n}$ ways. Further, we analyse each factor y of length $2^{k-1} \log n$ starting within the first $\log n$ positions of the chosen k-block z. For each such factor y we find in $\mathcal{O}(\log \frac{n}{\log n} + \log\log n + \alpha)$ time the representation of the occurrences of the block encoding the occurrence of y from u_λ. From each of the $\mathcal{O}(\alpha)$ single occurrences we check whether it is possible to construct a maximal α-gapped repeat in $\mathcal{O}(1)$ time. We also have $\mathcal{O}(\alpha)$ occurrences of the block encoding y in runs, and each of them is processed in $\mathcal{O}(N_{z,y})$ time, where $N_{z,y}$ is the number of maximal α-gapped repeats we find for some z and y. Overall, this adds up to a total time of $\mathcal{O}(n \log n + \alpha n)$, as the total number of maximal α-gapped repeats in w is upper bounded by $\mathcal{O}(\alpha n)$. If $\alpha \geq \log n$, the statement of the lemma follows. If $\alpha < \log n$, we proceed as follows.

Initially, we run the algorithm only for $k > \log\log n$ and find the maximal α-gapped repeats $u_\lambda u' u_\rho$ with $2^{\log\log n} \log n \leq |u_\lambda|$, in $\mathcal{O}(\alpha n)$ time. Further, we search maximal α-gapped repeats with shorter arms. Now, $|u_\lambda|$ is upper bounded by $2^{\log\log n + 1} \log n = 2(\log n)^2$, so $|u_\lambda u' u_\rho| \leq \ell_0$, for $\ell_0 = \alpha \cdot 2(\log n)^2 + 2(\log n)^2 = 2(\alpha+1)(\log n)^2$. Such an α-gapped repeat $u_\lambda u' u_\rho$ is, thus, contained in (at least) one factor of length $2\ell_0$ of w, starting at a position of the form $1 + m\ell_0$ for $m \geq 0$. So, we take the factors $w[1 + m\ell_0, (m+2)\ell_0]$ of w, for $m \geq 0$, and apply for each such factor, separately, the same strategy as above to detect the maximal α-gapped repeats contained completely in each of them. To this end, we first encode the factors $w[1 + m\ell_0, (m+2)\ell_0]$ of w so that each of them corresponds to an integer in $[0, 2\ell_0]$; we attain the encoding of the factors $w[1 + m\ell_0, (m+2)\ell_0]$ by sorting the symbols of w in linear time, and then using their order. The total time needed to do that is $\mathcal{O}\Big(n + \alpha\ell_0 \frac{n}{\ell_0} + N_{\ell_0}\Big) = \mathcal{O}(\alpha n)$, where N_{ℓ_0} is the number of repeats we find; moreover, we can easily ensure that a maximal repeat is not output twice (that is, ensure always that the gapped repeats we produce were not already contained in a previously processed interval). Hence, we find all maximal α-gapped repeats $u_\lambda u' u_\rho$ with $2^{\log\log(2\ell_0)} \log(2\ell_0) \leq |u|$. This means we find all the maximal α-gapped repeats with $|u| \geq 2^{\log\log(2\ell_0)+1} \log(2\ell_0)$. Since $2^{\log\log(2\ell_0)+1} \log(2\ell_0) \leq 16 \log n$ (for n large enough, as $\alpha \leq \log n$), we can apply Lemma 13 for gapped repeats with an arm-length smaller than $2^{\log\log(2\ell_0)+1} \log(2\ell_0)$. ◀

Putting together the results of Lemmas 13 and 14 we get the following theorem.

▶ **Theorem 15.** *Given a word w and $\alpha \geq 1$, we can compute $\mathcal{G}_\alpha(w)$ in time $\mathcal{O}(\alpha n)$.*

By a completely similar approach we can compute $\mathcal{G}^{\intercal}_{\alpha}(w)$, generalizing the algorithm of [15]. To this end, we construct *LCP*-structures for $ww^{\intercal}$ (allowing us to test efficiently whether a factor $w[i,j]^{\intercal}$ occurs at some position in w). When we search the α-gapped palindromes u_λ, v, u_ρ (with $u_\rho \equiv u_\lambda{}^{\intercal}$), we split again w in blocks and k-blocks, for each $k \leq \log|w|$, to check whether there exists such an u_λ, v, u_ρ with $2^k \leq |u_\lambda| \leq 2^{k+1}$. This search is conducted pretty much as in the case of repeats, only that now when we fix some factor y of u_ρ, we have to look for the occurrences of $y^{\intercal}$ in the factor of length $\mathcal{O}(\alpha|u_\rho|)$ preceding it; the *LCP*-structures for $ww^{\intercal}$ are useful for this, because, as explained above, they allow us to efficiently search the mirror images of factors of w inside w. Thus, given a word w and $\alpha \geq 1$, we can compute $\mathcal{G}^{\intercal}_{\alpha}(w)$ in time $\mathcal{O}(\alpha n)$.

Acknowledgement. The work of Florin Manea was supported by the DFG grant 596676.

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. Proceedings*, volume 7608 of *Lecture Notes in Computer Science*, pages 61–72. Springer, 2012.

2 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The Runs Theorem. *CoRR*, abs/1406.0263, 2014. URL: `http://arxiv.org/abs/1406.0263`.

3 Gerth Stølting Brodal, Rune B. Lyngsø, Christian N. S. Pedersen, and Jens Stoye. Finding maximal pairs with bounded gap. In *CPM*, volume 1645 of *LNCS*, pages 134–149. Springer, 1999.

4 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Wojciech Rytter, and Tomasz Walen. Efficient algorithms for two extensions of LPF table: The power of suffix arrays. In *Proc. SOFSEM 2010*, volume 5901 of *LNCS*, pages 296–307, 2010.

5 Maxime Crochemore, Roman Kolpakov, and Gregory Kucherov. Optimal searching of gapped repeats in a word. *ArXiv e-prints 1309.4055*, 2015. `arXiv:1309.4055`.

6 Maxime Crochemore and German Tischler. Computing longest previous non-overlapping factors. *Inf. Process. Lett.*, 111(6):291–295, February 2011.

7 Marius Dumitran and Florin Manea. Longest gapped repeats and palindromes. In *Proc. MFCS 2015*, volume 9234 of *LNCS*, pages 205–217. Springer, 2015.

8 Pawel Gawrychowski. Pattern matching in Lempel-Ziv compressed strings: Fast, simple, and deterministic. In *Proc. ESA*, volume 6942 of *LNCS*, pages 421–432, 2011.

9 Pawel Gawrychowski, Tomohiro I, Shunsuke Inenaga, Dominik Köppl, and Florin Manea. Efficiently finding all maximal α-gapped repeats. *ArXiv e-prints abs/1509.09237*, 2015.

10 Pawel Gawrychowski and Florin Manea. Longest α-gapped repeat and palindrome. In *Proc. FCT 2015*, volume 9210 of *LNCS*, pages 27–40. Springer, 2015.

11 Dan Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge University Press, New York, NY, USA, 1997.

12 Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *J. ACM*, 53:918–936, 2006.

13 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Efficient data structures for the factor periodicity problem. In *Proc. SPIRE*, volume 7608 of *LNCS*, pages 284–294, 2012.

14 Roman Kolpakov and Gregory Kucherov. Finding repeats with fixed gap. In *Proc. SPIRE*, pages 162–168, 2000.

15 Roman Kolpakov and Gregory Kucherov. Searching for gapped palindromes. *Theoretical Computer Science*, 410(51):5365–5373, 2009. Combinatorial Pattern Matching. doi:10.1016/j.tcs.2009.09.013.

16 Roman Kolpakov, Mikhail Podolskiy, Mikhail Posypkin, and Nickolay Khrapov. Searching of gapped repeats and subrepetitions in a word. In *Proc. CPM*, volume 8486 of *LNCS*, pages 212–221, 2014.

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 *Proc. SPIRE 2015*, volume 9309 of *LNCS*, pages 124–136. Springer, 2015.

On the Number of Lambda Terms With Prescribed Size of Their De Bruijn Representation*

Bernhard Gittenberger[1] and Zbigniew Gołębiewski[2]

1 **Institute for Discrete Mathematics and Geometry, Technische Universität Wien, Wiedner Hauptstraße 8-10/104, 1040 Wien, Austria**
gittenberger@dmg.tuwien.ac.at

2 **Institute for Discrete Mathematics and Geometry, Technische Universität Wien, Wiedner Hauptstraße 8-10/104, 1040 Wien, Austria**
zbigniew.golebiewski@pwr.edu.pl

Abstract

John Tromp introduced the so-called 'binary lambda calculus' as a way to encode lambda terms in terms of $0-1$-strings. Later, Grygiel and Lescanne conjectured that the number of binary lambda terms with m free indices and of size n (encoded as binary words of length n) is $o\left(n^{-3/2}\tau^{-n}\right)$ for $\tau \approx 1.963448\ldots$. We generalize the proposed notion of size and show that for several classes of lambda terms, including binary lambda terms with m free indices, the number of terms of size n is $\Theta\left(n^{-3/2}\rho^{-n}\right)$ with some class dependent constant ρ, which in particular disproves the above mentioned conjecture. A way to obtain lower and upper bounds for the constant near the leading term is presented and numerical results for a few previously introduced classes of lambda terms are given.

1998 ACM Subject Classification G.2.1 Combinatorics, D.1.6 Logic Programming

Keywords and phrases lambda calculus, terms enumeration, analytic combinatorics

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.40

1 Introduction

The objects of our interest are lambda terms which are a basic object of lambda calculus. A lambda term is a formal expression which is described by the grammar $M ::= x \mid \lambda x.M \mid (M\,N)$ where x is a variable, the operation $(M\,N)$ is called application, and using the quantifier λ is called abstraction. In a term of the form $\lambda x.M$ each occurrence of x in M is called a bound variable. We say that a variable x is free in a term M if it is not in the scope of any abstraction. A term with no free variables is called closed, otherwise open. Two terms are considered equivalent if they are identical up to renaming of the variables, *i.e.*, more formally speaking, they can be transformed into each other by α-conversion.

In this paper we are interested in counting lambda terms whose size corresponds to their De Bruijn representation (*i.e.* nameless expressions in the sense of [3]).

▶ **Definition 1.** A De Bruijn representation is a word described by the following specification:

$$M ::= n \mid \lambda M \mid M\,M$$

* This work was partially supported by FWF grant SFB F50-03.

© Bernhard Gittenberger and Zbigniew Gołębiewski;
licensed under Creative Commons License CC-BY
33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 40; pp. 40:1–40:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

where n is a positive integer, called a De Bruijn index. Each occurrence of a De Bruijn index is called a variable and each λ an abstraction. A variable n of a De Bruijn representation w is bound if the prefix of w which has this variable as its last symbol contains at least n times the symbol λ, otherwise it is free. The abstraction which binds a variable n is the nth λ before the variable when parsing the De Bruijn representation from that variable n backwards to the first symbol.

For the purpose of the analysis we will use the notation consistent with the one used in [1]. This means that the variable n will be represented as a sequence of n symbols, namely as a string of $n-1$ so-called 'successors' S and a so-called 'zero' 0 at the end. Obviously, there is a one to one correspondence between equivalence classes of lambda terms (as described in the first paragraph) and De Bruijn representations. For instance, the De Bruijn representation of the lambda-term $\lambda x.\lambda y.xy$ (which is e.g. equivalent to $\lambda a.\lambda b.ab$ or $\lambda y.\lambda x.yx$) is $\lambda\lambda 21$; using the notation with successors this becomes $\lambda\lambda((S0)0)$.

In this paper we are interested in counting lambda terms of given size where we use a general notion of size which covers several previously studied models from the literature. We count the building blocks of lambda terms, zeros, successors, abstractions and applications, with size a, b, c and d, respectively. Formally, if M and N are lambda terms, then

$$|0| = a, \qquad |Sn| = |n| + b, \qquad |\lambda M| = |M| + c, \qquad |MN| = |M| + |N| + d.$$

Thus we have for the example given above $|\lambda\lambda((S0)0)| = 2a + b + 2c + d$. Assigning sizes for the symbols like above covers several previously introduced notions of size:

- so called 'natural counting' (introduced in [1]) where $a = b = c = d = 1$,
- so called 'less natural counting' (introduced in [1]) where $a = 0, b = c = 1, d = 2$.
- binary lambda calculus (introduced in [8]) where $b = 1, a = c = d = 2$,

▶ **Assumption 1.** *Throughout the paper we will make the following assumptions about the constants a, b, c, d:*

1. *a, b, c, d are nonnegative integers,*
2. *$a + d \geq 1$,*
3. *$b, c \geq 1$,*
4. *$\gcd(b, c, a + d) = 1$.*

If the zeros and the applications both had size 0 (*i.e.* $a + d = 0$), then we would have infinitely many terms of the given size, because one could insert arbitrarily many applications and zeros into a term without increasing its size. If the successors or the abstractions had size 0 (*i.e.* b or c equals to 0), then we would again have infinitely many terms of given size, because one could insert arbitrarily long strings of successors or abstractions into a term without increasing its size. The last assumption is more technical in its nature. It ensures that the generating function associated with the sequence of the number of lambda-terms will have exactly one singularity on the circle of convergence, which is on the positive real line. The case of several singularities is not only technically more complicated, but it is for instance not even *a priori* clear which singularities are important and which are negligible. So we cannot expect that it differs from the single singularity case only by a multiplicative constant.

We mention that in [6] lambda terms with size function corresponding to $a = b = 0$ and $c = d = 1$ were considered, but another restriction was imposed on the terms.

Notations. We introduce some notations which will be frequently used throughout the paper: If p is a polynomial, then RootOf $\{p\}$ will denote the smallest positive root of p. Moreover, we will write $[z^n]f(z)$ for the nth coefficient of the power series expansion of $f(z)$ at $z = 0$ and $f(z) \prec g(z)$ (or $f(z) \preceq g(z)$) to denote that $[z^n]\, f(z) < [z^n]\, g(z)$ (or $[z^n]\, f(z) \leq [z^n]\, g(z)$) for all integers n.

Plan of the Paper. The primary aim of this paper is the asymptotic enumeration of closed lambda terms of given size with the size tending to infinity. In the next section we define several classes of lambda terms as well as the generating function associated with them, present our main results and prove several auxiliary results which will be important in the sequel. We derive the asymptotic equivalent of the number of closed terms of given size up to a constant factor. This is established by construction of upper and lower bounds for the coefficients of the generating functions. These constructions are done in Sections 3 and 4. To get fairly accurate numerical bounds we present a method for improving the previously obtained bounds in Section 5. Finally, Section 6 is devoted to the derivation of very accurate results for classes of lambda terms which have been previously studied in the literature.

2 Main Results

In order to count lambda terms of a given size we set up a formal equation which is then translated into a functional equation for generating functions. For this we will utilise the symbolic method developed in [5].

Let us introduce the following atomic classes: the class of zeros $\mathcal{Z}$, the class of successors $\mathcal{S}$, the class of abstractions $\mathcal{U}$ and the class of applications $\mathcal{A}$. Then the class $\mathcal{L}_\infty$ of lambda terms can be described as follows:

$$\mathcal{L}_\infty = \textsc{Seq}(\mathcal{S}) \times \mathcal{Z} + \mathcal{U} \times \mathcal{L}_\infty + \mathcal{A} \times \mathcal{L}_\infty^2 \tag{1}$$

The number of lambda terms of size n, denoted by $L_{\infty,n}$, is $|\{t \in \mathcal{L}_\infty : |t| = n\}|$. Let $L_\infty(z) = \sum_{n\geq 0} L_{\infty,n} z^n$ be the generating function associated with $\mathcal{L}_\infty$. Then specification (1) gives rise to a functional equation for the generating function $L_\infty(z)$:

$$L_\infty(z) = z^a \sum_{j=0}^{\infty} z^{bj} + z^c L_\infty(z) + z^d L_\infty(z)^2. \tag{2}$$

Solving (2) we get

$$L_\infty(z) = \frac{1 - z^c - \sqrt{(1-z^c)^2 - \frac{4z^{a+d}}{1-z^b}}}{2z^d}, \tag{3}$$

which defines an analytic function in a neighbourhood of $z = 0$.

▶ **Proposition 2.** *Let* $\rho = \text{RootOf}\left\{(1-z^b)(1-z^c)^2 - 4z^{a+d}\right\}$. *Then*

$$L_\infty(z) = a_\infty + b_\infty \left(1 - \frac{z}{\rho}\right)^{\frac{1}{2}} + O\left(\left|1 - \frac{z}{\rho}\right|\right), \tag{4}$$

for some constants $a_\infty > 0, b_\infty < 0$ *that depend on* a, b, c, d.

Proof. Let $f(z) = (1-z^b)(1-z^c)^2 - 4z^{a+d}$. Then ρ is the smallest positive solution of $f(z) = 0$. If we compute derivative $f'(z) = -4(a+b)z^{a+b-1} - 2cz^{c-1}(1-z^b)(1-z^c) - bz^{b-1}(1-z^c)^2$

we can observe that all three terms are negative for $0 < z < 1$. Since $0 < \rho < 1$, the function $f(z)$ does not have a double root at ρ and thus $L_\infty(z)$ has an algebraic singularity of type $\frac{1}{2}$ which means that its Newton-Puiseux expansion is of the form (4).

Since $L_\infty(z)$ is a power series with positive coefficients, we know that $a_\infty = L_\infty(\rho) > 0$ and $b_\infty < 0$. ◀

▶ **Corollary 3.** *The coefficients of $L_\infty(z)$ satisfy $[z^n]L_\infty(z) \sim C\rho^{-n}n^{-3/2}$, as $n \to \infty$, where $C = -b_\infty/(2\sqrt{\pi})$.*

Let us define the class of m-open lambda terms, denoted $\mathcal{L}_m$, as

$$\mathcal{L}_m = \{t \in \mathcal{L}_\infty : \text{a prefix of } m \text{ abstractions } \lambda \text{ makes } t \text{ a closed term}\}.$$

We remark that any m-open lambda term is obviously $m+1$-open as well. The number of m-open lambda terms of size n is denoted by $L_{m,n}$ and the generating function associated with the class by $L_m(z) = \sum_{n\geq 0} L_{m,n}z^n$. Similarly to $\mathcal{L}_\infty$, the class $\mathcal{L}_m$ can be specified, and this specification yields the functional equation

$$L_m(z) = z^a \sum_{j=0}^{m-1} z^{bj} + z^c L_{m+1}(z) + z^d L_m(z)^2 \tag{5}$$

Note that $L_0(z)$ is the generating function of the set $\mathcal{L}_0$ of closed lambda terms.

Let $\mathcal{K}_m = \mathcal{L}_\infty \setminus \mathcal{L}_m$ and $K_m(z) = L_\infty(z) - L_m(z)$. Then using (2) and (5) we obtain

$$K_m(z) = z^a \sum_{j=m}^{\infty} z^{bj} + z^c K_{m+1}(z) + z^d K_m(z)L_\infty(z) + z^d K_m(z)L_m(z). \tag{6}$$

which implies

$$K_m(z) = \frac{z^{a+bm}}{(1-z^b)(1-z^d(L_\infty(z)+L_m(z)))} + \frac{z^c}{1-z^d(L_\infty(z)+L_m(z))}K_{m+1}(z). \tag{7}$$

Note that $K_m(z)$ as well as $L_m(z)$ define analytic functions in a neighbourhood of $z = 0$.

Let us state the main theorem of the paper:

▶ **Theorem 4.** *Let $\rho = \text{RootOf}\left\{(1-z^b)(1-z^c)^2 - 4z^{a+d}\right\}$. Then there exist positive constants $\underline{C}$ and $\overline{C}$ (depending on a, b, c, d and m) such that the number of m-open lambda terms of size n satisfies*

$$\liminf_{n\to\infty} \frac{[z^n]\,L_m(z)}{\underline{C}n^{-\frac{3}{2}}\rho^{-n}} \geq 1 \quad \textit{and} \quad \limsup_{n\to\infty} \frac{[z^n]\,L_m(z)}{\overline{C}n^{-\frac{3}{2}}\rho^{-n}} \leq 1, \tag{8}$$

▶ Remark. In case of given a, b, c, d and m we can compute numerically such constants $\underline{C}$ and $\overline{C}$. This will be done for some of the models mentioned in the introduction.

Before proving this theorem we will present the key ideas needed for our proof. We introduce the class $\mathcal{L}_m^{(h)}$ of lambda terms in $\mathcal{L}_m$ where the length of each string of successors is bounded by a constant integer h. As before, set $L_{m,n}^{(h)} = \left|\left\{t \in \mathcal{L}_m^{(h)} : |t| = n\right\}\right|$ and $L_m^{(h)}(z) = \sum_{n\geq 0} L_{m,n}^{(h)} z^n$. Then $L_m^{(h)}(z)$ satisfies the functional equation

$$L_m^{(h)}(z) = \begin{cases} z^a \sum_{j=0}^{m-1} z^{bj} + z^c L_{m+1}^{(h)}(z) + z^d L_m^{(h)}(z)^2 & \text{if } m < h, \\ z^a \sum_{j=0}^{h-1} z^{bj} + z^c L_h^{(h)}(z) + z^d L_h^{(h)}(z)^2 & \text{if } m \geq h. \end{cases} \tag{9}$$

Notice that for $m \geq h$ we have a quadratic equation for $L_m^{(h)}(z) = L_h^{(h)}(z)$ that has the solution

$$L_h^{(h)}(z) = \frac{1 - z^c - \sqrt{(1-z^c)^2 - 4z^{a+d}\frac{1-z^{bh}}{1-z^b}}}{2z^d}.$$

For $m < h$ we have a relation between $L_m^{(h)}(z)$ and $L_{m+1}^{(h)}(z)$ which gives rise to a representation of $L_m^{(h)}(z)$ in terms of nested radicands (*cf.* [2]) after all. Indeed, for $m < h$ we have

$$L_m^{(h)}(z) = \frac{1 - \sqrt{r_m(z) + 2z^c\sqrt{r_{m+1}(z) + 2z^c\sqrt{\cdots\sqrt{r_{h-1}(z) + 2z^c\sqrt{r_h(z)}}}}}}{2z^d} \tag{10}$$

where

$$r_j(z) = \begin{cases} 1 - 4z^{a+d}\frac{1-z^{jb}}{1-z^b} - 2z^c & \text{if } m \leq j < h-1, \\ 1 - 4z^{a+d}\frac{1-z^{(h-1)b}}{1-z^b} - 2z^c + 2z^{2c} & \text{if } j = h-1, \\ (1-z^c)^2 - 4z^{a+d}\frac{1-z^{bh}}{1-z^b} & \text{if } j = h. \end{cases}$$

▶ **Lemma 5.** *For all $m \geq 0$ the dominant singularity $\rho^{(h)} := \rho_m^{(h)}$ of $L_m^{(h)}(z)$ is independent of m. Moreover, we have $\lim_{h\to\infty} \rho^{(h)} = \rho$.*

Proof. One can check that the dominant singularity of $L_m^{(h)}(z)$ comes from smallest positive root of $r_h(z)$ and that it is of type $\frac{1}{2}$ (*i.e.* $L_m^{(h)}(z) = a_m^{(h)} + b_m^{(h)}\left(1 - \frac{z}{\rho^{(h)}}\right)^{\frac{1}{2}} + O\left(\left|1 - \frac{z}{\rho^{(h)}}\right|\right)$ as $z \to \rho_m^{(h)}$ for some constants $a_m^{(h)}, b_m^{(h)}$ depending on m and h). Consequently, $\rho_m^{(h)}$ is independent of m. Notice that in the unit interval $r_h(z)$ converges uniformly to $r(z)$, the radicand in (3). Thus $\lim_{h\to\infty} \rho^{(h)} = \rho$ since ρ is the smallest positive root of $r(z)$. ◀

Let us begin with computing the radii of convergence of the functions $K_m(z)$ and $L_m(z)$. For the case of binary lambda calculus Lemmas 7 and 8 were already proven in [7]. To extend those results to our more general setting, we will use different techniques.

▶ **Lemma 6.** *For all $m \geq 0$ the radius of convergence of $K_m(z)$ equals ρ (the radius of convergence of $L_\infty(z)$).*

Proof. Inspecting (7) reveals that the key part is $\frac{1}{1-z^d(L_\infty(z)+L_m(z))}$. This is the generating function of a sequence of combinatorial structures associated with the generating function $z^d(L_\infty(z) + L_m(z))$. One can check that we are not in the supercritical sequence schema case (*i.e.* a singularity of considered fraction does not come from the root of its denominator, see [5, pp. 293]) because $1 - \rho^d\left(L_\infty(\rho) + L_m(\rho)\right) > 0$. This follows from

$$\rho^d\left(L_\infty(\rho) + L_m(\rho)\right) \leq 2\rho^d L_\infty(\rho) = 1 - \rho^c < 1.$$

The first inequality holds because $L_\infty(\rho) \geq L_m(\rho)$ for all $m \geq 0$ and the second one because $\rho > 0$. Moreover, the radius of convergence of $L_m(z)$ is larger than or equal to the radius of convergence of $L_\infty(z)$ because $\mathcal{L}_m \subseteq \mathcal{L}_\infty$. Therefore, for all $m \geq 0$ the radius of convergence of $K_m(z)$ equals ρ, the radius of convergence of $L_\infty(z)$. ◀

▶ **Lemma 7.** *All the functions $L_m(z)$, $m \geq 0$, have the same radius of convergence.*

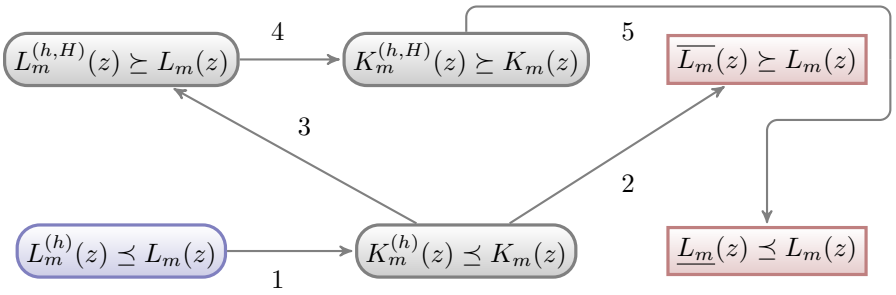


Figure 1 The diagram illustrates the idea how we obtain the upper and the lower bound (in terms of the coefficients) for the function $L_m(z)$. Starting point is denoted by a blue node, the finish nodes are red.

Proof. Let ρ_m denote the radius of convergence of the function $L_m(z)$. From the definition of the function $L_m(z)$ it is known that for all $m \geq 0$ and for all n we have $[z^n]\, L_m(z) \leq [z^n]\, L_{m+1}(z)$ and therefore $\rho_m \geq \rho_{m+1}$. Moreover, from (5) we know

$$L_{m+1}(z) = -z^{a-c} \sum_{j=0}^{m-1} z^{bj} + z^{-c} L_m(z) - z^{d-c} L_m(z)^2.$$

Notice that $\rho_m \leq 1$ because $\rho_m \leq \rho^{(h)} < 1$ for $h \geq m$. Then due to the fact that $z^{-c} L_m(z) - z^{d-c} L_m(z)^2$ has radius of convergence bigger or equal ρ_m, we have $\rho_m \leq \rho_{m+1}$. ◀

▶ **Lemma 8.** *For all $m \geq 0$ the radius of convergence of $L_m(z)$ equals ρ.*

Proof. Take $L_m^{(m)}$, defined in (9). Recall that $\rho^{(m)}, \rho_m$ and ρ denote the radii of convergence of $L_m^{(m)}(z), L_m(z)$ and $L_\infty(z)$, respectively. Notice that for all $m, n \geq 0$ we have $[z^n]\, L_m^{(m)}(z) \leq [z^n]\, L_m(z) \leq [z^n]\, L_\infty(z)$ and thus $\rho^{(m)} \geq \rho_m \geq \rho$. Now, the assertion follows from Lemmas 5 and 7. ◀

In the next sections we will present how to obtain an upper and a lower bound for $[z^n]\, L_m(z)$. The idea is to construct auxiliary functions satisfying certain inequalities and to use them to construct further ones until we have the desired bound. The procedure follows the flowchart depicted in Fig. 1.

3 Upper Bound for $[z^n]\, L_m(z)$

Notice that for all integers h and m we have $\mathcal{L}_m^{(h)} \subset \mathcal{L}_m$. Moreover, for all $m, h \geq 0$ there exists $n_m^{(h)}$ such that $[z^n]\, L_m^{(h)}(z) = [z^n]\, L_m(z)$ if $n < n_m^{(h)}$ and $[z^n]\, L_m^{(h)}(z) \leq [z^n]\, L_m(z)$ else. We will use those properties of $L_m^{(h)}(z)$ in order to derive a lower bound for the asymptotics of $[z^n]\, K_m(z)$.

Note that (6) corresponds to an equation of the form $\mathcal{K}_m = \mathcal{F}(\mathcal{K}_m, \mathcal{K}_{m+1}, \mathcal{L}_\infty, \mathcal{L}_m)$ where $\mathcal{F}$ is obtained from (6) by replacing addition and multiplication by their combinatorial counterparts and generating functions by their corresponding sets. Now, define the new set $\mathcal{K}_m^{(h)} := \mathcal{F}\left(\mathcal{K}_m^{(h)}, \mathcal{K}_{m+1}^{(h)}, \mathcal{L}_\infty, \mathcal{L}_m^{(h)}\right)$. From the construction of $\mathcal{F}$ and the properties of $\mathcal{L}_m^{(h)}$ we know that $\mathcal{K}_m^{(h)} \subseteq \mathcal{K}_m$. Let $K_{m,n}^{(h)} = \left|\left\{t \in \mathcal{K}_m^{(h)} : |t| = n\right\}\right|$ and $K_m^{(h)}(z) = \sum_{n \geq 0} K_{m,n}^{(h)} z^n$.

Then $K_m^{(h)}(z)$ satisfies the functional equation

$$K_m^{(h)}(z) = z^a \sum_{j=m}^{\infty} z^{bj} + z^c K_{m+1}^{(h)}(z) + z^d K_m^{(h)}(z) L_\infty(z) + z^d K_m^{(h)}(z) L_m^{(h)}(z). \tag{11}$$

In fact, what we did is that we replaced in the application operation every m-open lambda term (corresponding to the subterm $z^d K_m(z) L_m(z)$ of (6)) by an m-open lambda term where each string of successors has bounded length (corresponding to $z^d K_m(z) L_m^{(h)}(z)$). Solving (11) we get after some computations

$$K_m^{(h)}(z) = \frac{z^{a-cm}}{1-z^b} \sum_{j=m}^{\infty} z^{j(b+c)} \prod_{i=m}^{j} \frac{1}{1 - z^d \left(L_\infty(z) + L_i^{(h)}(z) \right)} =: S_{m,\infty}(z). \tag{12}$$

▶ **Lemma 9.** *Let ρ, a_∞, b_∞ be as in Proposition 2 and $\tilde{c}_i = 1/(1-\rho^d(a_\infty + L_i^{(h)}(\rho)))$ and $\tilde{d}_i = b_\infty \rho^d / (1-\rho^d(a_\infty + L_i^{(h)}(\rho)))^2$. Then $K_m^{(h)}(z)$ admits the expansion*

$$K_m^{(h)}(z) = c_m^{(h)} + d_m^{(h)} \left(1 - \frac{z}{\rho}\right)^{\frac{1}{2}} + O\left(\left| 1 - \frac{z}{\rho} \right| \right), \text{ as } z \to \rho, \tag{13}$$

where

$$c_m^{(h)} = \begin{cases} S_{m,h-1}(\rho) + R_{c_m^{(h)}} & \text{if } m < h, \\ \frac{\rho^{a+bm}}{(1-\rho^b)\left(1-\rho^{b+c}-\rho^d\left(a_\infty + L_h^{(h)}(\rho)\right)\right)} & \text{else,} \end{cases}$$

$$d_m^{(h)} = \begin{cases} \frac{\rho^{a-cm}}{1-\rho^b} \sum_{j=m}^{h-1} \rho^{j(b+c)} \sum_{i=m}^{j} \frac{\tilde{d}_i}{\tilde{c}_i} \prod_{k=m}^{j} \tilde{c}_k + R_{d_m^{(h)}} & \text{if } m < h, \\ \frac{b_\infty \rho^{a+bm+d}}{(1-\rho^b)\left(1-\rho^{b+c}-\rho^d\left(a_\infty + L_h^{(h)}(\rho)\right)\right)^2} & \text{else,} \end{cases}$$

with

$$R_{c_m^{(h)}} = \frac{\rho^{a+bh+c(h-m)}}{(1-\rho^b)\left(1-\rho^{b+c}-\rho^d\left(a_\infty + L_h^{(h)}(\rho)\right)\right)} \prod_{i=m}^{h-1} \tilde{c}_i,$$

$$R_{d_m^{(h)}} = \frac{b_\infty \rho^{a+bh+c(h-m)+d}}{(1-\rho^b)\left(1-\rho^{b+c}-\rho^d\left(a_\infty + L_h^{(h)}(\rho)\right)\right)} \left(\prod_{i=m}^{h-1} \tilde{c}_i \right) \times \left(\sum_{i=m}^{h-1} \tilde{c}_i + \frac{1}{1-\rho^{b+c}-\rho^d\left(a_\infty + L_h^{(h)}(\rho)\right)} \right).$$

Proof. Let us recall that for all $m \geq h$ we have $L_m^{(h)}(z) = L_h^{(h)}(z)$. Therefore we can split the infinite sum $S_{m,\infty}(z)$ in (12) into the finite one $S_{m,h-1}(z)$ and the rest $S_{h,\infty}(z)$.

Case I: $m < h$**.** First, consider the finite sum $S_{m,h-1}(z)$. As in the proof of Lemma 6 we identify the key term, show that we are not in the supercritical case, and expand by means of Proposition 2. Eventually, this yields

$$\prod_{i=m}^{j} \frac{1}{1 - z^d (L_\infty(z) + L_i^{(h)}(z))} = \tilde{c}_{m,j} + \tilde{d}_{m,j} \left(1 - \frac{z}{\rho}\right)^{\frac{1}{2}} + O\left(\left| 1 - \frac{z}{\rho} \right| \right)$$

where $\tilde{c}_{m,j} = \prod_{i=m}^{j} \tilde{c}_i$ and $\tilde{d}_{m,j} = \sum_{i=m}^{j} \frac{\tilde{d}_i}{\tilde{c}_i} \prod_{k=m}^{j} \tilde{c}_k$. Since $\left(1 - \frac{z}{\rho}\right)^{\frac{1}{2}}$ does neither depend on m nor on j and $\frac{z^{a-cm}}{1-z^b}$ has poles only on the unit circle, for all $m \geq 0$ we have

$$S_{m,h-1}(z) = \tilde{\tilde{c}}_m + \tilde{\tilde{d}}_m \left(1 - \frac{z}{\rho}\right)^{\frac{1}{2}} + O\left(\left|1 - \frac{z}{\rho}\right|\right)$$

where $\tilde{\tilde{c}}_m = \frac{\rho^{a-cm}}{1-\rho^b} \sum_{j=m}^{h-1} \rho^{j(b+c)} \tilde{c}_{m,i}$ and $\tilde{\tilde{d}}_m = \frac{\rho^{a-cm}}{1-\rho^b} \sum_{j=m}^{h-1} \rho^{j(b+c)} \tilde{d}_{m,i}$.

Now, let us look on the infinite part of the sum in (12). We will refer to its contributions to the first and second coefficient of the Newton-Puiseux expansion (13) as the remainders $R_{c_m^{(h)}}$ and $R_{d_m^{(h)}}$, respectively. Since for all $m \geq h$ we have $L_m^{(h)}(z) = L_h^{(h)}(z)$, the sum can be rewritten as

$$S_{h,\infty}(z) = \left(\prod_{i=m}^{h-1} \frac{1}{1 - z^d \left(L_\infty(z) + L_i^{(h)}(z)\right)}\right) \cdot \frac{z^{a+bh+c(h-m)}}{(1-z^b)\left(1 - z^{b+c} - z^d \left(L_\infty(z) + L_h^{(h)}(z)\right)\right)}.$$

We already know how to handle the product part of this expression, so let us consider the fraction $\frac{z^{a+bh+c(h-m)}}{(1-z^b)\left(1-z^{b+c}-z^d\left(L_\infty(z)+L_h^{(h)}(z)\right)\right)}$. Similarly to before, we have to check that the singularity of this function does not come from the root of the denominator but from $L_\infty(z)$ (it cannot come from $L_h^{(h)}$ because it has a bigger radius of convergence than $L_\infty(z)$). So, we have to show the inequality $1 - \rho^{b+c} - \rho^d \left(L_\infty(\rho) + L_h^{(h)}(\rho)\right) > 0$. But from $L_\infty(\rho) \geq L_h^{(h)}(\rho)$ and $0 < \rho^{b+c} < \rho^c < 1$ we obtain $\rho^d(L_\infty(\rho) + L_h^{(h)}(\rho)) \leq 2\rho^d L_\infty(\rho) = 1 - \rho^c < 1 - \rho^{b+c}$ and hence the desired inequality indeed holds. Now, similarly to the previous case we use the Newton-Puiseux expansion of $L_\infty(z)$ at ρ to derive an expansion of the infinite part of the sum in (12) and get the asserted result.

Case II: $m \geq h$**.** This case is easier, because the finite part of the sum in (12) does not exist and the other part can be evaluated to a closed form which can be treated as above. ◀

Using the transfer lemmas of [4] (applied to $K_m^{(h)}(z)$) and $[z^n]\, K_m^{(h)}(z) \leq [z^n]\, K_m(z)$, we get $\liminf_{n\to\infty} ([z^n]\, K_m(z)) \cdot \Gamma(-1/2) n^{3/2} \rho^n / d_m^{(h)} \geq 1$.

▶ **Corollary 10.** *The number of m-open lambda terms of size n satisfies*

$$\limsup_{n\to\infty} \frac{[z^n]\, L_m(z)}{\overline{C} n^{-\frac{3}{2}} \rho^{-n}} \leq 1 \text{ where } \overline{C} = \frac{b_\infty - d_m^{(h)}}{\Gamma\left(-\frac{1}{2}\right)}.$$

4 Lower Bound for $[z^n]\, L_m(z)$

The idea behind obtaining a lower bound for $[z^n]\, L_m(z)$ is similar to the one used for the upper bound. First we will find an upper bound for $[z^n]\, K_m(z)$ using the function

$$L_m^{(h,H)}(z) = \sum_{n\geq 0} L_{m,n}^{(h,H)} z^n = \begin{cases} L_\infty(z) - K_m^{(h)}(z) & \text{if } m < H, \\ L_\infty(z) & \text{else.} \end{cases} \tag{14}$$

Notice that for all $m, h, H, n \geq 0$ we have $[z^n]\, L_m(z) \leq [z^n]\, L_m^{(h,H)}(z)$. Let $\mathcal{L}_m^{(h,H)}$ denote the class of combinatorial structures associated with $L_m^{(h,H)}(z)$ and define the new set $\mathcal{K}_m^{(h,H)} := \mathcal{F}\left(\mathcal{K}_m^{(h,H)}, \mathcal{K}_{m+1}^{(h,H)}, \mathcal{L}_\infty, \mathcal{L}_m^{(h,H)}\right)$. From the construction of $\mathcal{F}$ and the above

properties of $\mathcal{L}_m^{(h,H)}$ we know that $\mathcal{K}_m \subseteq \mathcal{K}_m^{(h,H)}$. Let $K_{m,n}^{(h,H)} = \left|\left\{t \in \mathcal{K}_m^{(h,H)} : |t| = n\right\}\right|$ and $K_m^{(h,H)}(z) = \sum_{n\geq 0} K_{m,n}^{(h,H)} z^n$. Then $K_m^{(h,H)}(z)$ is given by

$$K_m^{(h,H)}(z) = z^a \sum_{j=m}^{\infty} z^{bj} + z^c K_{m+1}^{(h,H)}(z) + z^d K_m^{(h,H)}(z) L_\infty(z) + z^d K_m^{(h,H)}(z) L_m^{(h,H)}(z).$$

Solving this equation and using (14) we get

$$\begin{aligned} K_m^{(h,H)}(z) &= \frac{z^{a-cm}}{1-z^b} \sum_{j=m}^{H-1} z^{j(b+c)} \prod_{i=m}^{j} \frac{1}{1 - z^d \left(L_\infty(z) + L_m^{(h,H)}(z)\right)} \\ &= \frac{z^{a-cm}}{1-z^b} \sum_{j=m}^{H-1} z^{j(b+c)} \prod_{i=m}^{j} \frac{1}{1 - z^d \left(2L_\infty(z) - K_i^{(h)}(z)\right)} + \\ &\quad \frac{z^{a+bH+c(H-m)}}{(1-z^d)\left(1 - z^{b+c} - 2z^d L_\infty(z)\right)} \left(\prod_{i=m}^{H-1} \frac{1}{1 - z^d \left(2L_\infty(z) - K_i^{(h)}(z)\right)} \right). \end{aligned} \tag{15}$$

▶ **Lemma 11.** *Let ρ be the radius of convergence of the function $L_\infty(z)$. Then the generating function $K_m^{(h,H)}(z)$ admits the following expansion*

$$K_m^{(h,H)}(z) = c_m^{(h,H)} + d_m^{(h,H)} \left(1 - \frac{z}{\rho}\right)^{\frac{1}{2}} + O\left(\left|1 - \frac{z}{\rho}\right|\right), \tag{16}$$

where

$$\begin{aligned} c_m^{(h,H)} &= \frac{\rho^{a-cm}}{1-\rho^b} \sum_{j=m}^{H-1} \rho^{j(b+c)} \prod_{i=m}^{j} \frac{1}{1 - \rho^d \left(2a_\infty - c_i^{(h)}\right)} + R_{c_m^{(h,H)}}, \\ d_m^{(h,H)} &= \frac{\rho^{a-cm}}{1-\rho^b} \sum_{j=m}^{H-1} \rho^{j(b+c)} \sum_{i=m}^{j} \frac{\rho^d \left(2b_\infty - d_i^{(h)}\right)}{1 - \rho^d \left(2a_\infty - c_i^{(h)}\right)} \prod_{i=m}^{j} \frac{1}{1 - \rho^d \left(2a_\infty - c_i^{(h)}\right)} + R_{d_m^{(h,H)}}, \end{aligned}$$

a_∞, b_∞ and $c_i^{(h)}, d_i^{(h)}$ come from the expansion of $L_\infty(z)$ and $K_i^{(h)}(z)$, respectively, at ρ (see Proposition 2 and the proof of Lemma 9) and

$$\begin{aligned} R_{c_m^{(h,H)}} &= \frac{\rho^{a+bH+c(H-m)}}{(1-\rho^d)\left(1-\rho^{b+c} - 2\rho^d a_\infty\right)} \left(\prod_{i=m}^{H-1} \frac{1}{1 - \rho^d \left(2a_\infty - c_i^{(h)}\right)} \right), \\ R_{d_m^{(h,H)}} &= \frac{\rho^{a+bH+c(H-m)+d}}{(1-\rho^d)\left(1-\rho^{b+c} - 2\rho^d a_\infty\right)} \left(\prod_{i=m}^{H-1} \frac{1}{1 - \rho^d \left(2a_\infty - c_i^{(h)}\right)} \right) \\ &\quad \times \left(\frac{2b_\infty}{1-\rho^{b+c} - 2\rho^d a_\infty} + \sum_{i=m}^{H-1} \frac{\left(2b_\infty - d_i^{(h)}\right)}{1 - \rho^d \left(2a_\infty - c_i^{(h)}\right)} \right). \end{aligned}$$

As in the previous section, we apply the transfer lemmas of [4] to $K_m^{(h,H)}(z)$ and use $[z^n]\, K_m^{(h,H)}(z) \geq [z^n]\, K_m(z)$ to arrive at the following result:

▶ **Corollary 12.** *The number of m-open lambda terms of size n satisfies*

$$\liminf_{n\to\infty} \frac{[z^n]\, L_m(z)}{\underline{C} n^{-\frac{3}{2}} \rho^{-n}} \geq 1 \text{ where } \underline{C} = \frac{b_\infty - d_m^{(h,H)}}{\Gamma\left(-\frac{1}{2}\right)}.$$

5 Improvement of the Bounds

The bounding functions $\underline{L_m}(z) = L_\infty(z) - K_m^{(h,H)}(z)$ and $\overline{L_m}(z) = L_\infty(z) - K_m^{(h)}(z)$ derived in the previous sections can be used in a straightforward way to compute numerical values for $\underline{C}$ and $\overline{C}$, given concrete values for a, b, c, d. If we choose h and H big enough, then this will give us proper bounds. But in practice they still leave a large gap, at least for values h and H that allow us to perform the computation within a few hours on a standard PC. For instance, in case of the natural counting for $h = H = 15$ we get $\underline{C}^{(nat)} \approx 0.00404525\ldots$ and $\overline{C}^{(nat)} \approx 0.18086721\ldots$.

We show a simple way to improve $\underline{C}$ and $\overline{C}$. Let us introduce the functions $L_{m,M}^{(h)}(z)$ and $L_{m,M}^{(h,H)}(z)$, defined by the following equations:

$$L_{m,M}^{(h)}(z) = \begin{cases} z^a \sum_{j=0}^{m-1} z^{bj} + z^c L_{m+1,M}^{(h)}(z) + z^d L_{m,M}^{(h)}(z)^2 & \text{if } m < M, \\ L_\infty(z) - K_M^{(h)}(z) & \text{if } m = M, \end{cases}$$

$$L_{m,M}^{(h,H)}(z) = \begin{cases} z^a \sum_{j=0}^{m-1} z^{bj} + z^c L_{m+1,M}^{(h,H)}(z) + z^d L_{m,M}^{(h,H)}(z)^2 & \text{if } m < M, \\ L_\infty(z) - K_M^{(h,H)}(z) & \text{if } m = M. \end{cases}$$

These two functions admit a representation in terms of nested radicals which is similar to (10):

$$L_{m,M}^{(h)}(z) = \frac{1}{2z^d} \cdot \left(1 - \sqrt{1 - 4z^{a+d}\frac{1-z^{mb}}{1-z^b} - 2z^c + 2z^c\sqrt{\cdots\sqrt{1 - 4z^{a+d}\frac{1-z^{(M-1)b}}{1-z^b} - 2z^c + 2z^c\sqrt{L_\infty(z) - K_M^{(h)}(z)}}}}\right),$$

$$L_{m,M}^{(h,H)}(z) = \frac{1}{2z^d} \cdot \left(1 - \sqrt{1 - 4z^{a+d}\frac{1-z^{mb}}{1-z^b} - 2z^c + 2z^c\sqrt{\cdots\sqrt{1 - 4z^{a+d}\frac{1-z^{(M-1)b}}{1-z^b} - 2z^c + 2z^c\sqrt{L_\infty(z) - K_M^{(h,H)}(z)}}}}\right).$$

So, in $L_{m,M}^{(h)}(z)$ and $L_{m,M}^{(h,H)}(z)$ we are using exact expressions for $L_m(z)$ up to some constant M and then replace $L_M(z)$ by a function that is its upper and lower bound, respectively. Numerical results for some previously studied notions of size (see Sections 6.1 and 6.2) reveal a significant improvement in closing the gap between the constants $\overline{C}, \underline{C}$ obtained by utilising the functions $L_{m,M}^{(h)}(z), L_{m,M}^{(h,H)}(z)$.

6 Results for Some Previously Introduced Notions of Size

6.1 Natural Counting

▶ **Lemma 13.** *The following bounds hold*

$$\liminf_{n\to\infty} \frac{[z^n]\, L_0^{(nat)}(z)}{\underline{C}^{(nat)} n^{-\frac{3}{2}} \rho^{-n}} \geq 1 \quad \text{and} \quad \limsup_{n\to\infty} \frac{[z^n]\, L_0^{(nat)}(z)}{\overline{C}^{(nat)} n^{-\frac{3}{2}} \rho^{-n}} \leq 1 \tag{17}$$

where $\rho = \mathrm{RootOf}\{-1 + 3x + x^2 + x^3\} \approx 0.295598\ldots$ *and* $\underline{C}^{(nat)}, \overline{C}^{(nat)}$ *are computable constants with numerical values* $\underline{C}^{(nat)} \approx 0.00404525\ldots$ *and* $\overline{C}^{(nat)} \approx 0.18086721\ldots$.

Table 1 Numbers rounded up to 7 digits.

h, H	$c_0^{(h)}$	$d_0^{(h)}$	$c_0^{(h,H)}$	$d_0^{(h,H)}$
1	0.855448	−1.153959	1.086200	−3.803686
2	0.898032	−1.313246	0.979519	−2.581823
3	0.917305	−1.397536	0.958215	−2.324953
4	0.927248	−1.444672	0.950295	−2.236290
5	0.932849	−1.472308	0.946185	−2.192353
6	0.936128	−1.488826	0.943824	−2.167379
7	0.938055	−1.498647	0.942443	−2.152790
8	0.939174	−1.504385	0.941643	−2.144335
9	0.939813	−1.507673	0.941187	−2.139511
10	0.940172	−1.509525	0.940931	−2.136799
11	0.940372	−1.510556	0.940788	−2.135291
12	0.940482	−1.511125	0.940710	−2.134460
13	0.940543	−1.511438	0.940667	−2.134004
14	0.940576	−1.511608	0.940643	−2.133755
15	0.940594	−1.511701	0.940630	−2.133619

Proof. For 'natural counting' we have $a = b = c = d = 1$. From Theorem 4 we know that the radius of convergence of $L_0^{(nat)}(z)$ equals $\rho = \text{RootOf}\left\{-1 + 3x + x^2 + x^3\right\} \approx 0.295598\ldots$. We can also easily get $L_\infty(z) \sim a_\infty + b_\infty\sqrt{1 - z/\rho}$ with

$$a_\infty = \frac{1-\rho}{2\rho} \approx 1.19149\ldots \text{ and } b_\infty = \frac{1}{\rho - 1}\sqrt{\frac{1+\rho+\rho^2-\rho^3}{2\rho}} \approx 2.15093\ldots,$$

and from the transfer lemmas of [4] we obtain $[z^n]\, L_\infty(z) \sim b_\infty n^{-3/2}\rho^{-n}/\Gamma(-1/2) \simeq (0.606767\ldots)\cdot n^{-\frac{3}{2}}(3.38298\ldots)^n$, as $n \to \infty$.

Since we are most interested in the enumeration of closed lambda terms, we examine the multiplicative constants in the leading term of the asymptotical lower and upper bound for $[z^n]\, L_0(z)$. From the formulas in Lemmas 9 and 11 we have computed the values for $c_0^{(h)}, d_0^{(h)}$ and $c_0^{(h,H)}, d_0^{(h,H)}$ for different constants h and H (see Table 1).

As expected, the bigger h and H are, the more accurate is the bound we get (in this case for $[z^n]\, K_0(z)$). Taking the values of $d_0^{(h,H)}$ and $d_0^{(h)}$ for $h = H = 15$, Corollaries 12 and 10 yield $\underline{C}^{(nat)} \approx 0.00404525\ldots$ and $\overline{C}^{(nat)} \approx 0.18086721\ldots$. Notice that using the values of $d_0^{(h,H)}$ to compute $\underline{C}^{(nat)}$ gives non-trivial values only for $h > 7$ (for $1 \le h \le 7$ we get negative numbers because in this case we have $\left|d_0^{(h,H)}\right| > |b_\infty|$). ◀

Figure 2 illustrates the bounds we obtained and the exact values of the coefficients $[z^n]\, L_0^{(nat)}(z)$ for $10 \le n \le 150$. Applying the approach discussed in Section 5 for $h = H = M = 13$ we get the following improvement.

▶ **Lemma 14.** *The following bounds hold*

$$\liminf_{n\to\infty} \frac{[z^n]\, L_0^{(nat)}(z)}{\underline{\underline{C}}^{(nat)} n^{-\frac{3}{2}}\rho^{-n}} \ge 1 \quad \textit{and} \quad \limsup_{n\to\infty} \frac{[z^n]\, L_0^{(nat)}(z)}{\overline{\overline{C}}^{(nat)} n^{-\frac{3}{2}}\rho^{-n}} \le 1 \tag{18}$$

where $\rho = \text{RootOf}\{-1 + 3x + x^2 + x^3\} \approx 0.295598\ldots$ *and* $\underline{\underline{C}}^{(nat)}, \overline{\overline{C}}^{(nat)}$ *are computable constants with numerical values* $\underline{\underline{C}}^{(nat)} \approx 0.07790995266\ldots$ *and* $\overline{\overline{C}}^{(nat)} \approx 0.07790998229\ldots$.

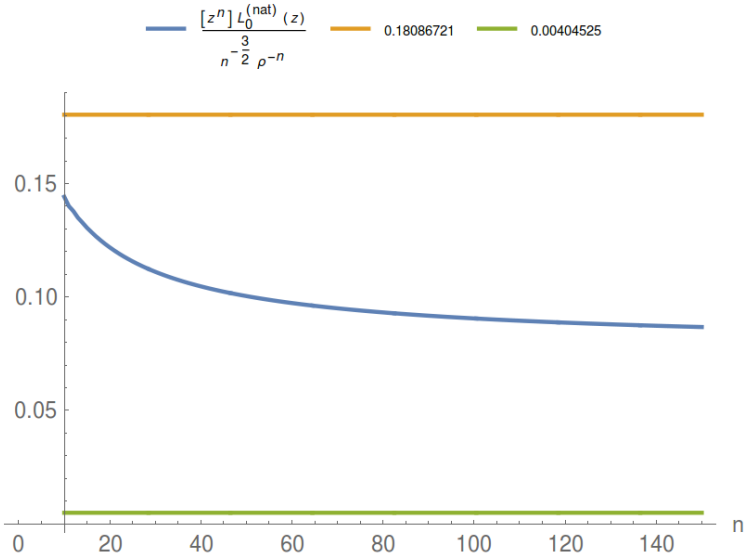


Figure 2 Plot of the exact value of $\frac{[z^n]L_0^{(nat)}(z)}{n^{-\frac{3}{2}}\rho^{-n}}$ and computed lower and upper bound for $h = H = 15$ and $10 \leq n \leq 150$.

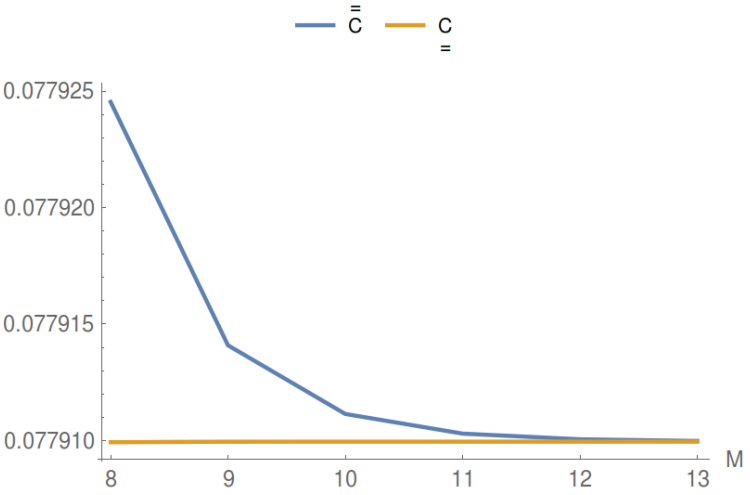


Figure 3 Plot of the numerical values for the constants $\underline{\underline{C}}^{(nat)}, \overline{\overline{C}}^{(nat)}$ for $8 \leq h = H = M \leq 13$.

Figure 3 illustrates how the improvement discussed in Section 5 allows to reduce the gap between the constants $\underline{\underline{C}}^{(nat)}, \overline{\overline{C}}^{(nat)}$ for the lower and the upper bound.

6.2 Binary Lambda Calculus

▶ **Lemma 15.** *The following bounds hold*

$$\liminf_{n\to\infty} \frac{[z^n]\, L_0^{(bin)}(z)}{\underline{\underline{C}}^{(bin)} n^{-\frac{3}{2}} \rho^{-n}} \geq 1 \quad \textit{and} \quad \limsup_{n\to\infty} \frac{[z^n]\, L_0^{(bin)}(z)}{\overline{\overline{C}}^{(bin)} n^{-\frac{3}{2}} \rho^{-n}} \leq 1 \tag{19}$$

where $\rho = \mathrm{RootOf}\{-1+x+2x^2-2x^3+3x^4+x^5\} \approx 0.509308\ldots$ *and* $\underline{\underline{C}}^{(bin)}, \overline{\overline{C}}^{(bin)}$ *are computable constants with numerical values* $\underline{\underline{C}}^{(bin)} \approx 0.01252417\ldots$ *and* $\overline{\overline{C}}^{(bin)} \approx 0.01254593\ldots$.

In order to prove this lemma it is enough to recall that in case of binary lambda calculus the size defining constants are $b = 1$ and $a = c = d = 2$. Then we used the functions $L_{0,13}^{13,13}(z), L_{0,13}^{13,13}(z)$ to obtain the numerical constants stated in Lemma 15.

References

1 Maciej Bendkowski, Katarzyna Grygiel, Pierre Lescanne, and Marek Zaionc. A natural counting of lambda terms. *CoRR*, abs/1506.02367, 2015. URL: `http://arxiv.org/abs/1506.02367`.

2 Olivier Bodini, Danièle Gardy, and Bernhard Gittenberger. Lambda-terms of bounded unary height. In Philippe Flajolet and Daniel Panario, editors, *Proceedings of the Eighth Workshop on Analytic Algorithmics and Combinatorics, ANALCO 2011, San Francisco, California, USA, January 22, 2011*, pages 23–32. SIAM, 2011. `doi:10.1137/1.9781611973013.3`.

3 N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Nederl. Akad. Wetensch. Proc. Ser. A* **75**=*Indag. Math.*, 34:381–392, 1972.

4 Philippe Flajolet and Andrew M. Odlyzko. Singularity analysis of generating functions. *SIAM J. Discrete Math.*, 3(2):216–240, 1990. `doi:10.1137/0403019`.

5 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, New York, NY, USA, 1 edition, 2009.

6 Katarzyna Grygiel and Pierre Lescanne. Counting and generating lambda terms. *Journal of Functional Programming*, 23:594–628, 2013. `doi:10.1017/S0956796813000178`.

7 Katarzyna Grygiel and Pierre Lescanne. Counting terms in the binary lambda calculus. In *DMTCS. 25th International Conference on Probabilistic, Combinatorial and Asymptotic Methods for the Analysis of Algorithms*. Discrete Mathematics & Theoretical Computer Science, Jun 2014.

8 John Tromp. Binary lambda calculus and combinatory logic. In Marcus Hutter, Wolfgang Merkle, and Paul M.B. Vitanyi, editors, *Kolmogorov Complexity and Applications*, number 06051 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. URL: `http://drops.dagstuhl.de/opus/volltexte/2006/628`.

Tightening the Complexity of Equivalence Problems for Commutative Grammars*

Christoph Haase[1] and Piotr Hofman[2]

1 **Laboratoire Spécification et Vérification (LSV), CNRS & ENS Cachan Université Paris-Saclay, France**
haase@lsv.ens-cachan.fr

2 **Laboratoire Spécification et Vérification (LSV), CNRS & ENS Cachan Université Paris-Saclay, France**
hofman@lsv.ens-cachan.fr

Abstract

Given two finite-state automata, are the Parikh images of the languages they generate equivalent? This problem was shown decidable in coNEXP by Huynh in 1985 within the more general setting of context-free commutative grammars. Huynh conjectured that a Π_2^{P} upper bound might be possible, and Kopczyński and To established in 2010 such an upper bound when the size of the alphabet is fixed. The contribution of this paper is to show that the language equivalence problem for regular and context-free commutative grammars is actually coNEXP-complete. In addition, our lower bound immediately yields further coNEXP-completeness results for equivalence problems for regular commutative expressions, reversal-bounded counter automata and communication-free Petri nets. Finally, we improve both lower and upper bounds for language equivalence for exponent-sensitive commutative grammars.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases language equivalence, commutative grammars, presburger arithmetic, semi-linear sets, petri nets

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.41

1 Introduction

Language equivalence is one of the most fundamental decision problems in formal language theory. Classical results include PSPACE-completeness of deciding language equivalence for regular languages generated by non-deterministic finite-state automata (NFA) [4, p. 265], and the undecidability of language equivalence for languages generated by context-free grammars [12, p. 318].

Equivalence problems for formal languages which are undecidable over the free monoid may become decidable in the commutative setting. The problem then is to decide whether the Parikh images of two languages coincide. Given a word w over an alphabet Σ consisting of m alphabet symbols, the Parikh image of w is a vector in $\mathbb{N}^m$ counting in its i-th component how often the i-th alphabet symbol occurs in w. This definition can then be lifted to languages, and the Parikh image of a language consequently becomes a subset of $\mathbb{N}^m$, or, equivalently, a subset of $\Sigma^{\odot}$, the free commutative monoid generated by Σ. Parikh's theorem states that Parikh images of context-free languages are semi-linear sets. Since the latter are closed

* Supported by Labex Digicosme, Univ. Paris-Saclay, project VERICONISS.

© Christoph Haase and Piotr Hofman;
licensed under Creative Commons License CC-BY

under all Boolean operations [5], deciding equivalence between Parikh images of context-free languages is decidable.

When dealing with Parikh images of formal languages, it is technically more convenient to directly work with commutative grammars, which were introduced by Huynh in his seminal paper [13] and are "generating devices for commutative languages [that] use [the] free commutative monoid instead of [the] free monoid." In [13], Huynh studied the uniform word problem for various classes of commutative grammars; the complexity of equivalence problems for commutative grammars was subsequently investigated in a follow-up paper [14]. One of the main results in [14] is that the equivalence problem for regular and context-free commutative grammars is Π_2^{P}-hard and in coNEXP. Huynh remarks that a better upper bound might be possible and states as an open problem the question whether the equivalence problem for context-free commutative grammars is Π_2^{P}-complete [14, p. 117]. Some progress towards answering this question was made by Kopczyński and To, who showed that inclusion and *a fortiori* equivalence for regular and context-free commutative grammars are coNP-complete respectively Π_2^{P}-complete when the size of the alphabet is fixed [18, 17]. One of the main contributions of this paper is to answer Huynh's question negatively: we show that already for regular commutative grammars the equivalence problem is coNEXP-complete.

Our coNEXP lower bound is established by showing how to reduce validity in the coNEXP-complete Π_2-fragment of Presburger arithmetic [7, 8] (i.e. its $\forall^*\exists^*$-fragment) to language inclusion for regular commutative grammars. A reduction from this fragment of Presburger arithmetic has recently been used in [9] in order to show coNEXP-completeness of inclusion for integer vector addition systems with states ($\mathbb{Z}$-VASS), and this reduction is our starting point. Similarly to the standard definition of vector addition systems with states, $\mathbb{Z}$-VASS comprise a finite-state controller with a finite number of counters which, however, range over the integers. Consequently, counters can be incremented and decremented, may drop below zero, and the order in which transitions in $\mathbb{Z}$-VASS are taken may commute along a run—those properties are crucial to the hardness proof in [9]. The corresponding situation is different and technically challenging for regular commutative grammars. In particular, alphabet symbols can only be produced but not deleted, and, informally speaking, we cannot produce negative quantities of alphabet symbols.

A further contribution of our paper is to establish a new upper bound for the equivalence problem for exponent-sensitive commutative grammars, a generalisation of context-free commutative grammars where the left-hand sides of productions may contain an arbitrary number of some non-terminal symbol. Exponent-sensitive commutative grammars were recently introduced by Mayr and Weihmann in [21], who showed PSPACE-completeness of the word problem and membership in 2-EXPSPACE of the equivalence problem. Our hardness result implies that the equivalence problem is coNEXP-hard, and we also improve the 2-EXPSPACE-upper bound to co-2NEXP.

Finally, commutative grammars are closely related to Petri nets, cf. [13, 3, 27, 23]. We also discuss implications of our results to equivalence problems for various classes of Petri nets as well as regular commutative expressions [2] and reversal-bounded counter automata [16].

Due to space constraints, we only sketch some of the proofs in this paper. Full proofs can, however, be found in a technical report accompanying this paper which can be obtained from the authors' homepages.[1]

[1] http://www.mimuw.edu.pl/~ph209519/Papers/Stacs2016.pdf

2 Preliminaries

2.1 Commutative Grammars

Let $\Sigma = \{a_1, \ldots, a_m\}$ be a finite alphabet. The free monoid generated by Σ is denoted by Σ^*, and we denote by $\Sigma^\odot$ the free commutative monoid generated by Σ. We interchangeably use different equivalent ways in order to represent a word $w \in \Sigma^\odot$. For $1 \leq j \leq m$, let i_j be the number of times a_j occurs in w, we equivalently write w as $w = a_1^{i_1} a_2^{i_2} \cdots a_m^{i_m}$, $w = (i_1, i_2, \ldots, i_m) \in \mathbb{N}^m$ or $w : \Sigma \to \mathbb{N}$ with $w(a_j) = i_j$, whatever is most convenient. By $|w| = \sum_{1 \leq j \leq m} i_j$ we denote the length of w, and the representation size $\#w$ of w is $\sum_{1 \leq j \leq m} \lceil \log i_j \rceil$. Given $v, w \in \Sigma^\odot$, we sometimes write $v + w$ in order to denote the concatenation $v \cdot w$ of v and w. The empty word is denoted by ϵ, and as usual $\Sigma^+ \stackrel{\text{def}}{=} \Sigma^* \setminus \{\epsilon\}$ is the free semi-group and $\Sigma^\oplus \stackrel{\text{def}}{=} \Sigma^\odot \setminus \{\epsilon\}$ the free commutative semi-group generated by Σ. For $\Gamma \subseteq \Sigma$, $\pi_\Gamma(w)$ denotes the projection of w onto alphabet symbols from Γ.

A commutative grammar (sometimes just grammar in the following) is a tuple $G = (N, \Sigma, S, P)$, where

- N is the finite set of non-terminal symbols;
- Σ is a finite alphabet, the set of terminal symbols, such that $N \cap \Sigma = \emptyset$;
- $S \in N$ is the axiom; and
- $P \subseteq N^\oplus \times (N \cup \Sigma)^\odot$ is a finite set of productions.

The size of G, denoted by $\#G$, is defined as

$$\#G \stackrel{\text{def}}{=} |N| + |\Sigma| + \sum_{(V,W) \in P} |V| + |W|.$$

Note that commutative words in G are encoded in unary. Unless stated otherwise, we use this definition of the size of a commutative grammar in this paper.

Subsequently, we write $V \to W$ whenever $(V, W) \in P$. Let $D, E \in (N \cup \Sigma)^\odot$, we say D directly generates E, written $D \Rightarrow_G E$, iff there are $F \in (N \cup \Sigma)^\odot$ and $V \to W \in P$ such that $D = V + F$ and $E = F + W$. We write $\Rightarrow_G^*$ to denote the reflexive transitive closure of $\Rightarrow_G$, and if $U \Rightarrow_G^* V$ we say that U generates V. If G is clear from the context, we omit the subscript G. For $U \in N^\oplus$, the reachability set $\mathcal{R}(G, U)$ and the language $\mathcal{L}(G, U)$ generated by G starting at U are defined as

$$\mathcal{R}(G, U) \stackrel{\text{def}}{=} \{W \in (N \cup \Sigma)^\odot : U \Rightarrow^* W\} \qquad \mathcal{L}(G, U) \stackrel{\text{def}}{=} \mathcal{R}(G, U) \cap \Sigma^\odot.$$

The reachability set $\mathcal{R}(G)$ and the language $\mathcal{L}(G)$ of G are then defined as $\mathcal{R}(G) \stackrel{\text{def}}{=} \mathcal{R}(G, S)$ and $\mathcal{L}(G) \stackrel{\text{def}}{=} \mathcal{L}(G, S)$. The word problem is, given a commutative grammar G and $w \in \Sigma^\odot$, is $w \in \mathcal{L}(G)$? The main focus of our paper is on the complexity of deciding language inclusion and equivalence for commutative grammars: Given commutative grammars G, H, language inclusion is to decide $\mathcal{L}(G) \subseteq \mathcal{L}(H)$, and language equivalence is to decide $\mathcal{L}(G) = \mathcal{L}(H)$. Since our grammars admit non-determinism, language inclusion and equivalence are logarithmic-space inter-reducible.

By imposing restrictions on the set of productions, we obtain various classes of commutative grammars. Following [13, 21], given $G = (N, \Sigma, S, P)$, we say that G is

- of *type*-0 if there are no restrictions on P;
- *context-sensitive* if $|W| \geq |V|$ for each $V \to W \in P$;
- *exponent-sensitive* if $V \in \{\{U\}^\oplus : U \in N\}$ for each $V \to W \in P$;
- *context-free* if $V \in N$ for each $V \to W \in P$;
- *regular* if $V \in N$ and $W \in (N \cup \{\epsilon\}) \cdot \Sigma^\odot$ for each $V \to W \in P$.

Equivalence problems for commutative grammars were studied by Huynh, who showed that it is undecidable for context-sensitive and hence type-0 grammars, and Π_2^{P}-hard and in coNEXP for regular and context-free commutative grammars [14]. The main contribution of this paper is to prove the following theorem.

▶ **Theorem 1.** *The language equivalence problem for regular and context-free commutative grammars problem is* coNEXP*-complete.*

Exponent-sensitive grammars were only recently introduced by Mayr and Weihmann [21]. They showed that the word problem is PSPACE-hard, and that language equivalence is PSPACE-hard and in 2-EXPSPACE. The lower bounds require commutative words on the left-hand sides of productions to be encoded in binary. The second main contribution of our paper is to improve those results as follows.

▶ **Theorem 2.** *The language equivalence problem for exponent-sensitive commutative grammars is* coNEXP*-hard and in* co-2NEXP*.*

2.2 Presburger Arithmetic, Linear Diophantine Inequalities and Semi-Linear Sets

Let $\boldsymbol{u} = (u_1, \ldots, u_m)$, $\boldsymbol{v} = (v_1, \ldots, v_m) \in \mathbb{Z}^m$, the sum of $\boldsymbol{u}$ and $\boldsymbol{v}$ is defined component-wise, i.e., $\boldsymbol{u} + \boldsymbol{v} = (u_1 + v_1, \ldots, u_m + v_m)$. Given $u \in \mathbb{Z}$, $\hat{\boldsymbol{u}}$ denotes the vector consisting of u in every component and any appropriate dimension. Let $1 \leq i \leq j \leq m$, we define $\pi_{[i,j]}(\boldsymbol{u}) \stackrel{\text{def}}{=} (u_i, \ldots, u_j)$. By $\|\boldsymbol{u}\|_\infty$ we denote the maximum norm of $\boldsymbol{u}$, i.e., $\|\boldsymbol{u}\|_\infty \stackrel{\text{def}}{=} \max\{|u_i| : 1 \leq i \leq n\}$. Let $M, N \subseteq \mathbb{Z}^m$ and $k \in \mathbb{Z}$, as usual $M + N$ is defined as $\{\boldsymbol{m} + \boldsymbol{n} : \boldsymbol{m} \in M,\ \boldsymbol{n} \in N\}$ and $k \cdot M \stackrel{\text{def}}{=} \{k \cdot \boldsymbol{m} : \boldsymbol{m} \in M\}$. Moreover, $\|M\|_\infty \stackrel{\text{def}}{=} \max\{\|\boldsymbol{z}\|_\infty : \boldsymbol{z} \in M\}$. The size $\#\boldsymbol{u}$ of $\boldsymbol{u}$ is $\#\boldsymbol{u} \stackrel{\text{def}}{=} \sum_{1 \leq i \leq m} \lceil \log |u_i| \rceil$, i.e., numbers are encoded in binary, and the size of M is $\#M \stackrel{\text{def}}{=} \sum_{\boldsymbol{u} \in M} \#\boldsymbol{u}$. For an $m \times n$ matrix A consisting of elements $a_{ij} \in \mathbb{Z}$, $\|A\|_{1,\infty} \stackrel{\text{def}}{=} \max\{\sum_{1 \leq j \leq n} |a_{ij}| : 1 \leq i \leq m\}$.

Presburger arithmetic is the is the first-order theory of the structure $\langle \mathbb{N}, 0, 1, +, \geq \rangle$. In this paper, atomic formulas of Presburger arithmetic are linear Diophantine inequalities of the form

$$\sum_{1 \leq i \leq n} a_i \cdot x_i \geq z_i,$$

where $a_i, z_i \in \mathbb{Z}$ and the x_i are first-order variables. Formulas of Presburger arithmetic can then be obtained in the usual way via positive Boolean combinations of atomic formulas and existential and universal quantification over first-order variables, i.e., according to the following grammar:

$$\phi ::= \forall \boldsymbol{x}.\phi \mid \exists \boldsymbol{x}.\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid t$$

Here, the $\boldsymbol{x}$ range over tuples of first-order variables, and t ranges over linear Diophantine inequalities as above. We assume that formulas of Presburger arithmetic are represented as a syntax tree, with no sharing of sub-formulas.

Given a formula ϕ of Presburger arithmetic with no free variables, validity is to decide whether ϕ holds with respect to the standard interpretation in arithmetic. By $\|\phi\|_\infty$ we denote the largest constant occurring in ϕ, and $|\phi|$ is the length of ϕ, i.e., the number of symbols required to write down ϕ, where constants are represented in unary. In analogy to matrices,

we define $\|\phi\|_{1,\infty} \stackrel{\text{def}}{=} \|\phi\|_\infty \cdot |\phi|$. Let $\psi(\boldsymbol{x})$ be a quantifier-free formula open in $\boldsymbol{x} = (x_1, \dots, x_m)$ and $\boldsymbol{x}^* = (x_1^*, \dots, x_m^*) \in \mathbb{N}^m$, we denote by $\psi[\boldsymbol{x}^*/\boldsymbol{x}]$ the formula obtained from ψ by replacing every x_i in ψ by x_i^*. Finally, given a quantifier-free Presburger formula ψ containing linear Diophantine inequalities $t_1, \dots, t_k$ and $b_1, \dots, b_k \in \{0, 1\}$, $\psi[b_1/t_1, \dots, b_k/t_k]$ denotes the Boolean formula obtained from ψ by replacing every t_i with b_i.

In this paper, we are in particular interested in the Π_2-fragment of Presburger arithmetic, i.e. the fragment in which formulas are restricted to a form $\phi = \forall \boldsymbol{x}.\exists \boldsymbol{y}.\psi(\boldsymbol{x}, \boldsymbol{y})$ where $\psi(\boldsymbol{x}, \boldsymbol{y})$ is quantifier free, for which the following is known.

▶ **Theorem 3** ([7, 8]). *Validity in the Π_2-fragment of Presburger arithmetic is* coNEXP*-complete (with hardness under polynomial-time many-one reductions).*

The sets of natural numbers definable in Presburger arithmetic are semi-linear sets [6]. Let $\boldsymbol{b} \in \mathbb{N}^m$ and $P = \{\boldsymbol{p}_1, \dots, \boldsymbol{p}_n\}$ be a finite subset of $\mathbb{N}^m$, define

$$\operatorname{cone}(P) \stackrel{\text{def}}{=} \{\lambda_1 \cdot \boldsymbol{p}_1 + \cdots + \lambda_n \cdot \boldsymbol{p}_n : \lambda_i \in \mathbb{N},\ 1 \leq i \leq n\}.$$

A linear set $L(\boldsymbol{b}, P)$ with base $\boldsymbol{b}$ and periods P is defined as $L(\boldsymbol{b}, P) \stackrel{\text{def}}{=} \boldsymbol{b} + \operatorname{cone}(P)$. A semi-linear set is a finite union of linear sets. For convenience, given a finite subset B of $\mathbb{N}^m$, we define $L(B, P) \stackrel{\text{def}}{=} \bigcup_{\boldsymbol{b} \in B} L(\boldsymbol{b}, P)$. The size of a semi-linear set $M = \bigcup_{i \in I} L(B_i, P_i) \subseteq \mathbb{N}^m$ is defined as

$$\#M \stackrel{\text{def}}{=} \sum_{i \in I} \#B_i + |B_i| \cdot \#P_i.$$

In particular, numbers are encoded in binary. Given a semi-linear set $N \subseteq \mathbb{N}^m$, $\#N$ is the minimum over the sizes of all semi-linear sets $M = \bigcup_{i \in I} L(\boldsymbol{b}_i, P_i)$ such that $N = M$.

A system of linear Diophantine inequalities D is a conjunction of linear inequalities over the same first-order variables $\boldsymbol{x} = (x_1, \dots, x_n)$, which we write in the standard way as $D : A \cdot \boldsymbol{x} \geq \boldsymbol{c}$, where A is a $m \times n$ integer matrix and $\boldsymbol{c} \in \mathbb{N}^m$. The size $\#D$ of D is the number of symbols required to write down D, where we assume binary encoding of numbers. The set of solutions of D is denoted by $[\![D]\!] \subseteq \mathbb{N}^n$. We say that D is feasible if $[\![D]\!] \neq \emptyset$. In [24, 1], bounds on the semi-linear representation of $[\![D]\!]$ are established. The following theorem is a consequence of Corollary 1 in [24] and Theorem 5 in [1].

▶ **Theorem 4.** *Let $D : A \cdot \boldsymbol{x} \geq \boldsymbol{c}$ be a system of linear Diophantine inequalities such that A is an $m \times n$ matrix. Then $[\![D]\!] = L(B, P)$ for $B, P \subseteq \mathbb{N}^n$ such that $|P| \leq \binom{m+n}{m}$ and*

$$\|B\|_\infty, \|P\|_\infty \leq (\|A\|_{1,\infty} + \|\boldsymbol{c}\|_\infty + 2)^{m+n}.$$

3 Lower Bounds

In this section, we establish the coNEXP-lower bound of Theorems 1 and 2. For the sake of a clear presentation, we will first describe the reduction for context-free commutative grammars, and then outline how the approach can be adapted to regular commutative grammars.

As stated in the introduction, we reduce from validity in the Π_2-fragment of Presburger arithmetic. To this end, let $\phi = \forall \boldsymbol{x}.\exists \boldsymbol{y}.\psi(\boldsymbol{x}, \boldsymbol{y})$ such that $\boldsymbol{x} = (x_1, \dots, x_m)$, $\boldsymbol{y} = (y_1, \dots, y_n)$, and ψ is a positive Boolean combination of atomic formulas $t_1, \dots, t_k$. For our reduction, we write atomic formulas of ψ as

$$t_i : \sum_{1 \leq j \leq m} (a_{i,j}^+ - a_{i,j}^-) \cdot x_j + z_i^+ - z_i^- \geq \sum_{1 \leq j \leq n} (b_{i,j}^+ - b_{i,j}^-) \cdot y_j, \tag{1}$$

where the $a^+_{i,j}, a^-_{i,j} \in \mathbb{N}$ are such that $a^+_{i,j} = 0$ or $a^-_{i,j} = 0$, and likewise the $b^+_{i,j}, b^-_{i,j} \in \mathbb{N}$ are such that $b^+_{i,j} = 0$ or $b^-_{i,j} = 0$, and the $z^+_i, z^-_i \in \mathbb{N}$ such that $z^+_i = 0$ or $z^-_i = 0$. Moreover, in the following we set $a_{i,j} \stackrel{\text{def}}{=} a^+_{i,j} - a^-_{i,j}$, $b_{i,j} \stackrel{\text{def}}{=} b^+_{i,j} - b^-_{i,j}$ and $z_i \stackrel{\text{def}}{=} z^+_i - z^-_i$.

▶ **Example 5.** Let $\phi = \forall x.\exists y.\psi(x,y)$ with $\psi(x,y) = (t_1 \wedge t_2) \vee (t_3 \wedge t_4)$ and

$$t_1 = x \geq 2 \cdot y \qquad t_2 = -x \geq -2 \cdot y \qquad t_3 = x + 1 \geq 2 \cdot y \qquad t_4 = -x - 1 \geq -2 \cdot y,$$

which expresses that every natural number is either even or odd. Here, for instance, $a^+_{2,1} = 0$, $a^-_{2,1} = 1$, $z^+_1 = z^-_1 = 0$, $b^+_{2,1} = 0$ and $b^-_{2,1} = 2$. Hence $a_{2,1} = -1$, $z_2 = 0$ and $b_{2,1} = -2$.

With no loss of generality and due to unary encoding of numbers in ϕ, we may assume that the following inequalities hold:

$$|\phi| \geq 2 + m + n + k \qquad\qquad |\phi| \geq \|\phi\|_\infty \tag{2}$$

We furthermore define a constant $c \in \mathbb{N}$, whose bit representation is polynomial in $|\phi|$, as

$$c \stackrel{\text{def}}{=} \min\{2^n : n \in \mathbb{N},\ 2^n \geq |\phi|^{3\cdot|\phi|+2} \cdot 2^{|\phi|}\}. \tag{3}$$

Let $\Sigma \stackrel{\text{def}}{=} \{t^+_1, t^-_1, \ldots, t^+_k, t^-_k\}$, we now show how to construct in logarithmic space context-free commutative grammars G, H over Σ such that $\mathcal{L}(G) \subseteq \mathcal{L}(H)$ iff ϕ is valid. The underlying idea is as follows: the language of G consists of all possible values of the left-hand sides of the inequalities t_i for every choice of $\boldsymbol{x}$, where the value of some t_i is represented by a word $w \in \Sigma^\odot$ via the difference $w(t^+_i) - w(t^-_i)$. For every $w \in \Sigma^\odot$ and $1 \leq i \leq k$, we misuse notation and define $w(t_i) \stackrel{\text{def}}{=} w(t^+_i) - w(t^-_i) \in \mathbb{Z}$; note that in particular $t_i \notin \Sigma$. The grammar H can then be defined in an analogous way and produces the values of the right-hand sides of H for a choice of $\boldsymbol{y}$, but can in addition simulate the Boolean structure of ψ in order to tweak those t_i for which, informally speaking, it cannot obtain a good value.

Before we define G, we remark that in context-free commutative grammars we can assume commutative words to be encoded in binary. This is not possible in regular grammars.

▶ Remark. For any class of commutative grammars containing context-free commutative grammars, it is with no loss of generality possible to assume binary encoding of commutative words, which has, for instance, been observed in [26]. For example, given a production $V \to a^{2^n}$, $n > 0$, we can introduce fresh non-terminal symbols $V_1, \ldots, V_n$ and replace $V \to a^{2^n}$ by $V \to V_1V_1$, $V_n \to a$ and $V_i \to V_{i+1}V_{i+1}$ for every $1 \leq i < n$. Clearly, the grammar obtained by this procedure generates the same language and only results in a sub-quadratic blow-up of the size of the resulting grammar.

Recall that we may represent commutative words of $\Sigma^\odot$ as vectors of natural numbers. We define:

$$u \stackrel{\text{def}}{=} (z^+_1, z^-_1, \ldots, z^+_k, z^-_k) \in \Sigma^\odot \qquad v_i \stackrel{\text{def}}{=} (a^+_{1,i}, a^-_{1,i}, \ldots, a^+_{k,i}, a^-_{k,i}) \in \Sigma^\odot \quad (1 \leq i \leq m) \tag{4}$$

where $a^+_{j,i}, a^-_{j,i}, z^+_j, z^-_j$ are defined in Equation (1).

The grammar G is constructed as $G \stackrel{\text{def}}{=} (N_G, \Sigma, S_G, P_G)$, where $N_G \stackrel{\text{def}}{=} \{S, X\}$ and P_G is defined as follows:

$$S_G \to X\hat{\boldsymbol{c}}u \qquad\qquad X \to \epsilon \qquad\qquad X \to X\hat{\boldsymbol{c}}v_i \qquad\qquad (1 \leq i \leq m)$$

Here, c is the constant from (3) whose addition ensures that the values of the t^+_i and t^-_i generated by G are large. Clearly, G can be constructed in logarithmic space even though c is exponential in $|\phi|$. The following lemma captures the essential properties of G.

▶ **Lemma 6.** *Let G be as above. The following hold:*

(i) *For every $\boldsymbol{x} \in \mathbb{N}^m$ there exists $w \in \mathcal{L}(G)$ such that for all $1 \leq i \leq k$,*

$$w(t_i) = \sum_{1 \leq j \leq m} (a^+_{i,j} - a^-_{i,j}) \cdot x_j + z^+_i - z^-_i.$$

(ii) *For every $w \in \mathcal{L}(G)$ there exists $\boldsymbol{x} \in \mathbb{N}^m$ such that for all $1 \leq i \leq k$,*

$$w(t_i) = \sum_{1 \leq j \leq m} (a^+_{i,j} - a^-_{i,j}) \cdot x_j + z^+_i - z^-_i \tag{5}$$

$$w(t^+_i) \geq c + z^+_i + \sum_{1 \leq j \leq m} c \cdot x_j \geq c \cdot (1 + \|\boldsymbol{x}\|_\infty) \tag{6}$$

$$w(t^-_i) \geq c + z^-_i + \sum_{1 \leq j \leq m} c \cdot x_j \geq c \cdot (1 + \|\boldsymbol{x}\|_\infty). \tag{7}$$

We now turn towards the construction of $H \stackrel{\text{def}}{=} (N_H, \Sigma, S_H, P_H)$ and define the set of non-terminals N_H and productions P_H of H in a step-wise fashion. Starting in S_H, H branches into three gadgets starting at the non-terminal symbols Y, F_ψ and I:

$$S_H \to Y F_\psi I$$

Here, Y is an analogue to X in G. Informally speaking, it allows for obtaining the right-hand sides of the inequalities t_i for a choice of $\boldsymbol{y} \in \mathbb{N}^n$. In analogy to G, we define

$$w_i \stackrel{\text{def}}{=} (b^+_{1,i}, b^-_{1,i}, \ldots, b^+_{k,i}, b^-_{k,i}) \in \Sigma^\odot \qquad (1 \leq i \leq n)$$

$$Y \to Y w_i \qquad (1 \leq i \leq n)$$

$$Y \to \epsilon$$

In contrast to X from G, note that Y does not add $\hat{\boldsymbol{c}}$ every time it loops. The following lemma is the analogue of H to Lemma 6 and can be shown along the same lines.

▶ **Lemma 7.** *Let Y be the non-terminal of H as defined above. The following hold:*

(i) *For every $\boldsymbol{y} \in \mathbb{N}^n$ there exists $w \in \mathcal{L}(H, Y)$ such that for all $1 \leq i \leq k$, $w(t^+_i) = \sum_{1 \leq j \leq n} b^+_{i,j} \cdot y_j$, $w(t^-_i) = \sum_{1 \leq j \leq n} b^-_{i,j} \cdot y_j$, and*

$$w(t_i) = \sum_{1 \leq j \leq n} (b^+_{i,j} - b^-_{i,j}) \cdot y_j.$$

(ii) *For every $w \in \mathcal{L}(H, Y)$ there exists $\boldsymbol{y} \in \mathbb{N}^n$ such that for all $1 \leq i \leq k$,*

$$w(t_i) = \sum_{1 \leq j \leq n} (b^+_{i,j} - b^-_{i,j}) \cdot y_j.$$

It is clear that the w_Y generated by Y may not be able to generate all t_i in a way that match all w generated by G (i.e., all choices of $\boldsymbol{x}$ made through G). For now, let us even assume that $w(t^+_i) \geq w_Y(t^+_i)$ and $w(t^-_i) \geq w_Y(t^-_i)$ holds for every $1 \leq i \leq k$. Later, we will show that if there is a good choice for $\boldsymbol{y}$, we can find a good $w_Y \in \mathcal{L}(H, Y)$ with this property. After generating w_Y, informally speaking, H should produce t^+_i and t^-_i in order match w, provided that ψ is valid.

In particular, the Boolean structure of ψ enables us to produce arbitrary quantities of some t_i. This is the duty of the gadget F_ψ which allows for assigning arbitrary values to some

atomic formulas t_i via gadgets R_{t_i} defined below. The gadget F_ψ recursively traverses the matrix formula ψ and invokes some R_γ whenever a disjunction is processed and a disjunct γ is evaluated to false:

$$F_{t_i} \to \epsilon \qquad F_{\alpha\wedge\beta} \to F_\alpha F_\beta \qquad F_{\alpha\vee\beta} \to F_\alpha R_\beta \qquad F_{\alpha\vee\beta} \to R_\alpha F_\beta \qquad F_{\alpha\vee\beta} \to F_\alpha F_\beta$$

The definition of R_γ for every subformula γ of ψ occurring in the syntax tree of ψ is now not difficult: we traverse γ until we reach a leaf t_i of the syntax tree of γ and then allow for generating an arbitrary number of alphabet symbols t_i^+ and t_i^-. Let $1 \leq i \leq k$, we define the following productions:

$$R_{t_i} \to \epsilon \qquad R_{t_i} \to R_{t_i} t_i^+ \qquad R_{t_i} \to R_{t_i} t_i^- \qquad R_{\alpha\wedge\beta} \to R_\alpha R_\beta \qquad R_{\alpha\vee\beta} \to R_\alpha R_\beta$$

Finally, it remains to provide a possibility to increase $w_Y(t_i)$ for those t_i that were not processed by some R_{t_i} in order to match w. For a good choice of w_Y, we certainly should have that for those t_i, the number of t_i generated by w_Y in H is at least as much as the number generated by G. Hence, in order to make w_Y agree with w on t_i, all we have to do to w_Y is to non-deterministically increment, i.e., produce, t_i^+ at least as often as t_i^-. This is the task of the gadget I of H, whose production rules are as follows:

$$I \to \epsilon \qquad I \to I t_i^+ t_i^- \qquad I \to I t_i^+ \qquad (1 \leq i \leq k)$$

The subsequent lemma, whose proof is immediate, states the properties of I formally.

▶ **Lemma 8.** $\mathcal{L}(H, I) = \left\{(n_1^+, n_1^-, \ldots, n_k^+, n_k^-) \in \Sigma^\odot : n_j^+ \geq n_j^-,\ 1 \leq j \leq k\right\}$.

This completes the construction of H. We now prove the correctness of our construction.

▶ **Lemma 9.** *Suppose $\mathcal{L}(G) \subseteq \mathcal{L}(H)$, then $\phi = \forall \boldsymbol{x}.\exists \boldsymbol{y}.\psi(\boldsymbol{x}, \boldsymbol{y})$ is valid.*

Proof. The idea underlying the proof is to show how to construct for every choice of $\boldsymbol{x} \in \mathbb{N}^m$ some $\boldsymbol{y} \in \mathbb{N}^n$ such that $\psi(\boldsymbol{x}, \boldsymbol{y})$ evaluates to true. By Lemma 6(i), for any $\boldsymbol{x} \in \mathbb{N}^m$ there exists some corresponding $w \in \mathcal{L}(G)$ and by assumption $w \in \mathcal{L}(H)$. In particular, w is composed of some $w_Y \in \mathcal{L}(H, Y)$ from which by Lemma 7(ii) some suitable $\boldsymbol{y} \in \mathbb{N}^n$ can be obtained. ◀

The converse direction is slightly more involved. Informally speaking, on the first sight one might be worried that H produces more t_i^+ or t_i^- than G which cannot be "erased." However, the addition of c in every component for every production applied by G together with Theorem 4 allows us to overcome this obstacle.

▶ **Lemma 10.** *Suppose $\phi = \forall \boldsymbol{x}.\exists \boldsymbol{y}.\psi(\boldsymbol{x}, \boldsymbol{y})$ is valid, then $\mathcal{L}(G) \subseteq \mathcal{L}(H)$.*

Proof. Let $w \in \mathcal{L}(G)$, by Lemma 6(ii) there exists $\boldsymbol{x}^* \in \mathbb{N}^m$ such that (5), (6) and (7) hold. By assumption, there is $\boldsymbol{y}^* \in \mathbb{N}^n$ such that $\psi(\boldsymbol{x}^*, \boldsymbol{y}^*)$ holds. Hence, there is $\xi : \{1, \ldots, k\} \to \{0, 1\}$ such that for all i where $\xi(i) = 1$,

$$\sum_{1 \leq j \leq m} a_{i,j} \cdot x_j^* + z_i \geq \sum_{1 \leq j \leq n} b_{i,j} \cdot y_j^*$$

and $\psi[\xi(1)/t_1, \ldots, \xi(k)/t_k]$ evaluates to true. With no loss of generality, write $\{i : \xi(i) = 1\} = \{1, \ldots, h\}$ for some $1 \leq h \leq k$. Consider the system $D : A \cdot (\boldsymbol{x}, \boldsymbol{y}) \geq \boldsymbol{z}$ of linear Diophantine inequalities over the unknowns $\boldsymbol{x}$ and $\boldsymbol{y}$, where

$$A \stackrel{\text{def}}{=} \begin{pmatrix} a_{1,1} & \cdots & a_{1,m} & -b_{1,1} & \cdots & -b_{1,n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{h,1} & \cdots & a_{h,m} & -b_{h,1} & \cdots & -b_{h,n} \end{pmatrix} \qquad \boldsymbol{z} \stackrel{\text{def}}{=} \begin{pmatrix} -z_1 \\ \vdots \\ -z_h \end{pmatrix}.$$

By assumption, D has a non-empty solution set. We have that A is a $h \times (m+n)$ matrix with $\|M\|_{1,\infty} \leq \|\psi\|_{1,\infty}$ and $\|\boldsymbol{z}\|_\infty \leq \|\psi\|_\infty$. By Theorem 4, there are $B, P \subseteq \mathbb{N}^{m+n}$ such that $[\![D]\!] = B + \mathrm{cone}(P)$. Consequently, $\boldsymbol{x}^* = \pi_{[1,m]}(\boldsymbol{b} + \lambda_1 \cdot \boldsymbol{p}_1 + \cdots + \lambda_\ell \cdot \boldsymbol{p}_\ell)$ for some $\boldsymbol{b} \in B$, $\boldsymbol{p}_i \in P$ and $\lambda_i \in \mathbb{N}$. In particular, since $|P| \leq \binom{h+m+n}{h} \leq 2^{|\phi|}$ we have

$$0 \leq \sum_{1 \leq i \leq \ell} \lambda_i \leq \|\boldsymbol{x}^*\|_\infty \cdot \ell \leq \|\boldsymbol{x}^*\|_\infty \cdot 2^{|\phi|}. \tag{8}$$

Now let

$$\boldsymbol{y}^\dagger \stackrel{\text{def}}{=} \pi_{[m+1,m+n]}(\boldsymbol{b} + \lambda_1 \cdot \boldsymbol{p}_1 + \cdots + \lambda_\ell \cdot \boldsymbol{p}_\ell).$$

We have $(\boldsymbol{x}^*, \boldsymbol{y}^\dagger)$ is a solution of D and henceforth $\psi[\boldsymbol{x}^*/\boldsymbol{x}, \boldsymbol{y}^\dagger/\boldsymbol{y}]$ evaluates to true. In fact, it is not difficult to show that

$$\|\boldsymbol{y}^\dagger\|_\infty \leq (1 + \|\boldsymbol{x}^*\|_\infty) \cdot \frac{c}{|\phi|^2}.$$

Combining the estimation of $\|\boldsymbol{y}^\dagger\|_\infty$ with (6) and (7) of Lemma 6, for every $1 \leq i \leq k$ we obtain

$$w(t_i^+), w(t_i^-) \geq c \cdot (1 + \|\boldsymbol{x}^*\|_\infty) \geq \|\boldsymbol{y}^\dagger\|_\infty \cdot |\phi|^2 \geq \|\boldsymbol{y}^\dagger\|_\infty \cdot \|\phi\|_\infty \cdot |\phi|. \tag{9}$$

By Lemma 7(i) there is $w_Y \in \mathcal{L}(H, Y)$ such that (9) yields

$$w(t_i^+) \geq \sum_{1 \leq j \leq n} \|\boldsymbol{y}^\dagger\|_\infty \cdot \|\phi\|_\infty \geq \sum_{1 \leq j \leq n} b_{i,j}^+ \cdot y_j^\dagger = w_Y(t_i^+)$$
$$w(t_i^-) \geq \sum_{1 \leq j \leq n} \|\boldsymbol{y}^\dagger\|_\infty \cdot \|\phi\|_\infty \geq \sum_{1 \leq j \leq n} b_{i,j}^- \cdot y_j^\dagger = w_Y(t_i^-).$$

Moreover, the construction of F_ψ is such that

$$\left\{w_F \in \Sigma^\odot : w_F(t_i^+) = w_F(t_i^-) = 0,\ \xi(i) = 1,\ 1 \leq i \leq k\right\} \subseteq \mathcal{L}(H, F_\psi).$$

Hence, we can find some $w_F \in \mathcal{L}(H, F_\psi)$ which allows us to adjust those t_i for which $\xi(i) = 0$. More formally, for $1 \leq i \leq k$ such that $\xi(i) = 0$,

$$(w_Y + w_F)(t_i^+) = w(t_i^+) \text{ and } (w_Y + w_F)(t_i^-) = w(t_i^-).$$

On the hand, for all $1 \leq i \leq k$ such that $\xi(i) = 1$,

$$(w_Y + w_F)(t_i^+) = w_Y(t_i^+) \text{ and } (w_Y + w_F)(t_i^+) = w_Y(t_i^+),$$

i.e., those t_i remain untouched by w_F.

Consequently, it remains to show that there is a suitable $w_I \in \mathcal{L}(H, I)$ such that we can adjust those t_i which were left untouched by w_F above. For all $1 \leq i \leq k$ such that $\xi(i) = 1$, since $\boldsymbol{y}^\dagger$ is a solution of D, we have

$$\begin{aligned}
& w(t_i) = w(t_i^+) - w(t_i^-) \geq w_Y(t_i^+) - w_Y(t_i^-) = w_Y(t_i) \\
\iff & w(t_i^+) - w_Y(t_i^+) \geq w(t_i^-) - w_Y(t_i^-) \\
\iff & \text{there are } m_i, n_i \in \mathbb{N} \text{ such that } w(t_i^+) = w_Y(t_i^+) + m_i + n_i \text{ and} \\
& w(t_i^-) = w_Y(t_i^-) + m_i.
\end{aligned}$$

But then Lemma 8 yields the required $w_I \in \mathcal{L}(H, I)$ such that $w_I(t_i^+) = m_i + n_i$, $w_I(t_i^-) = m_i$, and $w_I(t_j^+) = w_I(t_j^+) = 0$ for all j such that $\xi(j) = 0$.

Summing up, we have $w = w_Y + w_I + w_F$, and hence $w \in \mathcal{L}(H)$ as required. ◀

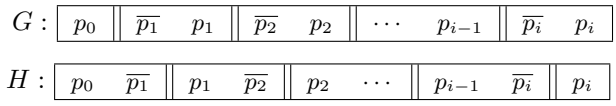


Figure 1 Illustration of the pairing of alphabet symbols. In G, we require that in each cell $w(p_{j+1}) = 2 \cdot w(\overline{p_j})$, and in H that $w(p_j) = w(\overline{p_j})$. Any word fulling these conditions has the property that $w(p_i) = 2^i \cdot w(p_0)$.

Lemmas 9 and 10 together with Theorem 3 yield the coNEXP-lower bound of Theorems 1 and 2 of the language inclusion problem for context-free and exponent-sensitive grammars, and hence coNEXP-hardness of the equivalence problem.

▶ Remark. It is not difficult to see that one can derive from G and H commutative context-free grammars G^e and H^e such that an even stronger statement holds:

$$\phi \text{ is valid} \iff \mathcal{R}(G^e) = \mathcal{R}(H^e).$$

3.1 Hardness for Regular Commutative Grammars

It remains to show how our reduction can be adapted in order to prove coNEXP-hardness of the equivalence problem for regular commutative grammars. Due to space constraints, we only sketch the main ideas, full details can be found in the technical report accompanying this paper.

As constructed above, neither G nor H are regular, the main problem being the following rules of G:

$$S_G \to X\hat{\boldsymbol{c}}u \qquad X \to X\hat{\boldsymbol{c}}v_i \qquad (1 \leq i \leq m).$$

Here, $\hat{\boldsymbol{c}}$ is a word of exponential length, cf. Equation (3), and, informally speaking, we cannot force a regular commutative grammar to generate an exponential quantity of an alphabet symbol. However, the interplay between G and H allows us to do so. The main idea is that in order to generate $\hat{\boldsymbol{c}}$ we use additional alphabet symbols $p_0, \ldots, p_i$ such that we require that number of occurrences of p_{j+1} is twice as much as p_j in a word w accepted by G for all $0 \leq j < i$, or otherwise this word is trivially accepted by H. With this approach we get that that if w witnesses $\mathcal{L}(G) \not\subseteq \mathcal{L}(H)$ then $w(p_i) = 2^i \cdot w(p_0)$, which is exactly what we need. In some more detail, the construction actually uses further additional alphabet symbols $\overline{p_1}, \ldots, \overline{p_i}$. Then, we enforce in G that $w(p_{j+1}) = 2 \cdot w(\overline{p_j})$, and in H that $w(p_j) = w(\overline{p_{j+1}})$. This can be achieved by accepting any word w in H for which $w(p_j) \neq w(\overline{p_{j+1}})$. Figure 1 illustrates this technique of pairing alphabet symbols p_j and $\overline{p_j}$.

Finally, the gadget F_ψ of H is also not regular. However, we can alternatively simulate ψ by a regular grammar in which conjunction in ψ corresponds to sequential composition and disjunction to branching.

4 Exponent-Sensitive Commutative Grammars

We now turn towards the equivalence problem for exponent-sensitive commutative grammars and sketch the proof of Theorem 2, i.e., show that language inclusion is coNEXP-hard and in co-2NEXP. The lower bound immediately follows from Theorem 1. Hence, it remains to provide a co-2NEXP upper bound, thereby improving the 2EXPSPACE upper bound from [21]. All formal details can be found in the technical report accompanying this paper.

It is sufficient to show that language inclusion between exponent-sensitive commutative grammars can be decided in co-2NEXP. To this end, we adapt an approach proposed by

Huynh used to show that language inclusion between context-free commutative grammars is in coNEXP [14]. Let G and H be exponent-sensitive commutative grammars. The starting point of Huynh's approach is to derive bounds on the size of a commutative word witnessing non-inclusion via the semi-linear representation of the reachability sets of G and H. For exponent-sensitive commutative grammars, in [22] $\mathcal{R}(G)$ and $\mathcal{R}(H)$ are shown semi-linear with a representation size doubly exponential in $\#G$ and in $\#H$, respectively, and this representation is also computable in doubly-exponential time. Given semi-linear sets M and N such that $M \setminus N$ is non-empty, Huynh shows in [15] that there is some $\boldsymbol{v} \in M \setminus N$ whose bit-size is polynomial in $\#M + \#N$. Consequently, if $\mathcal{L}(G) \not\subseteq \mathcal{L}(H)$ then the binary representation of some word $w \in \mathcal{L}(G) \setminus \mathcal{L}(H)$ has size bounded by $2^{2^{p(\#G+\#H)}}$ for some polynomial p. Since the word problem for exponent-sensitive commutative grammars is in PSPACE, deciding $\mathcal{L}(G) \subseteq \mathcal{L}(H)$ is in 2-EXPSPACE, as observed in [22, Thm. 5.5]. Now comes the second part of Huynh's approach into play. In [14], a Carathéodory-type theorem for semi-linear sets is established: given a linear set $M = L(\boldsymbol{b}, P) \subseteq \mathbb{N}^m$, Huynh shows that $M = \bigcup_{i \in I} L(\boldsymbol{b}_i, P_i)$, where $\boldsymbol{b}_i \in L(\boldsymbol{b}, P)$, each $\boldsymbol{b}_i$ has bit-size polynomial in $\#M$, and $P_i \subseteq P$ has full column rank and hence in particular $|P_i| \leq m$. The key point is that deciding membership in a linear set with such properties is obviously in P using Gaussian elimination, and that we can show that a semi-linear representation of $\mathcal{R}(G)$ and $\mathcal{R}(H)$ in which every linear set has those properties is computable in deterministic doubly-exponential time in $\#G$ and in $\#H$, respectively. Consequently, a co-2NEXP algorithm to decide $\mathcal{L}(G) \subseteq \mathcal{L}(H)$ can initially guess a word w whose representation is doubly-exponential in $\#G + \#H$, then compute the semi-linear representations of $\mathcal{R}(G)$ and $\mathcal{R}(H)$ in the special form of Huynh, and check in time polynomial in $\#w$ that w belongs to $\mathcal{L}(G)$ and not to $\mathcal{L}(H)$.

5 Applications to Further Equivalence Problems

Here, we discuss immediate corollaries of Theorem 1 for various other equivalence problems in formal language and automata theory. Due to space constraints, we cannot provide formal definitions of the objects we consider; they can be found in the references in the respective paragraphs.

In [2], Eilenberg and Schützenberger studied properties of regular languages in commutative monoids which are generated by regular commutative expressions. Such regular expressions are the same as standard regular expression which use the free commutative monoid instead of the free monoid. From Theorem 1, we obtain the following statement.

▶ **Theorem 11.** *Language equivalence between regular commutative expressions is* coNEXP*-complete.*

The upper bound can easily be obtained via a reduction to equivalence between regular commutative grammars. The lower bound follows from the observation that the construction outlined in Section 3.1 can be adjusted in a way such that the directed graph underlying the constructed regular commutative grammar does not contain nested cycles, and can hence be translated into an equivalent regular commutative expression of linear size.

Regular commutative grammars can also be viewed as 0-reversal-bounded counter automata in which every counter corresponds to an alphabet symbol. Reversal-bounded counter automata were introduced by Ibarra [16]. Along a run of a k-reversal bounded counter automaton, every counter may only change from incrementing to decrementing mode at most k times. Given two reversal-bounded counter automata with the same number of counters, equivalence is to decide whether their sets of counter values occurring in a final configuration is the same.

Table 1 Complexity of the word and the equivalence problem for classes of commutative grammars.

commutative grammar	word problem	language equivalence
type-0	EXPSPACE-h. [20], $\in \mathbf{F}_{\omega^3}$ [19]	undecidable [10]
context-sensitive	PSPACE-complete [13]	undecidable [14]
exponent-sensitive	PSPACE-complete [21]	coNEXP-h., $\in$ co-2NEXP
context-free	NP-complete [13, 3]	coNEXP-complete
regular		

▶ **Theorem 12.** *The equivalence problem for reversal-bounded counter automata is* coNEXP*-complete.*

The lower bound immediately follows from Theorem 1. For the upper bound, Hague and Lin [11] have shown that the set of counter values occurring in a final configuration is definable in existential Presburger arithmetic. Consequently, given two reversal-bounded counter automata whose reachability sets are defined by existential Presburger formulas $\varphi(\boldsymbol{x})$ and $\psi(\boldsymbol{x})$, respectively, they are equivalent iff $\phi \stackrel{\text{def}}{=} \forall \boldsymbol{x}.\varphi(\boldsymbol{x}) \leftrightarrow \psi(\boldsymbol{x})$ is valid. Since ϕ is a Π_2-sentence of Presburger arithmetic, Theorem 3 yields a coNEXP-upper bound for the equivalence problem.

Finally, it has, for instance, been observed in [3, 27, 23] that context-free commutative grammars can be seen as notational variants of communication-free Petri nets and basic parallel process nets (BPP-nets). In particular, language equivalence is logarithmic-space interreducible with reachability equivalence for such nets. Hence, Theorem 1 together with Remark 3 yields the following theorem.

▶ **Theorem 13.** *The equivalence problem for communication-free Petri nets and BPP-nets is* coNEXP*-complete.*

6 Conclusion

We showed that language inclusion and equivalence for regular and context-free commutative grammars are coNEXP-complete, resolving a long-standing open question posed by Huynh [14]. Our lower bound also carries over to the equivalence problem for exponent-sensitive commutative grammars, for which we could also improve the 2-EXPSPACE-upper bound [21] to co-2NEXP. The precise complexity of this problem remains an open problem of this paper. An overview over the complexity of word and equivalence problems for commutative grammars together with references to the literature is provided in Table 1.

One major open problem related to the problems discussed in this paper is weak bisimilarity between basic parallel processes. This problem is not known to be decidable and PSPACE-hard [25]. Unfortunately, it does not seem possible to adjust the construction of our coNEXP-lower bound to also work for weak bisimulation.

References

1 Eric Domenjoud. Solving systems of linear Diophantine equations: An algebraic approach. In *Mathematical Foundations of Computer Science (MFCS)*, pages 141–150, 1991.

2 Samuel Eilenberg and M.P. Schützenberger. Rational sets in commutative monoids. *Journal of Algebra*, 13(2):173–191, 1969.

3 Javier Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundamenta Informaticae*, 31(1):13–25, 1997. doi:10.3233/FI-1997-3112.

4 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

5 Seymour Ginsburg. *The mathematical theory of context free languages*. McGraw-Hill, 1966.

6 Seymour Ginsburg and Edwin H. Spanier. Bounded ALGOL-like languages. *Transactions of the American Mathematical Society*, pages 333–368, 1964.

7 Erich Grädel. Dominoes and the complexity of subclasses of logical theories. *Annals of Pure Applied Logic*, 43(1):1–30, 1989. doi:10.1016/0168-0072(89)90023-7.

8 Christoph Haase. Subclasses of Presburger arithmetic and the weak EXP hierarchy. In *Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) (CSL-LICS)*, pages 47:1–47:10. ACM, 2014. doi:10.1145/2603088.2603092.

9 Christoph Haase and Simon Halfon. Integer vector addition systems with states. In *Proceedings of the 8th International Workshop on Reachability Problems (RP)*, volume 8762 of *Lecture Notes in Computer Science*, pages 112–124. Springer, 2014. doi:10.1007/978-3-319-11439-2_9.

10 Michel Hack. The equality problem for vector addition systems is undecidable. *Theoretical Computer Science*, 2(1):77–95, 1976. doi:10.1016/0304-3975(76)90008-6.

11 Matthew Hague and Anthony Widjaja Lin. Model checking recursive programs with numeric data types. In *Proccedings of the 23rd International Conference on Computer Aided Verification (CAV)*, volume 6806 of *Lecture Notes in Computer Science*, pages 743–759. Springer, 2011. doi:10.1007/978-3-642-22110-1_60.

12 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation – international edition (2. ed)*. Addison-Wesley, 2003.

13 Dung T. Huynh. Commutative grammars: The complexity of uniform word problems. *Information and Control*, 57(1):21–39, 1983. doi:10.1016/S0019-9958(83)80022-9.

14 Dung T. Huynh. The complexity of equivalence problems for commutative grammars. *Information and Control*, 66(1–2):103–121, 1985. doi:10.1016/S0019-9958(85)80015-2.

15 Dung T. Huynh. A simple proof for the Σ_2^p upper bound of the inequivalence problem for semilinear sets. *Elektronische Informationsverarbeitung und Kybernetik*, 22(4):147–156, 1986.

16 Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, 1978. doi:10.1145/322047.322058.

17 Eryk Kopczyński. Complexity of problems of commutative grammars. *Logical Methods in Computer Science*, 11(1), 2015. doi:10.2168/lmcs-11(1:9)2015.

18 Eryk Kopczyński and Anthony Widjaja To. Parikh images of grammars: Complexity and applications. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, (LICS)*, pages 80–89. IEEE Computer Society, 2010. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5570020, doi:10.1109/LICS.2010.21.

19 Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 56–67. IEEE, 2015. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7174833.

20 Richard Lipton. The reachability problem is exponential-space-hard. Technical report, Yale University, New Haven, CT, 1976.

21 Ernst W. Mayr and Jeremias Weihmann. Completeness results for generalized communication-free Petri nets with arbitrary edge multiplicities. In *Proceedings of the 7th International Workshop on Reachability Problems (RP)*, volume 8169 of *Lecture Notes in Computer Science*, pages 209–221. Springer, 2013. doi:10.1007/978-3-642-41036-9_19.

22 Ernst W. Mayr and Jeremias Weihmann. Completeness results for generalized communication-free Petri nets with arbitrary edge multiplicities. Technical Report TUM-I1335, Technische Universität München, 2013. URL: http://mediatum.ub.tum.de/node?id=1169599.

23 Ernst W. Mayr and Jeremias Weihmann. Complexity results for problems of communication-free Petri nets and related formalisms. *Fundamenta Informaticae*, 137(1):61–86, 2015. doi:10.3233/FI-2015-1170.

24 Loïc Pottier. Minimal solutions of linear Diophantine systems: Bounds and algorithms. In *Proceedings of the 4th International Conference on Rewriting Techniques and Applications (RTA)*, volume 488 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 1991. doi:10.1007/3-540-53904-2_94.

25 Jirí Srba. Complexity of weak bisimilarity and regularity for BPA and BPP. *Mathematical Structures in Computer Science*, 13(4):567–587, 2003. doi:10.1017/S0960129503003992.

26 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *5th Annual ACM Symposium on Theory of Computing, STOC*, pages 1–9. ACM, 1973.

27 Hsu-Chun Yen. On reachability equivalence for BPP-nets. *Theoretical Computer Science*, 179(1-2):301–317, 1997. doi:10.1016/S0304-3975(96)00147-8.

Autoreducibility of NP-Complete Sets*

John M. Hitchcock[1] and Hadi Shafei[2]

1 **Department of Computer Science, University of Wyoming, USA**
2 **Department of Computer Science, University of Wyoming, USA**

Abstract

We study the polynomial-time autoreducibility of NP-complete sets and obtain separations under strong hypotheses for NP. Assuming there is a p-generic set in NP, we show the following:

- For every $k \geq 2$, there is a k-T-complete set for NP that is k-T autoreducible, but is not k-tt autoreducible or $(k-1)$-T autoreducible.
- For every $k \geq 3$, there is a k-tt-complete set for NP that is k-tt autoreducible, but is not $(k-1)$-tt autoreducible or $(k-2)$-T autoreducible.
- There is a tt-complete set for NP that is tt-autoreducible, but is not btt-autoreducible.

Under the stronger assumption that there is a p-generic set in NP $\cap$ coNP, we show:

- For every $k \geq 2$, there is a k-tt-complete set for NP that is k-tt autoreducible, but is not $(k-1)$-T autoreducible.

Our proofs are based on constructions from separating NP-completeness notions. For example, the construction of a 2-T-complete set for NP that is not 2-tt-complete also separates 2-T-autoreducibility from 2-tt-autoreducibility.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases computational complexity, NP-completeness, autoreducibility, genericity

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.42

1 Introduction

Autoreducibility measures the redundancy of a set. For a reducibility $\mathcal{R}$, a set A is $\mathcal{R}$-autoreducible if there is a $\mathcal{R}$-reduction from A to A where the instance is never queried [15]. Understanding the autoreducibility of complete sets is important because of applications to separating complexity classes [5]. We study the polynomial-time autoreducibility [1] of NP-complete sets.

Natural problems are paddable and easily shown to be m-autoreducible. In fact, Glaßer et al. [8] showed that all nontrivial m-complete sets for NP and many other complexity classes are m-autoreducible. Beigel and Feigenbaum [4] showed that T-complete sets for NP and the levels of the polynomial-time hierarchy are T-autoreducible. We focus on intermediate reducibilities between many-one and Turing.

Previous work has studied separations of these autoreducibility notions for larger complexity classes. Buhrman et al. [5] showed there is a 3-tt-complete set for EXP that is not btt-autoreducible. For NEXP, Nguyen and Selman [13] showed there is a 2-T-complete set that is not 2-tt-autoreducible and a tt-complete set that is not btt-autoreducible. We investigate whether similar separations hold for NP.

Since all NP sets are 1-tt-autoreducible if P = NP, it is necessary to use a hypothesis at least as strong as P $\neq$ NP to separate autoreducibility notions. We work with the *Genericity*

* This research was support in part by NSF grant 0917417.

© John M. Hitchcock and Hadi Shafei;
licensed under Creative Commons License CC-BY

Hypothesis that there is a p-generic set in NP [3, 2]. This is stronger than P $\neq$ NP, but weaker than the *Measure Hypothesis* [12, 10] that there is a p-random set in NP. Under the Genericity Hypothesis, we separate many autoreducibility notions for NP-complete sets. Our main results are summarized in Table 1.

Previous work has used the measure and genericity hypotheses to separate completeness notions for NP. Consider the set

$$C = G\dot{\cup}(G \cap \mathrm{SAT})\dot{\cup}(G \cup \mathrm{SAT}),$$

where $G \in$ NP and $\dot{\cup}$ is disjoint union. Then C is 2-T-complete for NP, and if G is p-generic, C is not 2-tt-complete [12, 2]. There is a straightforward 3-T (also 5-tt) autoreduction of C based on padding SAT.[1] However, since C is 2-T-honest-complete, we indirectly obtain a 2-T (also 3-tt) autoreduction by first reducing through SAT (Lemma 2.1). In Theorem 3.1 we show C is not 2tt-autoreducible.

It turns out this idea works in general. We show that many sets which separate completeness notions also separate autoreducibility notions. Ambos-Spies and Bentzien [2] also separated both k-T-completeness and $(k+1)$-tt-completeness from both k-tt-completeness and $(k-1)$-T-completeness for every $k \geq 3$ under the Genericity Hypothesis. We show that the same sets also separate k-T-autoreducibility and $(k+1)$-tt-autoreducibility from k-tt-autoreducibility and $(k-1)$-T-autoreducibility (Theorems 3.4 and 3.5). We also obtain that there is a tt-complete set for NP that is tt-autoreducible and not btt-autoreducible (Theorem 3.6), again using a construction of Ambos-Spies and Bentzien.

In the aforementioned results, there is a gap – we only separate k-tt-autoreducibility from $(k-2)$-T-autoreducibility (for $k \geq 3$), where we can hope for a separation from $(k-1)$-T-autoreducibility. The separation of k-tt from $(k-1)$-T is also open for completeness under the Genericity Hypothesis (or the Measure Hypothesis). To address this gap, we use a stronger hypothesis on the class NP$\cap$coNP. Pavan and Selman [14] showed that if NP$\cap$coNP contains a DTIME(2^{n^ϵ})-bi-immune set, then 2-tt-completeness is different from 1-tt-completeness for NP. We show that if NP $\cap$ coNP contains a p-generic set, then k-tt-completeness is different from $(k-1)$-T-completeness for all $k \geq 3$ (Theorem 4.2). We then show these constructions also separate autoreducibility: if there is a p-generic set in NP $\cap$ coNP, then for every $k \geq 2$, there is a k-tt-complete set for NP that is k-tt autoreducible, but is not $(k-1)$-T autoreducible (Theorems 4.1 and 4.3).

This paper is organized as follows. Preliminaries are in Section 2. The results using the Genericity Hypothesis are presented in Section 3. We use the stronger hypothesis on NP $\cap$ coNP in Section 4. Section 5 concludes with some open problems.

2 Preliminaries

We use the standard enumeration of binary strings, i.e $s_0 = \lambda, s_1 = 0, s_2 = 1, s_3 = 00, ...$ as an order on binary strings. All languages in this paper are subsets of $\{0,1\}^*$ identified with their characteristic sequences. In other words, every language $A \in \{0,1\}^*$ is identified with $\chi_A = A[s_0]A[s_1]A[s_2]....$ If X is a set, equivalently a binary sequence, and $x \in \{0,1\}^*$ then

[1] Given an instance x of C, pad x to an instance y such that SAT$[x]$ = SAT$[y]$. We query $G[y]$ and then query either $G \cap \mathrm{SAT}[y]$ if $G[y] = 1$ or $G \cup \mathrm{SAT}[y]$ if $G[y] = 0$ to learn SAT$[y]$. Finally, if our instance is $G[x]$ the answer is obtained by querying $G \cap \mathrm{SAT}[x]$ if SAT$[y] = 1$ or by querying $G \cup \mathrm{SAT}[x]$ if SAT$[y] = 0$. If our instance is $G \cup \mathrm{SAT}[x]$ or $G \cap \mathrm{SAT}[x]$, we query $G[x]$ and combine that answer with SAT$[y]$.

Table 1 If $\mathcal{C}$ contains a p-generic set, then there is a $\mathcal{S}$-complete set in NP that is $\mathcal{S}$-autoreducible but not $\mathcal{R}$-autoreducible.

$\mathcal{C}$	$\mathcal{S}$	$\mathcal{R}$	notes
NP	k-T	k-tt	Theorem 3.1 ($k=2$), Theorem 3.4 ($k \geq 3$)
NP	k-T	$(k-1)$-T	Theorem 3.1 ($k=2$), Theorem 3.5 ($k \geq 3$)
NP	k-tt	$(k-1)$-tt	Corollary 3.2 ($k=3$), Theorem 3.4 ($k \geq 4$)
NP	k-tt	$(k-2)$-T	Corollary 3.3 ($k=3$), Theorem 3.5 ($k \geq 4$)
NP	tt	btt	Theorem 3.6
NP $\cap$ coNP	k-tt	$(k-1)$-T	Theorem 4.1 ($k=2$), Theorem 4.3 ($k \geq 3$)

$X \upharpoonright x$ is the initial segment of X for all strings before x, i.e the subset of X that contains every $y \in X$ that $y < x$.

All reductions in this paper are polynomial-time reductions, therefore we may not emphasize this every time we define a reduction. We use standard notions of reducibilities [11].

Given A, B, and $\mathcal{R} \in \{\text{m}, \text{T}, \text{tt}, k\text{-T}, k\text{-tt}, \text{btt}\}$, A is *polynomial-time $\mathcal{R}$-honest reducible* to B ($A \leq^{\text{p}}_{\mathcal{R}\text{-}h} B$) if $A \leq^{\text{p}}_{\mathcal{R}} B$ and there exist a constant c such that for every input x, every query q asked from B has the property $|x|^{1/c} < |q|$. In particular, a reduction $\mathcal{R}$ is called *length-increasing* if on every input the queries asked from the oracle are all longer than the input.

For any reduction $\mathcal{R} \in \{\text{m}, \text{T}, \text{tt}, k\text{-T}, k\text{-tt}, \text{btt}\}$ a language A is *$\mathcal{R}$-autoreducible* if $A \leq^{\text{p}}_{\mathcal{R}} A$ via a reduction where on every instance x, x is not queried.

The following lemma states that any honest-complete set for NP is also autoreducible under the same type of reduction. This follows because NP has a paddable, length-increasing complete set.

▶ **Lemma 2.1.** *Let $\mathcal{R} \in \{\text{m}, \text{T}, \text{tt}, k\text{-T}, k\text{-tt}, \text{btt}, \ldots\}$ be a reducibility. Then every $\mathcal{R}$-honest-complete set for* NP *is $\mathcal{R}$-autoreducible.*

Proof. Let $A \in$ NP be $\mathcal{R}$-honest-complete. Then there is an $\mathcal{R}$-honest reduction M from SAT to A. There exists $m \geq 1$ such that every query q output by M on an instance x satisfies $|q| \geq |x|^{\frac{1}{m}}$.

Since SAT is NP-complete via length-increasing many-one reductions, $A \leq^{\text{p}}_{\text{m}}$ SAT via a length-increasing reduction g. Since SAT is paddable, there is a polynomial-time function h such that for any y, $\text{SAT}[h(y)] = \text{SAT}[y]$ and $|h(y)| > |y|^m$.

To obtain our $\mathcal{R}$-autoreduction of A, we combine g, h, and M. On instance x of A, compute the instance $h(g(x))$ of SAT and use M to reduce $h(g(x))$ to A. Since $|h(g(x))| > |g(x)|^m > |x|^m$, every query q of M has $|q| > |h(g(x))|^{\frac{1}{m}} > |x|$. Therefore all queries are different than x and this is an autoreduction. ◀

Most of the results in this paper are based on a non-smallness hypothesis for NP called the *Genericity Hypothesis* that NP contains a p-generic set [3, 2]. In order to define genericity first we need to define what a *simple extension function* is. For any k, a simple n^k-extension function is a partial function from $\{0,1\}^*$ to $\{0,1\}$ that is computable in $O(n^k)$. Given a set A and an extension function f we say that f is *dense along* A if f is defined on infinitely many initial segments of A. A set A *meets* a simple extension function f at x if $f(A \upharpoonright x)$ is defined and equal to $A[x]$. We say A meets f if A meets f at some x. A set G is called p-*generic* if it meets every simple n^k-extension function for any $k \geq 1$ [2]. A partial function

$f : \{0,1\}^* \to (\{0,1\}^* \times \{0,1\})^*$ is called a *k-bounded extension function* if whenever $f(X \upharpoonright x)$ is defined, $f(X \upharpoonright x) = (y_0, i_0)...(y_m, i_m)$ for some $m < k$, and $x \leq y_0 < y_1 < ... < y_m$, where y_j's are strings and i_j's are either 0 or 1. A set A meets f at x if $f(A \upharpoonright x)$ is defined, and A agrees with f on all y_j's, i.e. if $f(A \upharpoonright x) = (y_0, i_0)...(y_m, i_m)$ then $A[y_j] = i_j$ for all $j \leq m$ [2].

We will use the following routine extension of a lemma in [2].

▶ **Lemma 2.2.** *Let $l, c \geq 1$ and let f be an l-bounded partial extension function defined on initial segments $\alpha = X \upharpoonright 0^n$ of length 2^n $(n \geq 1)$. Whenever $f(\alpha)$ is defined we have*

$$f(\alpha) = (y_{\alpha,1}, i_{\alpha,1}), ..., (y_{\alpha,l_\alpha}, i_{\alpha,l\alpha}),$$

where $l_\alpha \leq l$, $pos(\alpha) = (y_{\alpha,1}, ..., y_{\alpha,l_\alpha})$ is computable in 2^{cn} steps and $i_{\alpha,j}$ is computable in $2^{c|y_{\alpha,j}|}$ steps. Then for every p*-generic set G, if f is dense along G then G meets f.*

3 Autoreducibility Under the Genericity Hypothesis

We begin by showing the Genericity Hypothesis implies there is a 2-T-complete set that separates 2-T-autoreducibility from 2-tt-autoreducibility. The proof utilizes the construction of [12, 2] that of a set that separates 2-T-completeness from 2-tt-completeness.

▶ **Theorem 3.1.** *If* NP *contains a* p*-generic language, then there exists a* 2-T*-complete set in* NP *that is* 2-T*-autoreducible, but not* 2-tt*-autoreducible.*

Proof. Let $G \in$ NP be p-generic and define $C = G \dot\cup (G \cap \text{SAT}) \dot\cup (G \cup \text{SAT})$, where $\dot\cup$ stands for disjoint union [12, 2]. Disjoint union can be implemented by adding a unique prefix to each set and taking their union. To be more clear, let $C = 0G \cup 10(G \cap \text{SAT}) \cup 11(G \cup \text{SAT})$. It follows from closure properties of NP that $C \in$ NP.

To see that C is 2-T-complete, consider an oracle Turing machine M that on input x first queries $0x$ from C. If the answer is positive, i.e. $x \in G$, M queries $10x$ from C, and outputs the result. Otherwise, M queries $11x$ from C, and outputs the answer. This Turing machine always makes two queries from C, runs in polynomial time, and $M^C(x) = \text{SAT}[x]$. This completes the proof that C is also 2-T-completeness. Since all queries from SAT to C are length-increasing, it follows from Lemma 2.1 that C is 2-T-autoreducible.

The more involved part of the proof is to show that C is not 2-tt-autoreducible. To get a contradiction assume that C is 2-tt-autoreducible. This means there exist polynomial-time computable functions h, g_1, and g_2 such that for every $x \in \{0,1\}^*$,

$$C[x] = h(x, C[g_1(x)], C[g_2(x)])$$

and moreover $g_i(x) \neq x$ for $i = 1, 2$. Note that W.L.O.G. we can assume that $g_1(x) < g_2(x)$. For $x = 0z$, $10z$, or $11z$ define the value of x to be z, and let $x = 0z$ for some string z. We have:

$$C[x] = G[z] = h(x, C[g_1(x)], C[g_2(x)])$$

To get a contradiction, we consider different cases depending on whether some of the queries have the same value as x or not, and the Boolean function $h(x, ., .)$. For some of these cases we show they can happen only for finitely many z's, and for the rest we show that SAT$[z]$ can be decided in polynomial time. As a result SAT is decidable in polynomial time a.e., which contradicts the assumption that NP contains a p-generic language.

The complete proof will appear in the full version of the paper. ◀

▶ **Corollary 3.2.** *If* NP *contains a* p*-generic language, then there exists a* 3-tt*-complete set for* NP *that is* 3-tt*-autoreducible, but not* 2-tt*-autoreducible.*

Proof. This follows immediately from Theorem 3.1 and the fact that every 2-T reduction is a 3-tt reduction. ◀

▶ **Corollary 3.3.** *If* NP *contains a* p*-generic language, then there exists a* 3-tt*-complete set for* NP *that is* 3-tt*-autoreducible, but not* 1-T*-autoreducible.*

Our next theorem separates $(k+1)$-tt-autoreducibility from k-tt-autoreducibility and k-T-autoreducibility from k-tt-autoreducibility under the Genericity Hypothesis. The proof uses the construction of Ambos-Spies and Bentzien [2] that separates the corresponding completeness notions.

▶ **Theorem 3.4.** *If* NP *contains a* p*-generic language, then for every* $k \geq 3$ *there exists a set that is*

- $(k+1)$*-*tt*-complete for* NP *and* $(k+1)$*-*tt*-autoreducible,*
- k*-*T*-complete for* NP *and* k*-*T*-autoreducible, and*
- *not* k*-*tt*-autoreducible.*

Proof. Let $G \in \text{NP}$ be a p-generic language, and $z_1, ..., z_{(k+1)}$ be the first $k+1$ strings of length k. For $m = 1, ..., k-1$ define

$$\hat{G_m} = \{x \mid xz_m \in G\} \tag{1}$$

$$\hat{G} = \bigcup_{m=1}^{k-1} \hat{G_m} \tag{2}$$

$$A = \bigcup_{m=1}^{k-1} \{xz_m \mid x \in \hat{G_m}\} \bigcup \{xz_k \mid x \in \hat{G} \cap \text{SAT}\} \bigcup \{xz_{k+1} \mid x \in \hat{G} \cup \text{SAT}\} \tag{3}$$

Here are some properties of the sets defined above:

- For every x, $x \in \hat{G} \Leftrightarrow \exists 1 \leq i \leq k-1.\ xz_i \in G$.
- A contains strings in G that end with $z_1, ...,$ or $z_{(k-1)}$, i.e. $A(xz_i) = G(xz_i)$ for every x and $1 \leq i \leq k-1$.
- $xz_k \in A$ if and only if $x \in \text{SAT} \wedge (\exists 1 \leq i \leq k-1 . xz_i \in G)$.
- $xz_{(k+1)} \in A$ if and only if $x \in \text{SAT} \vee (\exists 1 \leq i \leq k-1 . xz_i \in G)$.
- $xz_j \notin A$ for $j > k+1$.

It is easy to show that $\text{SAT} \leq^{\text{p}}_{(k+1)-\text{tt}} A$. On input x, make queries $xz_1, ..., xz_{(k+1)}$ from A. If at least one of the answers to the first $k-1$ queries is positive, then $\text{SAT}[x]$ is equal to the kth query, i.e. $\text{SAT}[x] = A[xz_k]$. Otherwise $\text{SAT}[x]$ is equal to $A[xz_{(k+1)}]$. As a result, A is $(k+1)$-tt-complete for NP. If the queries are allowed to be dependent, we can choose between xz_k and $xz_{(k+1)}$ based on the answers to the first $(k-1)$ queries. Therefore A is also k-T-complete for NP. Since all these queries are honest, in fact length-increasing, it follows from Lemma 2.1 that A is both $(k+1)$-tt-autoreducible and k-T-autoreducible.

To get a contradiction, assume A is k-tt-autoreducible via $h, g_1, \ldots, g_k$. In other words, assume that for every x:

$$A[x] = h(x, A[g_1(x)], ..., A[g_k(x)]) \tag{4}$$

and $\forall 1 \leq i \leq k.\ g_i(x) \neq x$. In particular, we are interested in the case where $x = 0^n z_1 = 0^{n+k}$, and we have:

$$A(0^{n+k}) = h(0^{n+k}, A[g_1(0^{n+k})], ..., A[g_k(0^{n+k})]) \tag{5}$$

and all $g_i(0^{n+k})$'s are different from 0^{n+k} itself.

In the following we will define a bounded extension function f that satisfies the condition in Lemma 2.2 such that if G meets f at 0^{n+k} then (5) will fail. We use the p-genericity of G to show that G has to meet f at 0^{n+k} for some n which completes the proof. In other words, we define a bounded extension function f such that given n and $X \upharpoonright 0^n$, $f(X \upharpoonright 0^n) = (y_0, i_0)...(y_m, i_m)$ and if

$$\begin{aligned} &G \upharpoonright 0^n = X \upharpoonright 0^n \text{ and} \\ &\forall 0 \leq j \leq m.\ G(y_j) = i_j \end{aligned} \tag{6}$$

then

$$A(0^{n+k}) \neq h(0^{n+k}, A[g_1(0^{n+k})], ..., A[g_k(0^{n+k})]) \tag{7}$$

Moreover, m is bounded by some constant that does not depend on n and $X \upharpoonright 0^n$. Note that we want f to satisfy the conditions in Lemma 2.2, so y_j's and i_j's must be computable in $O(2^n)$ and $O(2^{|y_j|})$ steps respectively. After defining such f, by Lemma 2.2 G must meet f at 0^{n+k} for some n. This means (6) must hold. As a result, (7) must happen for some n, which is a contradiction.

f can force values of $G[y_i]$'s for a constant number of y_i's. Because of the dependency between G and A we can force values for $A[w]$, where w is a query, by using f to force values in G. This is done based on the strings that have been queried, and their indices as follows.

- If $w = vz_i$ for some $1 \leq i \leq k-1$ then $A[w] = G[w]$. Therefore we can force $A[w]$ to 0 or 1 by forcing the same value for $G[w]$.
- If $w = vz_k$ then $A[w] = \text{SAT}[v] \wedge (\bigvee_{l=1}^{k-1} G[vz_l])$, so by forcing all $G[vz_l]$'s to 0 we can make $A[w] = 0$.
- If $w = vz_{k+1}$ then $A[w] = \text{SAT}[v] \vee (\bigvee_{l=1}^{k-1} G[vz_l])$. In this case by forcing one of the $G[vz_l]$'s to 1 we can make $A[w] = 1$.

We will use these facts to force the value of A on queries on input 0^{n+k} on the left hand side of (5), and then force a value for $A[0^{n+k}]$ such that (5) fails. The first problem that we encounter is the case where we have both vz_k and vz_{k+1} among our queries. If this happens for some v then the strategy described above does not work. To force $A[vz_k]$ and $A[vz_{k+1}]$ to 0 and 1 respectively, we need to compute SAT$[v]$. If SAT$[v] = 0$ then $A[vz_k] = 0$, and $A[vz_{k+1}]$ can be forced to 1 by forcing $G[vz_l] = 1$ for some $1 \leq l \leq k-1$. On the other hand, if SAT$[v] = 1$ then $A[vz_{k+1}] = 1$, and forcing all $G[vz_l]$'s to 0 makes $A[vz_k] = 0$. This process depends on the value of SAT$[v]$, and v can be much longer that 0^{n+k}. Because of the time bounds in Lemma 2.2 the value forced for $A[0^{n+k}]$ cannot depend on SAT$[v]$. But note that we have k queries, and two of them are vz_k and vz_{k+1}. Therefore at least one of the strings $vz_1, ..., vz_{k-1}$ is not among the queries. We use this string as vz_l, and make $G[vz_l] = 1$ when SAT$[v] = 0$.

Now we define an auxiliary function α from the set of queries, called QUERY, to 0 or 1. The idea is that α computes the value of A on queries without computing $G[v]$, given that G meets the extension function. α is defined in two parts based on the length of the queries. For queries $w = vz_p$ that are shorter than 0^{n+k}, i.e. $|w| < n + k$, we define:

$$\alpha(w) = \begin{cases} X[w] & \text{if } 1 \leq p \leq k-1 \\ 1 & \text{if } p = k \ \wedge\ v \in \text{SAT} \ \wedge\ \exists 1 \leq l \leq k-1.\ vz_l \in X \\ 1 & \text{if } p = k+1 \ \wedge\ (v \in \text{SAT} \ \vee\ \exists 1 \leq l \leq k-1.\ vz_l \in X) \\ 0 & \text{otherwise} \end{cases}$$

This means that if $X \upharpoonright 0^{n+k} = G \upharpoonright 0^{n+k}$ then $\alpha(w) = A(w)$ for every query $w = vz_p$ with $|w| < n + k$.
On the other hand, for queries $w = vz_p$ that $|w| \geq n + k$, α is defined as:

$$\alpha(w) = \begin{cases} 1 & \text{if } v = 0^n \ \wedge p = 2 \\ \text{SAT}[v] & \text{if } v = 0^n \ \wedge \ p = k \\ 1 & \text{if } v = 0^n \ \wedge \ p = k+1 \\ 1 & \text{if } v \neq 0^n \wedge p = k+1 \\ 1 & \text{if } v \neq 0^n \wedge p = k-1 \wedge \forall l \in \{1, ..., k-1, k+1\}.\ vz_l \in \text{QUERY} \\ 0 & \text{otherwise} \end{cases}$$

For this part of α, our definition of the extension function, which is provided below, guarantees that $\alpha(w) = A[w]$ if (6) holds. Note that the first case in the definition above implies that k must be greater than or equal to 3, and that is the reason this proof does not work for separating 3-tt-autoreducibility from 2-tt-autoreducibility.

Now we are ready to define the extension function f. For any string v which is the value for some query, i.e. $\exists 1 \leq p \leq k+1.vz_p \in \text{QUERY}$, we define pairs of strings and 0 or 1's. These pairs will be part of our extension function. Fix some value v, and let r be the smallest index that $vz_r \notin \text{QUERY}$, or $k-1$ if such index does not exist, i.e.

$$r = min\{s \geq 1 | vz_s \notin \text{QUERY} \vee s = k-1\} \tag{8}$$

We will have one of the following cases:

1. If $v = 0^n$ then pairs $(vz_2, 1), (vz_3, 0), ..., (vz_{k-1}, 0)$ must be added to f.
2. If $v \neq 0^n$ and $vz_{k+1} \notin \text{QUERY}$ then add pairs $(vz_1, 0)$,...,$(vz_{k-1}, 0)$ to f.
3. If $v \neq 0^n$, $vz_{k+1} \in \text{QUERY}$ and $vz_k \notin \text{QUERY}$ add pairs (vz_i, j) for $1 \leq i \leq k-1$ where $j = 0$ for all i's except $i = r$ where $j = 1$.
4. If $v \neq 0^n$, $vz_{k+1} \in \text{QUERY}$ and $vz_k \in \text{QUERY}$ add pairs (vz_i, j) for $1 \leq i \leq k-1$ where $j = 0$ for all i's except $i = r$ where $j = 1 - \text{SAT}[v]$.

This process must be repeated for every v that is the value of some query. Finally, we add $(0^{n+k}, 1 - h(0^{n+k}, \alpha(g_1(0^{n+k})), ..., \alpha(g_k(0^{n+k}))))$ to f in order to refute the autoreduction. It is worth mentioning that in the fourth case above, since both vz_k and vz_{k+1} are among queries, at least one of the strings vz_1,...,$vz_{k-1}$is not queried. Therefore by definition of r, $vz_r \notin \text{QUERY}$. This is important, as we describe in more detail later, because we forced $G[vz_r] = 1 - \text{SAT}[v]$, and if $vz_r \in \text{QUERY}$ then $\alpha(vz_r) = G[vz_r] = 1 - \text{SAT}[v]$. But α must be compuatable in $O(2^n)$ steps, which is not possible if v is much longer than 0^{n+k}.

Now that the extension function is defined completely, we need to show that it has the desired properties. First, we will show that if G meets f at 0^{n+k}, i.e. (6) holds, then α and A agree on every query w with $|w| \geq n + k$, i.e. $\alpha(w) = A[w]$.
Let $w = vz_p$, and $|w| \geq n + k$.

- If $v = 0^n$ and $p = 2$ then $\alpha(w) = 1$ and $A[w] = G[w] = 1$.
- If $v = 0^n$ and $p = k$ then $\alpha(w) = \text{SAT}[v]$ and $A[w] = \text{SAT}[v] \wedge (\bigvee_{l=1}^{k-1} G[vz_l])$. Since $G[vz_2] = 1$ is forced, $A[w] = \text{SAT}[v]$.
- If $v = 0^n$ and $p = k + 1$ then $\alpha(w) = 1$ and $A[w] = \text{SAT}[v] \vee (\bigvee_{l=1}^{k-1} G[vz_l]) = 1$ since $G[vz_2] = 1$.
- If $v = 0^n$ and $p \neq 2, k, k+1$ then $\alpha(w) = A[w] = 0$.
- If $v \neq 0^n$ and $p < k-1$ then $\alpha(w) = 0$. Since $p < k-1$, and $vz_p \in \text{QUERY}$, by definition of r, $r \neq p$. Therefore $G[vz_p]$ is forced to 0 by f. As a result, $A[w] = A[vz_p] = G[vz_p] = 0 = \alpha(w)$.

- If $v \neq 0^n$, $p = k-1$, and vz_1,...,vz_{k-1},$vz_{k+1} \in$ QUERY then $\alpha(w) = 1$. In this case $r = k-1$, so it follows from definition of f that $G[vz_{k-1}] = 1$. As a result, $A[w] = A[vz_{k-1}] = G[vz_{k-1}] = 1 = \alpha(w)$.
- If $v \neq 0^n$, $p = k-1$, and at least one of the strings vz_1,...,vz_{k-1},vz_{k+1} is not queried then we consider two cases. If $vz_{k+1} \notin$ QUERY then f forces $G[vz_{k-1}]$ to 0. On the other hand, if $vz_{k=1} \in$ QUERY, then at least one of vz_1,...,vz_{k-1} is not a query. Therefore by definition of r, $r \neq k-1$. This implies that $G[vz_{k-1}] = 0$ by f.
- If $v \neq 0^n$, $p = k$ then $\alpha(w) = 0$. Consider two cases. If $vz_{k+1} \notin$ QUERY then $G[vz_i] = 0$ for every $1 \leq i \leq k-1$. Therefore $A[w] = \text{SAT}[v] \wedge (\bigvee_{l=1}^{k-1} G[vz_l]) = 0$. Otherwise, when $vz_{k+1} \in$ QUERY, since we know that vz_k also belongs to QUERY, f forces $G[vz_r] = 1 - \text{SAT}[v]$, and $G[vz_i] = 0$ for every other $1 \leq i \leq k-1$. Therefore $A[w] = \text{SAT}[v] \wedge (\bigvee_{l=1}^{k-1} G[vz_l]) = \text{SAT}[v] \wedge (1 - \text{SAT}[v]) = 0$.
- If $v \neq 0^n$, $p = k+1$ then $\alpha(w) = 1$. If $vz_k \notin$ QUERY then $G[vz_r] = 1$ by f. Therefore $A[w] = \text{SAT}[v] \vee (\bigvee_{l=1}^{k-1} G[vz_l]) = 1$. On the other hand, if $vz_k \in$ QUERY then f forces $G[vz_r] = 1 - \text{SAT}[v]$. As a result, $A[w] = \text{SAT}[v] \vee (\bigvee_{l=1}^{k-1} G[vz_l]) = 1$.

This shows that in any case, $\alpha(w) = A[w]$ for $w \in$ QUERY, given that (6) holds, i.e G meets f. By combining this with (5) we have

$$\begin{aligned} A(0^{n+k}) =& h(0^{n+k}), A(g_1(0^{n+k})), ..., A(g_k(0^{n+k}))) \\ =& h(0^{n+k}, \alpha(g_1(0^{n+k})), ..., \alpha(g_k(0^{n+k}))) \end{aligned}$$

On the other hand, we forced $A[0^{n+k}] = 1 - h(0^{n+k}, \alpha(g_1(0^{n+k})), ..., \alpha(g_k(0^{n+k})))$ which gives us the desired contradiction.

The last part of our proof is to show that f satisfies the conditions in Lemma 2.2. For every value v which is the value of some query we added $k-1$ pairs to f, and there are k queries, which means at most k different values. Therefore, the number of pairs in f is bounded by k^2, i.e. f is a bounded extension function.

If $f(X \restriction 0^{n+k}) = (y_0, j_0), ..., (y_m, j_m)$ then y_i's are computable in polynomial ime in n, and j_i's are computable in $O(2^{|y_i|})$ because the most time consuming situation is when we need to compute SAT$[v]$ which is doable in $O(2^n)$. For the condition forced to the left hand side of (5), i.e $G[0^{n+k}] = 1 - h(0^{n+k}, \alpha(g_1(0^{n+k})), ..., \alpha(g_k(0^{n+k})))$, note that $\alpha(w)$ can be computed in at most $O(2^n)$ steps for $w \in$ QUERY, and h is computable in polynomial time. ◀

Next we separate $(k+1)$-tt-autoreducibility and k-T-autoreducibility from $(k-1)$-T-autoreducibility. The proof uses the same construction from the previous theorem, which Ambos-Spies and Bentzien [2] showed separates these completeness notions.

▶ **Theorem 3.5.** *If* NP *contains a* p*-generic language, then for every* $k \geq 3$ *there exists a set that is*

- $(k+1)$-tt*-complete for* NP *and* $(k+1)$-tt*-autoreducible,*
- k-T*-complete for* NP *and* k-T*-autoreducible, and*
- *not* $(k-1)$-T*-autoreducible.*

The proof of Theorem 3.5 will appear in the full version of the paper.

We now separate unbounded truth-table autoreducibility from bounded truth-table autoreducibility under the Genericity Hypothesis. This is based on the technique of Ambos-Spies and Bentzien [2] separating the corresponding completeness notions.

▶ **Theorem 3.6.** *If* NP *has a* p*-generic language, then there exists a* tt*-complete set for* NP *that is* tt*-autoreducible, but not* btt*-autoreducible.*

The proof of Theorem 3.6 will appear in the full version of the paper.

4 Stronger Separations Under a Stronger Hypothesis

Our results so far only separate k-tt-autoreducibility from $(k-2)$-T-autoreducibility for $k \geq 3$ under the genericity hypothesis. In this section we show that a stronger hypothesis separates k-tt-autoreducibility from $(k-1)$-T-autoreducibility, for all $k \geq 2$. We note that separating k nonadaptive queries from $k-1$ adaptive queries is an optimal separation of bounded query reducibilities.

First we consider 2-tt-autoreducibility versus 1-tt-autoreducibility (equivalently, 1-T-autoreducibility). Pavan and Selman [14] showed that if NP ∩ coNP contains a DTIME(2^{n^ϵ})-bi-immune set, then 2-tt-completeness is different from 1-tt-completeness for NP. We show under the stronger hypothesis that NP ∩ coNP contains a p-generic set, we can separate the autoreducibility notions.

▶ **Theorem 4.1.** *If* NP ∩ coNP *has a* p*-generic language, then there exists a* 2-tt*-complete set for* NP *that is* 2-tt*-autoreducible, but neither* 1-tt*-complete nor* 1-tt*-autoreducible.*

Proof. Assume $G \in \text{NP} \cap \text{coNP}$ is p-generic, and let $A = (G \cap \text{SAT})\dot\cup(\overline{G} \cap \text{SAT})$, where $\overline{G}$ is G's complement, and $\dot\cup$ stands for disjoint union. We implement disjoint union as $A = (G \cap \text{SAT})0 \dot\cup (\overline{G} \cap \text{SAT})1$. It follows from closure properties of NP and the fact that $G \in \text{NP} \cap \text{coNP}$ that $A \in \text{NP}$. It follows from definition of A that for every x, $x \in \text{SAT} \leftrightarrow (x0 \in A \vee x1 \in A)$. This means $\text{SAT} \leq^{\text{p}}_{2\text{tt}} A$. Therefore A is 2-tt-complete for NP. Since both queries in the above reduction are honest, in fact length increasing, it follows from Lemma 2.1 that A is 2-tt-autoreducible. To get a contradiction assume that A is 1-tt-autoreducible via polynomial-time computable functions h and g. In other words,

$$\forall x.\ A(x) = h(x, A[g(x)]) \tag{9}$$

and $g(x) \neq x$. Let $x = y0$ for some string y, then (9) turns into

$$\forall y.\ G \cap \text{SAT}[y] = h(y0, A[g(y0)]) \tag{10}$$

and $g(y0) \neq y0$. We define a bounded extension function f whenever $\text{SAT}[y] = 1$ as follows.

- Consider the case where $g(y0) = z0$ or $z1$ and $z > y$. If $g(y0) = z0$ then f forces $G[z] = 0$, and if $g(y0) = z1$ then f forces $G[z] = 1$. f also forces $G[y] = 1 - h(y0, 0)$. Since g and h are computable in polynomial time, so is f.
- On the other hand, if $g(y0) = z0$ or $z1$ and $z < y$ then define f such that it forces $G[y] = 1 - h(y0, A[g(y0)])$. Then f polynomial-time computable in this case as well because A may be computed on $g(y0)$ by looking up $G[z]$ from the partial characteristic sequence and deciding $\text{SAT}[z]$ in $2^{O(|z|)}$ time.
- If $g(y0) = y1$ and $h(y0, .) = c$ is a constant function, then define f such that it forces $G[y] = 1 - c$.

If $g(y0) \neq y1 \wedge \text{SAT}[y] = 1$ for infinitely many y, it follows from the p-genericity of G that G has to meet f, but this refutes the autoreduction. Similarly, $g(y0) = y1 \wedge h(y0, .) = const \wedge \text{SAT}[y] = 1$ cannot happen for infinitely many y's. As a result, $(g(y0) = y1 \vee \text{SAT}[y] = 0)$ and $h(y0, .)$ is not constant for all but finitely many y's. If $g(y0) = y1$ then h says either $G \cap \text{SAT}[y] = \overline{G} \cap \text{SAT}[y]$ or $G \cap \text{SAT}[y] = \neg(\overline{G} \cap \text{SAT}[y])$. It is easy to see this implies $\text{SAT}[y]$ has to be 0 or 1, respectively. Based on the facts above, we define Algorithm 1 that decides SAT in polynomial time. This contradicts the assumption that NP ∩ coNP has a p-generic language.

It is proved in [8] that every nontrival 1-tt-complete set for NP is 1-tt-autoreducible, so it follows that A is not 1-tt-complete. ◀

```
input y;
if g(y0) ≠ y1 ∨ h(y0, .) is constant then
    Output NO;
else
    if h(y0, .) is the identity function then
        Output YES;
    else
        Output NO;
    end
end
```

Algorithm 1. A polynomial-time algorithm for SAT

We will show the same hypothesis on NP ∩ coNP separates k-tt-autoreducibilty from $(k-1)$-T-autoreducibility for all $k \geq 3$. First, we show the corresponding separation of completeness notions.

▶ **Theorem 4.2.** *If* NP ∩ coNP *contains a* p*-generic set, then for every* $k \geq 3$ *there exists a* k-tt*-complete set for* NP *that is not* $(k-1)$-T*-complete.*

The proof of Theorem 4.2 will appear in the full version of the paper.

Now we show the same sets separate k-tt-autoreducibility from $(k-1)$-T-autoreducibility.

▶ **Theorem 4.3.** *If* NP ∩ coNP *contains a* p*-generic set, then for every* $k \geq 3$ *there exists a* k-tt*-complete set for* NP *that is* k-tt*-autoreducible, but is not* $(k-1)$-T*-autoreducible.*

Proof. Assume $G \in \mathrm{NP} \cap \mathrm{coNP}$ is p-generic, and let $G_m = \{x \mid xz_m \in G\}$ for $1 \leq m \leq k$ where $z_1, ..., z_k$ are the first k strings of length k as before. Define

$$A = \Big[\bigcup_{m=1}^{k-1} \{xz_m \mid x \in G_m \cap \mathrm{SAT}\} \Big] \cup \{xz_k \mid x \in \big[\cap_{m=1}^{k-1} \overline{G_m} \big] \cap \mathrm{SAT}\} \tag{11}$$

We showed that $\mathrm{SAT} \leq^{\mathrm{p}}_{k-\mathrm{tt}} A$ via length-increasing queries, therefore by Lemma 2.1 A is k-tt-autoreducible. For a contradiction, assume that A is $(k-1)$-T-autoreducible. This means there exists an oracle Turing machine M such that

$$\forall x.\ A[x] = M^A(x) \tag{12}$$

M runs in polynomial time, and on every input x makes at most $k-1$ queries, none of which is x. Given n and $X \upharpoonright 0^n$, we define a function α as follows.
If $w = vz_p$ and $|w| < n + k$ then

$$\alpha(w) = \begin{cases} X[w] \wedge \mathrm{SAT}[v] & \text{if } 1 \leq p \leq k-1 \\ \big[\bigwedge_{l=1}^{k-1} (1 - X[vz_l]) \big] \wedge \mathrm{SAT}[v] & \text{if } p = k \\ 0 & \text{otherwise} \end{cases}$$

It is easy to see that if $X \upharpoonright 0^n = G \upharpoonright 0^n$ then $\alpha(w) = A[w]$.
If $w = vz_p$ and $|w| \geq n + k$, α is defined as:

$$\alpha(w) = \begin{cases} 1 & \text{if } v = 0^n \wedge 2 \leq p \leq k-1 \\ 0 & \text{if } v = 0^n \wedge p = k \\ 0 & \text{otherwise} \end{cases}$$

Note that α is not defined on 0^{n+k}, but that is fine because we are using α to compute $A[w]$ for w's that are queried when the input is 0^{n+k}, therefore 0^{n+k} will not be queried. Later we will define the extension function f in a way that if G meets f at 0^n then $\alpha(w) = A[w]$ for all queries.

Before defining f, we run M on input 0^{n+k} with α as the oracle instead of A, and define QUERY to be the set of all queries made in this computation. We know that M makes at most $k-1$ queries, therefore $|\text{QUERY}| \leq k-1$. This implies that for every $v \neq 0^n$ which is the value of some element of QUERY one of the following cases must happen:

1. $vz_k \notin \text{QUERY}$
2. $vz_k \in \text{QUERY}$ and $\exists 1 \leq l \leq k-1 \,.\, vz_l \notin \text{QUERY}$

Given n and $X \upharpoonright 0^n$, $f(X \upharpoonright 0^n)$ is defined as follows if $\text{SAT}[0^n] = 1$.

For every v which is the value of some element of QUERY,

1. If $v = 0^n$, then add $(vz_2, 1), ..., (vz_{k-1}, 1)$ to f. In other words, f forces $G[0^n z_i] = 1$ for $2 \leq i \leq k-1$.
2. If $v \neq 0^n$ and $vz_k \notin \text{QUERY}$, then add $(vz_1, 0), ..., (vz_{k-1}, 0)$ to f.
3. If $v \neq 0^n$ and $vz_k \in \text{QUERY}$, then there must be some $1 \leq l \leq k-1$ such that $vz_l \notin \text{QUERY}$. In this case f forces $G[vz_i] = 0$ for every $1 \leq i \leq k-1$ except when $i = l$ for which we force $G[vz_l] = 1$.

To complete the diagonalization we add one more pair to f which is $(0^{n+k}, 1 - M^{\alpha}(0^n))$. It is straightforward, and similar to what has been done in the previous theorem, to show that if G meets f at 0^n for some n then α and A agree on every element of QUERY. Therefore $M^{\alpha}(0^n) = M^A(0^n)$, which results in a contradiction. It only remains to show that G meets f at 0^n for some n. This depends on the details of the encoding used for SAT. If $\text{SAT}[0^n] = 1$ for infinitely many n's, then f satisfies the conditions in Lemma 2.2. Therefore G has to meet f at 0^n for some n. On the other hand, if $\text{SAT}[0^n] = 0$ for almost all n, then we redefine A as:

$$A = \left[\bigcup_{m=1}^{k-1} \{xz_m \mid x \in G_m \cup \text{SAT}\} \right] \cup \{xz_k \mid x \in \left[\cup_{m=1}^{k-1} \overline{G_m} \right] \cup \text{SAT}\} \tag{13}$$

It can be proved, in a similar way and by using the assumption that $\text{SAT}[0^n] = 0$ for almost all n, that A is k-tt-complete, k-tt-autoreducible, but not $(k-1)$-T-autoreducible. ◀

5 Conclusion

We conclude with a few open questions.

For some k, is there a k-tt-complete set for NP that is not btt-autoreducible? We know this is true for EXP [5], so it may be possible to show under a strong hypothesis on NP. We note that by Lemma 2.1 any construction of a k-tt-complete set that is not k-tt-autoreducible must not be honest k-tt-complete. In fact, the set must be complete under reductions that are neither honest nor dishonest. On the other hand, for any $k \geq 3$, proving that all k-tt-complete sets for NP are btt-autoreducible would separate NP $\neq$ EXP.

Are the 2-tt-complete sets for NP 2-tt-autoreducible? The answer to this question is yes for EXP [7], so in this case a negative answer for NP would imply NP $\neq$ EXP. We believe that it may be possible to show the 2-tt-complete sets are nonuniformly 2-tt-autoreducible under the Measure Hypothesis – first show they are nonuniformly 2-tt-honest complete as an extension of [9, 6].

Nguyen and Selman [13] showed there is T-complete set for NEXP that is not tt-autoreducible. Can we do this for NP as well? Note that Hitchcock and Pavan [9] showed there is a T-complete set for NP that is not tt-complete.

Acknowledgment. We thank A. Pavan for extremely helpful discussions.

References

1 K. Ambos-Spies. P-mitotic sets. In *Logic and Machines: Decision Problems and Complexity, Proceedings of the Symposium "Rekursive Kombinatorik" held from May 23-28, 1983 at the Institut für Mathematische Logik und Grundlagenforschung der Universität Münster/Westfalen*, pages 1–23, 1983. doi:10.1007/3-540-13331-3_30.

2 K. Ambos-Spies and L. Bentzien. Separating NP-completeness notions under strong hypotheses. *Journal of Computer and System Sciences*, 61(3):335–361, 2000.

3 K. Ambos-Spies, H. Fleischhack, and H. Huwig. Diagonalizations over polynomial time computable sets. *Theoretical Computer Science*, 51:177–204, 1987.

4 R. Beigel and J. Feigenbaum. On being incoherent without being very hard. *Computational Complexity*, 2:1–17, 1992.

5 H. Buhrman, L. Fortnow, D. van Melkebeek, and L. Torenvliet. Separating complexity classes using autoreducibility. *SIAM Journal on Computing*, 29(5):1497–1520, 2000.

6 H. Buhrman, B. Hescott, S. Homer, and L. Torenvliet. Non-uniform reductions. *Theory of Computing Systems*, 47(2):317–341, 2010. doi:10.1007/s00224-008-9163-5.

7 H. Buhrman and L. Torenvliet. A Post's program for complexity theory. *Bulletin of the EATCS*, 85:41–51, 2005.

8 C. Glaßer, M. Ogihara, A. Pavan, A. L. Selman, and L. Zhang. Autoreducibility, mitoticity, and immunity. *J. Comput. Syst. Sci.*, 73(5):735–754, 2007. doi:10.1016/j.jcss.2006.10.020.

9 J. M. Hitchcock and A. Pavan. Comparing reductions to NP-complete sets. *Information and Computation*, 205(5):694–706, 2007. doi:10.1016/j.ic.2006.10.005.

10 J. M. Hitchcock and A. Pavan. Hardness hypotheses, derandomization, and circuit complexity. *Computational Complexity*, 17(1):119–146, 2008. doi:10.1007/s00037-008-0241-5.

11 R. E. Ladner, N. A. Lynch, and A. L. Selman. A comparison of polynomial-time reducibilities. *Theoretical Computer Science*, 1(2):103–123, 1975.

12 J. H. Lutz and E. Mayordomo. Cook versus Karp-Levin: Separating completeness notions if NP is not small. *Theoretical Computer Science*, 164(1–2):141–163, 1996.

13 D. T. Nguyen and A. L. Selman. Non-autoreducible sets for NEXP. In *31st International Symposium on Theoretical Aspects of Computer Science*, pages 590–601, 2014. doi:10.4230/LIPIcs.STACS.2014.590.

14 A. Pavan and A. L. Selman. Bi-immunity separates strong NP-completeness notions. *Information and Computation*, 188(1):116–126, 2004.

15 B. Trakhtenbrot. On autoreducibility. *Dokl. Akad. Nauk SSSR*, 192(6):1224—1227, 1970. Translation in Soviet Math. Dokl. 11(3): 814–817, 1970.

A Randomized Polynomial Kernel for Subset Feedback Vertex Set

Eva-Maria C. Hols[1] **and Stefan Kratsch**[2]

1 **Institut für Informatik, Universität Bonn, Germany**
hols@cs.uni-bonn.de
1 **Institut für Informatik, Universität Bonn, Germany**
kratsch@cs.uni-bonn.de

Abstract

The SUBSET FEEDBACK VERTEX SET problem generalizes the classical FEEDBACK VERTEX SET problem and asks, for a given undirected graph $G = (V, E)$, a set $S \subseteq V$, and an integer k, whether there exists a set X of at most k vertices such that no cycle in $G - X$ contains a vertex of S. It was independently shown by Cygan et al. (ICALP '11, SIDMA '13) and Kawarabayashi and Kobayashi (JCTB '12) that SUBSET FEEDBACK VERTEX SET is fixed-parameter tractable for parameter k. Cygan et al. asked whether the problem also admits a polynomial kernelization.

We answer the question of Cygan et al. positively by giving a randomized polynomial kernelization for the equivalent version where S is a set of edges. In a first step we show that EDGE SUBSET FEEDBACK VERTEX SET has a randomized polynomial kernel parameterized by $|S| + k$ with $\mathcal{O}(|S|^2 k)$ vertices. For this we use the matroid-based tools of Kratsch and Wahlström (FOCS '12). Next we present a preprocessing that reduces the given instance (G, S, k) to an equivalent instance (G', S', k') where the size of S' is bounded by $\mathcal{O}(k^4)$. These two results lead to a polynomial kernel for SUBSET FEEDBACK VERTEX SET with $\mathcal{O}(k^9)$ vertices.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases parameterized complexity, kernelization, subset feedback vertex set

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.43

1 Introduction

In the SUBSET FEEDBACK VERTEX SET (SUBSET FVS) problem we are given an undirected graph $G = (V, E)$, a set of vertices $S \subseteq V$, and an integer k, and have to determine whether there is a set X of at most k vertices that intersects all cycles that contain at least one vertex of S. Clearly, because we can choose $S = V$, this is a generalization of the well-studied FEEDBACK VERTEX SET (FVS) problem where, given G and k, we have to determine whether some set X of at most k vertices intersects *all cycles* in G. FEEDBACK VERTEX SET has been extensively studied in parameterized complexity: It is known to be fixed-parameter tractable (FPT) with parameter k, i.e., solvable in time $f(k) \cdot |V|^c$, and after a series of improvements the fastest known algorithms take deterministic time $\mathcal{O}^*(3.619^k)$ [11] and randomized time $\mathcal{O}^*(3^k)$ [2]. It is also known to admit a *polynomial kernelization* [1], i.e., there is an efficient algorithm that reduces any instance (G, k) of FVS to an equivalent instance of size polynomial in k; the best known kernelization creates an equivalent instance with $\mathcal{O}(k^2)$ vertices [19].

In 2011, Cygan et al. [3] and Kawarabayashi and Kobayashi [10] independently showed that SUBSET FVS is FPT. The algorithm of Cygan et al. runs in time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$, while the one of Kawarabayashi and Kobayashi runs in time $\mathcal{O}(f(k) \cdot n^2 m)$. Wahlström [21] then gave the first single-exponential algorithm with running time $4^k \cdot n^{\mathcal{O}(1)}$; an algorithm with


© Eva-Maria C. Hols and Stefan Kratsch;
licensed under Creative Commons License CC-BY
33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 43; pp. 43:1–43:14

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


subexponential dependence on k is ruled out under the Exponential-Time Hypothesis (e.g., because SUBSET FVS generalizes VERTEX COVER). More recently, Lokshtanov et al. [13] gave algorithms with deterministic time $2^{\mathcal{O}(k \log k)} \cdot (n+m)$ and randomized time $\mathcal{O}(25.6^k \cdot (n+m))$.

Cygan et al. [3] ask whether the SUBSET FVS problem also admits a polynomial kernelization and suggest that the matroid-based tools of Kratsch and Wahlström [12] could be applicable. The latter work uses representative sets of independent sets in matroids to obtain, amongst others, polynomial kernels for s-MULTIWAY CUT and DELETABLE TERMINAL MULTIWAY CUT (DTMWC) with $\mathcal{O}(k^{s+1})$ and $\mathcal{O}(k^3)$ vertices, respectively. In MULTIWAY CUT we are given a graph $G = (V, E)$, a set $T \subseteq V$ of terminals, and an integer k and have to determine whether deletion of at most k non-terminal vertices separates all terminals. In s-MULTIWAY CUT the terminal set has size at most s, and in DTMWC we are also allowed to delete terminals (which is essentially the same as restricting terminals to be degree one).

Interestingly, Cygan et al. [3] also provide a polynomial-time reduction from MULTIWAY CUT to SUBSET FVS that does not change the parameter value and, hence, is known to imply that SUBSET FVS is at least as hard as MULTIWAY CUT regarding existence of polynomial kernels. Accordingly, MULTIWAY CUT would be the natural next target problem for attempting to find a polynomial kernelization (after s-MULTIWAY CUT and DELETABLE TERMINAL MULTIWAY CUT). It appears, however, that the reduction of Cygan et al. is from DELETABLE TERMINAL MULTIWAY CUT rather than from the more general MULTIWAY CUT, and it is not obvious whether similar ideas could yield a reduction from MULTIWAY CUT to SUBSET FVS.

We apply the matroid-based tools of Kratsch and Wahlström [12] and develop a randomized polynomial kernelization that reduces instances (G, S, k) of SUBSET FVS to equivalent instances with at most $\mathcal{O}(k^9)$ vertices; this is our main result. Like Cygan et al. [3] we also work on EDGE SUBSET FVS where S is a set of edges of G and X needs to intersect all cycles that contain at least one edge of S; EDGE SUBSET FVS and SUBSET FVS are equivalent [3]. The result is obtained in two parts. In the first part (Section 3) we establish a randomized polynomial kernelization for EDGE SUBSET FVS parameterized by $|S| + k$ that reduces to equivalent instances with at most $\mathcal{O}(|S|^2 k)$ vertices. Note that nontrivial instances have $k < |S|$ since one could otherwise remove S by deleting one endpoint of each edge in S. Thus, parameter $|S|$ suffices, but $\mathcal{O}(|S|^2 k)$ gives a tighter overall bound than $\mathcal{O}(|S|^3)$.

At high level, this part is similar to the polynomial kernelization for DELETABLE TERMINAL MULTIWAY CUT. We show that certain solutions X, later called *dominant* solutions, allow particular path packings in the underlying graph G. For DTMWC this is achieved by a fairly simple replacement argument for solutions X that are not sufficiently well connected to connected components of $G - X$. For EDGE SUBSET FVS the endpoints $T = V(S)$ of edges in S can be regarded as terminals, but this gives a different separation property: Solutions X need not generate many connected components in $G - X$ since only S-cycles need to be prevented, and components may contain many vertices of T. Rather, in $G - X$ there must be a tree-like (or forest-like) structure with components without S-edges playing the role of nodes and with edges given by S. Nevertheless, using the tree-like structure, a replacement argument can be found, implying that dominant solutions must create many components in $(G - X) - S$ containing vertices of T and be well connected to them. This allows to set up a gammoid on $G - S$ with sources T and apply, as in [12], a result of Lovász [14] on representative sets in (linear) matroids that is then guaranteed to generate a superset of X. Randomization is only needed to generate a matrix representation for the gammoid.

In the second part (Section 4) we give a (deterministic) polynomial-time preprocessing that, given an instance (G, S, k) of EDGE SUBSET FVS, returns an equivalent instance (G', S', k') with $k' \leq k$ and $|S'| \in \mathcal{O}(k^4)$. Together with the randomized kernelization from the first part this implies the claimed randomized kernelization to $\mathcal{O}(k^9)$ vertices.

A reduction of the number of S-edges is also a crucial ingredient in the FPT algorithm for EDGE SUBSET FVS by Cygan et al. [3]. They achieve $|S| \in \mathcal{O}(k^3)$, but it is in a slightly more favorable setting: Using iterative compression, it suffices to solve the task of finding a solution X' of size k when given a solution X of size $k+1$. (This is well known in parameterized complexity, and we prefer not to repeat it here.) Considering some unknown solution X' of size k, one can guess the intersection D of X' with X, by trying all $\mathcal{O}(2^{k+1})$ possibilities. For the correct guess $D = X' \cap X$, the remaining problem is to find for $(G - D, S \setminus D, k - |D|)$ a solution Z' of size at most $k - |D|$ that is disjoint from $Z = X \setminus D$, since $Z' = X' \setminus D$ would be such a solution; here $S \setminus D$ denotes the set of edges in S with no endpoint in D. Cygan et al. make the nice observation that the guessing also allows to assume that there is no other solution X' with an even larger intersection with X.

In contrast, we cannot afford to run iterative compression for a kernelization to get a starting solution of size $k+1$ and, as is common, we have to start with an approximate solution Z, which can be assumed to be of size at most $8k$ using an 8-approximation algorithm of Even et al. [6]. The idea of guessing the intersection of an optimal solution with Z is infeasible regarding both time and the number of created instances. Thus, while several structures like z-flowers or disjoint x,y-paths containing S-edges appear in both approaches, many things have to be handled differently. For example, having $k+2$ disjoint x,y-paths containing S-edges for $x, y \in Z$ implies that one of x and y must be in every solution of size k; Cygan et al. can stop here because the solution would not be disjoint from Z; we need to instead store the information about x and y to later detect S-edges that can be safely removed. Like Cygan et al., we also use Gallai's A-path Theorem but we avoid the 2-expansion lemma by using the properties of a blocking set of size at most $2k$ differently. Moreover, we observe that z-flowers can be found by solving a matroid parity instance on an appropriate gammoid; this can be done in deterministic polynomial time using a specialized matroid parity algorithm by Tong et al. [20].

Proofs omitted in this extended abstract can be found in Hols and Kratsch [9].

2 Preliminaries

We use standard graph notation, mostly following Diestel [5]. All graphs are undirected and may contain multi-edges and loops; accordingly, they may contain cycles of length one and two (formed by loops and multi-edges, respectively.) An edge $e \in E$ is called a *bridge* if $(V, E \setminus \{e\})$ has more connected components than G. For a set $X \subseteq V$, let $G[X]$ denote the subgraph of G induced by X and let $N_G(X)$ denote the neighborhood of X in G, i.e., $N_G(X) = \{v \in V \setminus X \mid \exists u \in X \colon \{u, v\} \in E\}$. Given two sets $X, Y \subseteq V$, by $E(X, Y)$ we denote the set of edges that have one endpoint in X and one endpoint in Y. For a set $E' \subseteq E$ of edges let $V(E')$ be the set of vertices that are incident with at least one edge in E'. For $X \subseteq V$ and $F \subseteq E$ we shorthand $G - X$ for $G[V \setminus X]$ and $G - F$ for $(V(G), E(G) \setminus F)$; if $X = \{x\}$ then we may also write $G - x$ instead of $G - \{x\}$. Note that the graph $(G - X) - F$ is the same graph as the graph $(G - F) - X$ and we will drop the parentheses.

For $A \subseteq V$ a path with endpoints in A and internal vertices not in A is called an *A-path*. The following theorem about A-paths was already used by Cygan et al. [3] for SUBSET FVS and in the quadratic kernelization for FEEDBACK VERTEX SET by Thomassé [19].

▶ **Theorem 1** (Gallai [8]). *Let $A \subseteq V$ and $k \in \mathbb{N}$. If the maximum number of vertex-disjoint A-paths is strictly less than $k+1$, then there exists a set $B \subseteq V$ of at most $2k$ vertices that intersect every A-path.*

In particular it is possible to find either $(k+1)$-disjoint A-paths or a set B that intersects all A-paths in polynomial time. This follows from Schrijver's proof of Gallai's theorem [18].

Let (G, S, k) be an instance of the EDGE SUBSET FVS problem. We call a cycle C an *S-cycle*, if at least one edge of S is contained in C. Let x be a vertex of V. A set $\{C_1, C_2, \ldots, C_t\}$ of S-cycles that contain x is called an *x-flower* of order t, if the sets of vertices $C_i \setminus \{x\}$ are pairwise disjoint. Note that if there exists a x-flower of order at least $k+1$, then the vertex x must be in every solution for (G, S, k), if one exists. A set $B \subseteq V \setminus \{x\}$ of size t is called an *x-blocker* of size t, if each S-cycle through x also contains at least one vertex of B.

Matroids, gammoids, and representative sets. A matroid $M = (U, \mathcal{I})$ consists of a finite set U and a family $\mathcal{I}$ of subsets of U, called *independent sets*, fulfilling the following properties: (i) $\emptyset \in \mathcal{I}$; (ii) if $X \subseteq Y$ and $Y \in \mathcal{I}$ then also $X \in \mathcal{I}$; and (iii) if $X, Y \in \mathcal{I}$ with $|X| < |Y|$ then there exists $y \in Y \setminus X$ such that $X \cup \{y\} \in \mathcal{I}$.

The *rank* of of a matroid M, denoted by $r(M)$, is the size of the largest independent set of the matroid M.

Let A be a matrix over an arbitrary field F. Let U be the set of columns of A and let $\mathcal{I}$ be the family of all sets $X \subseteq U$ of columns that are linearly independent over F. Then $M = (U, \mathcal{I})$ is a matroid, called the *linear matroid* or *vector matroid* of A, and we say that A *represents* M. If $M = (U, \mathcal{I})$ is representable over some field, then it is also representable by an $r(M) \times |U|$ matrix; by Gaussian elimination we can always reduce a representing matrix for M to one with $r(M)$ many rows (cf. [15]). Let $M_1 = (U_1, \mathcal{I}_1)$ and $M_2 = (U_2, \mathcal{I}_2)$ be two matroids with $U_1 \cap U_2 = \emptyset$. The *direct sum* $M_1 \oplus M_2$ is a matroid over $U = U_1 \cup U_2$ with independent sets $\mathcal{I} = \{X \subseteq U \mid X \cap U_1 \in \mathcal{I}_1, X \cap U_2 \in \mathcal{I}_2\}$. If A_1 and A_2 represent the two matroids over the same field F, then matrix $A = \mathrm{diag}(A_1, A_2)$ represents $M_1 \oplus M_2$.

Let $G = (V, E)$ be a graph that may have both directed and undirected edges and let $S \subseteq V$. A set $T \subseteq V$ is *linked* to S if there exist $|T|$ vertex-disjoint paths from S to T. Thus every vertex in T is endpoint of a different path from S. It holds that $M = (U, \mathcal{I})$, where $U \subseteq V$ and $\mathcal{I}$ contains all sets $T \subseteq U$ that are linked to S in G, is a matroid [17]. The matroid M is also called the *gammoid* on G with sources S and ground set U; if $U = V$ then M is also called a *strict gammoid*. Marx [15] gave a randomized polynomial-time procedure for finding a matrix representation of a strict gammoid. The error probability can be made exponentially small in the size of the graph. (This is the only source of randomness and error in our kernelization.) A matrix representation for a gammoid for graph $G = (V, E)$ with ground set $U \subsetneq V$ and sources S can be obtained from one for the strict gammoid for G and S by simply deleting columns corresponding to elements of $V \setminus U$.

Let A, B be independent sets in a matroid. We say that A *extends* B if $A \cap B = \emptyset$ and $A \cup B$ is again an independent set. Note that from the independence of $A \cup B$ follows the independence of A and B due to the second matroid property.

▶ **Definition 2.** Let $M = (U, \mathcal{I})$ be a matroid, let $\mathcal{A} \subseteq \mathcal{I}$, and let $q \in \mathbb{N}$. A set $\mathcal{A}' \subseteq \mathcal{A}$ is *q-representative* for $\mathcal{A}$ if for every independent set B of size at most q there is a set $A \in \mathcal{A}$ that extends B if and only if there is also a set $A' \in \mathcal{A}'$ that extends B.

Observe that if $\mathcal{A}'$ is q-representative for $\mathcal{A}$ and there exists a set $A \in \mathcal{A}$ that *uniquely extends* some given independent set I of size at most q, then this implies that $A \in \mathcal{A}'$.

The following theorem of Lovász [14] proves that for any linear matroid there exist small representative sets. It was made algorithmic by Marx [15] and, thus, permits to find representative sets in polynomial time when given a matrix representation of the matroid. A faster algorithm for this task was developed recently by Fomin et al. [7].

▶ **Lemma 3** (Lovász [14], Marx [15]). *Let M be a linear matroid of rank $q+p$, and let $\mathcal{T} = \{I_1, I_2, \ldots, I_t\}$ be a collection of independent sets, each of size p. If $|\mathcal{T}| > \binom{q+p}{p}$, then there is a set $I \in \mathcal{T}$ such that $\mathcal{T} \setminus \{I\}$ is q-representative for $\mathcal{T}$. Furthermore, given a representation A of M, we can find such a set I in $f(q,p) \cdot (\|A\|t)^{\mathcal{O}(1)}$ time.*

3 Randomized polynomial kernelization for parameter $|S| + k$

In this section we present a randomized polynomial kernelization for EDGE SUBSET FVS parameterized by $|S| + k$. Because deletion of one endpoint of each edge in S always constitutes a feasible solution, nontrivial instances have $|S| > k$. Thus, our kernelization also works for parameter $|S|$ alone. However, to achieve a better bound for EDGE SUBSET FVS parameterized by k it is beneficial to give the size in terms of $|S|$ and k rather than $|S|$ alone.

We use representative sets of independent sets of matroids to obtain a kernel of size $\mathcal{O}(|S|^2 k)$. Our approach is similar to the kernelization of DELETABLE TERMINAL MULTIWAY CUT(k) [12]. As in that paper we construct path packings such that certain vertices can be shown to be in a representative set. Note that, unlike for multiway cut-type problems, a solution $X \subseteq V$ will not necessarily create many connected components. Rather, as used also in the FPT algorithm of Cygan et al. [3], it creates a particular tree-like structure in $G - X$. Nevertheless, endpoints of edges in S, denoted $T := V(S)$, will play the role of terminals that need to be separated in a certain way; hence a vertex x in T is called a *terminal*. We will focus on the graph $G - S$, i.e., with edges of S deleted, in which a solution X creates a grouping of (not deleted) terminals into connected components. The structure of these components will be crucial for a replacement argument (Lemma 5) that leads to the required path packing; this constitutes one of the key arguments for our result.

The kernelization consists of four steps. In the first step we show that if an instance is YES then there exists a solution X with a certain path packing from T to X. Then we define an appropriate gammoid to find in a next step a representative set of size $\mathcal{O}(|S|^2 k)$ which is (essentially) a superset of X using Lemma 3. Finally we explain how to reduce the graph G, using the superset of the last step, to obtain an equivalent instance of EDGE SUBSET FVS.

Analyzing solutions. Let (G, S, k) be a yes-instance of EDGE SUBSET FVS $(k + |S|)$. We say that a solution X for (G, S, k) is *dominant*, if it has minimum size and contains a maximal number of vertices from T among solutions of minimum size. The vertices in $X \cap T$ correspond to endpoints of edges in S that we delete and the vertices in $X_0 = X \setminus T$ block all x-y paths with $\{x, y\} \in S_0 = \{e \in S \mid e \cap X = \emptyset\}$, except the one that consists of the edge $\{x, y\}$. We show that X is linked to T in a strong sense.

▶ **Lemma 4.** *Let X be a dominant solution for (G, S, k) and x any vertex in the set $X_0 = X \setminus T$. There exist $|X| + 2$ paths from T to X in $G - S$ that are vertex-disjoint except for three paths ending in vertex x. Moreover, the paths can be chosen in such a way that each connected component of $G - X - S$ is intersected by at most one path.*

We use Hall's Theorem and the lemma below to prove this. For this purpose we consider two graphs $G - X$ and $G - X - S$. We call a connected component K of $G - X - S$ *interesting* if it contains a terminal, i.e., if $T \cap V(K) = (T \setminus X) \cap V(K) \neq \emptyset$, and we say that $x \in X_0$ *sees* a connected component K if x is adjacent to a vertex of K in G. We extend this definition by saying that $Y \subseteq X_0$ sees an interesting component K if at least one vertex $y \in Y$ sees K.

▶ **Lemma 5.** *If X is a dominant solution then every nonempty set $Y \subseteq X_0$ sees at least $|Y| + 2$ interesting components of $G - X - S$.*

Proof. Assume for contradiction that there exists a nonempty set $Y \subseteq X_0$ that sees at most $|Y|+1$ interesting components of $G-X-S$. Let $\mathcal{C}$ denote the set of connected components of $G-X$, let $\mathcal{C}_i \subseteq \mathcal{C}$ denote the set of interesting components seen by Y, and let $\mathcal{C}_o \subseteq \mathcal{C}$ denote the other components seen by Y. We will show that there is an alternative solution $X' = (X \setminus Y) \cup Y'$ that is smaller than X or that contains more vertices of T, contradicting the choice of X as a dominant solution. Let us consider the graphs $G-X$ and $G-(X \setminus Y)$.

We study the structure of $G-(X \setminus Y)$ to find a set Y' that intersects all S-cycles in $G-(X \setminus Y)$. Accordingly we are interested in the structure that is induced by the S-edges in $G-(X \setminus Y)$. To study them we define, for any subgraph $G-Z$ of G, the S-component graph H_Z which has a vertex for each connected component of $G-Z-S$ and for every S-edge e an edge between two (not necessary different) vertices which correspond to the connected components that contain the endpoints of e; note that H_Z can have parallel edges and loops. We say that $G-Z$ is an *S-forest* if the S-component graph H_Z is a forest. Observe that a set Z is a solution if and only if $G-Z$ is an S-forest. Note that vertices that correspond to components without terminals in $G-Z$ are isolated in H_Z because they are not incident with S-edges; e.g., this is true in H_X for non-interesting components of $G-X-S$.

To construct an alternative solution X' we compare the S-component graphs of $G-X$ and $G-(X \setminus Y)$; let $\mathcal{C}'$ denote the set of connected components of $G-(X \setminus Y)-S$. A component in $\mathcal{C}'$ either fully contains some components in $\mathcal{C}_i \cup \mathcal{C}_o$ and additionally it may contain vertices of Y or it is equal to a connected component in $\mathcal{C}$: This follows from the fact that we only delete the subset $X \setminus Y$ of X from $G-S$ instead of X. However, in $G-(X \setminus Y)$ the set of S-edges incident with components in $\mathcal{C}'$ is the same as the set of S-edges incident with $\mathcal{C}$ in $G-X$, because $Y \subseteq X_0 = X \setminus T$ and hence there are no additional vertices of T, i.e., $T \setminus X = T \setminus (X \setminus Y)$. Altogether, $H_{X \setminus Y}$ is obtained from H_X by merging vertices in H_X whose corresponding connected components are connected in $G-(X \setminus Y)-S$. In general, $G-(X \setminus Y)$ will not be an S-forest: The merging of vertices may lead to loops (from S-edges with both ends in the same component) and longer cycles in $H_{X \setminus Y}$.

We will see that deleting at most $|Y|$ edges of S, i.e., deleting a set Y' of at most $|Y|$ endpoints of S-edges, will suffice for $G-((X \setminus Y) \cup Y')$ to be an S-forest, making $(X \setminus Y) \cup Y'$ a valid solution. To see this consider an arbitrary connected component C^+ in $G-(X \setminus Y)$ whose corresponding connected component in $H_{X \setminus Y}$ is not cycle-free. Note that C^+ is a union of connected components in $\mathcal{C}'$ that are connected by S-edges. Therefore C^+ must contain connected components in $\mathcal{C}$ that are seen by Y. Let $C_i^1, \ldots C_i^a \in \mathcal{C}_i$ and $C_o^1, \ldots C_o^b \in \mathcal{C}_o$ be the connected components in $\mathcal{C}$ that are contained in C^+ and that are seen by Y.

In $G-X$ the connected component C^+ may decompose into several separate connected components because we additionally delete the vertices of Y. Since Y sees only components in $\mathcal{C}_i \cup \mathcal{C}_o$ the set C^+ decomposes into at most $a+b$ separate components by deleting Y. Recall that components in $\mathcal{C}_o$ are isolated in $G-X$ and contain no vertices of T and, thus, they do not contribute any S-edges to C^+. It remains to consider the components $C_i^1, \ldots, C_i^a$ that are contained in C^+.

The connected components $C_i^1, \ldots, C_i^a$ are part of at least one connected component in $G-X$. Thus, they correspond to a subforest F of H_X and not deleting Y corresponds to merging a vertices in this forest into $d \geq 1$ new vertices; let F' be the connected subgraph in $H_{X \setminus Y}$ that results from F by this operation. If the subforest F consists of c vertices and, thus, at most $c-1$ S-edges then we obtain $c-a+d$ vertices that are connected by at most $c-1$ edges for F'. It therefore suffices to delete at most $(c-1)-((c-a+d)-1) = a-d \leq a-1$ S-edges, i.e., to delete one endpoint of each of at most $a-1$ S-edges, to obtain a forest-structure in F'. (We cannot delete just *any* $a-1$ edges but we can keep any $c-a+d-1$ S-edges

spanning the $c - a + d$ components and delete the at most $a - 1$ remaining S-edges.)

Overall, we get that a connected component C^+ in $G - (X \setminus Y)$ that fully contains a interesting components from $\mathcal{C}_i$ requires at most $a-1$ vertex deletions of endpoints of S-edges to obtain an S-forest. Since Y sees at most $|Y| + 1$ such components, the worst case is achieved by a single component C^+ containing all $|Y| + 1$ interesting components in $\mathcal{C}_i$; this still costs at most $(|Y| + 1) - 1 = |Y|$ vertex deletions, as claimed.

Let Y' contain all the endpoints of S-edges that we delete to get an S-forest. We know that $|Y'| \leq |Y|$ and thus $|(X \setminus Y) \cup Y'| \leq |X|$. Moreover, by the initial considerations, we know that $X' = (X \setminus Y) \cup Y'$ is a feasible solution as $G - X'$ has the required S-forest. If $|Y'| < |Y|$, including the case that $Y' = \emptyset$, then $|X'| < |X|$ as $Y \neq \emptyset$; this contradicts optimality of X (required for being a dominant solution). If $|Y'| = |Y|$ then $Y' \neq \emptyset$ and X' is an optimal solution that contains more vertices of $T \supseteq Y'$, contradicting the choice of X as a dominant solution. Thus, every nonempty set Y must see at least $|Y| + 2$ connected components, as claimed. ◀

Lemma 4 can now be obtained via Hall's Theorem; the proof is similar to the one for DELETABLE TERMINAL MULTIWAY CUT [12].

Setting up the gammoid. The gammoid M that we use is the direct sum of two gammoids M_1 and M_2. To construct gammoid M_1 we define a graph $G_1 = (V_1, E_1)$ that is obtained from $G - S$ by adding two so called *sink-only copies* v' and v'' for every vertex $v \in V$. A sink-only copy of a vertex v is a vertex v' (or v'') that has a directed edge (u, v') for each edge $\{u, v\}$; these were already used in previous work [12]. Note that adding sink-only copies of vertices does not affect the possible path packings to other vertices since they can only be endpoints of paths; however, they are convenient to capture multiple vertex-disjoint paths that, intuitively, end in the same vertex. Matroid M_1 is defined as the gammoid on G_1 with sources $T = V(S)$ and ground set $V_1 = \{v, v', v'' \mid v \in V\}$; note that the sink-only copies of vertices in T are not sources of M_1. The rank of matroid M_1 is $|T|$, because the set of all trivial paths is independent and at most $|T|$ vertices can be linked to T.

Matroid M_2 is the gammoid on the directed graph $G_2 = K_{k,n} = (S_2 \dot\cup \hat{V}, E_2)$ with sources S_2 and ground set $\hat{V} = \{\hat{v} \mid v \in V\}$; the edges in E_2 are directed from S_2 to $\hat{V}$. In other words, gammoid M_2 is a uniform matroid and a (deterministic) matrix representation could also be obtained by using a Vandermonde matrix. The rank of M_2 is $k = |S_2|$ as no more than $|S_2|$ vertices can be linked to S_2 and every set of at most k vertices of $\hat{V}$ is linked to S_2.

For the application of Lemma 3 we will use the matroid $M = M_1 \oplus M_2$, which has rank $|T| + k$. Representations A_1 and A_2 for both M_1 and M_2 can be computed by a randomized polynomial-time algorithm with exponentially small error chance [15]; hence we get a representation for M by $\mathrm{diag}(A_1, A_2)$, i.e., the block-diagonal matrix with blocks A_1 and A_2. We may assume that A_1 has $|T|$ rows and A_2 has k rows (cf. [15]).

Applying the representative set lemma. Let $\mathcal{T} := \{\{v', v'', \hat{v}\} \mid v \in V\}$. For clarity, by the above notation, this means that $v', v'' \in V_1$ and $\hat{v} \in \hat{V}$ for each $v \in V$. Let $\mathcal{T}'$ be obtained by applying Lemma 3 to $\mathcal{T}$ using the above matrix representation for M; we have $|\mathcal{T}'| \in \mathcal{O}((|T|+k)^3) = \mathcal{O}(|S|^3)$. We will see later that we can find a $(|T|+k-3)$-representative set of size $\mathcal{O}(|S|^2 k)$ by a careful look at the proof of Lemma 3, using the fact that M is the direct sum of two gammoids and that all sets $\{v', v'', \hat{v}\}$ in $\mathcal{T}$ have two elements from the first and one element from the second gammoid; a similar argument for getting a smaller representative set was already used by Kratsch and Wahlström [12].

We will prove that for each dominant solution X we have $\{x', x'', \hat{x}\} \in \mathcal{T}'$ for each $x \in X_0 = X \setminus T$. To this end, we show that for each such set $\{x', x'', \hat{x}\}$ there exists an independent set I of size at most $|T| + k - 3$ such that $\{x', x'', \hat{x}\}$ uniquely extends I among triplets in $\mathcal{T}$. Thus, $\{x', x'', \hat{x}\}$ must be in every $(|T| + k - 3)$-representative set $\mathcal{T}'$ of $\mathcal{T}$.

▶ **Lemma 6.** *Let X be a dominant solution for (G, S, k) and let $T = V(S)$. For all $x \in X_0 = X \setminus T$ there exists an independent set I of size at most $|T| + k - 3$ in M such that $\{x', x'', \hat{x}\}$ uniquely extends I.*

We know now that for every vertex $x \in V \setminus T$ that is a vertex in a dominant solution the set $\{x', x'', \hat{x}\}$ is in every $(|T| + k - 3)$-representative set $\mathcal{T}'$. If we define $V(\mathcal{T}') = \{v \mid \{v', v'', \hat{v}\} \in \mathcal{T}'\}$ then this implies that $X_0 \subseteq V(\mathcal{T}')$ for each dominant solution X. Thus, every dominant solution X is contained in $V(\mathcal{T}') \cup T$.

Shrinking the input graph to $\mathcal{O}(|V(\mathcal{T}') \cup T|)$ vertices. In the previous parts we have shown that if there exists a solution for (G, S, k), then there exists a solution that is completely contained in $W := V(\mathcal{T}') \cup T$. Using this we can make all vertices in $V \setminus W$ undeletable. We achieve this by applying the so-called torso operation to vertex set W in G; let $G' = \text{torso}(G, W)$. By definition of $\text{torso}(G, W)$, the resulting graph G' has vertex set W and is derived from $G[W]$ by making each pair $\{u, v\} \subseteq W$ adjacent if there is a u,v-path in G with internal vertices from $V \setminus W$. Note that we do not create double edges or loops in G' and that all edges of S are preserved in G' because $T \subseteq W$. (The same can be achieved by iteratively selecting a vertex $v \in V \setminus W$, making its neighbors a clique, and deleting v.)

▶ **Lemma 7.** *(G', S, k) has a solution if and only if (G, S, k) has a solution.*

It follows from Lemma 7 that (G', S, k) is an equivalent instance and the graph of this instance contains at most $|W|$ vertices. The correctness of Lemma 7 follows from the fact that the torso operation preserves the separators that are contained in W (cf. [16]).

So far we have a kernelization that creates an equivalent instance (G', S, k) such that G' has $|W|$ vertices. As mentioned above, Lemma 3 guarantees that $|W| \in \mathcal{O}(|S|^3)$ and this implies a polynomial kernel for EDGE SUBSET FVS parameterized by $|S|$. If we use the fact that the gammoid M is the direct sum of two gammoids M_1 and M_2, and that all sets $\{v', v'', \hat{v}\} \in \mathcal{T}$ contain exactly two elements of M_1 and one element of M_2, then we can prove that $|W| \in \mathcal{O}(|S|^2 k)$, this is an improvement for all nontrivial instances with $k < |S|$.

▶ **Lemma 8.** *Let $M = M_1 \oplus M_2$ be the gammoid of rank $|T| + k$ as defined above and $\mathcal{T} = \{I_1, I_2, \ldots, I_t\}$ be the set of independent sets of M that we use for the kernelization. Let A be represented by $\text{diag}(A_1, A_2)$ as above. If $|\mathcal{T}| > \binom{|T|}{2} \cdot \binom{k}{1}$, then there exists a set $I \in \mathcal{T}$ such that $\mathcal{T} \setminus \{I\}$ is $(|T| + k - 3)$-representative for $\mathcal{T}$.*

The proof of Lemma 8 is similar to Marx [15, Lemma 4.2]. We additionally use the fact that M is the direct sum of two gammoids to get that the vectors in the exterior algebra which represent the sets in $\mathcal{T}$ span a space of smaller dimension. As mentioned above, Marx [15] showed that one can find in randomized polynomial-time a matrix with $r(M)$ rows that represents a given gammoid M. We can make this proof algorithmic in the same way. Combined with Lemma 8 it follows that we can find a $(|T| + k - 3)$-representative subset $\mathcal{T}'$ of $|\mathcal{T}|$ of size at most $\binom{|T|}{2} \cdot \binom{k}{1} \in \mathcal{O}(|S|^2 k)$. This implies a randomized polynomial kernel with $\mathcal{O}(|S|^2 k)$ vertices for EDGE SUBSET FVS parameterized by $|S|$ and k.

4 Reducing the size of S

We have seen that EDGE SUBSET FVS parameterized by $|S|$ and k has a polynomial kernel. Now the goal is to reduce the size of the set S until $|S|$ is polynomially bounded in k. This will lead to a polynomial kernel of EDGE SUBSET FVS parameterized by k.

To begin, we do some initial modifications to ensure that we can always find a solution of size at most k that contains no vertex of the set $V(S)$, if one exists. For this we first delete all vertices $v \in V$ with the property that $e = \{v, v\} \in S$ is a loop in G and we decrease the value k by one. Next we delete all remaining loops. We also reduce the number of edges between two vertices $v, w \in V(G)$. If no edge that is incident to v and w is contained in the set S, then we delete all except one edge. On the other hand, if at least one edge between v and w is contained in S, then we delete all except two edges. One of these edges is contained in S and the other not. In the next step we add for every edge $e = \{v, w\} \in S$ two new vertices v_e, u_e to the graph, subdivide the edge e into three edges $\{v, v_e\}, \{v_e, w_e\}, \{w_e, w\}$, and edit S by replacing edge e by the edge $\{v_e, w_e\}$ in S. If a solution X of EDGE SUBSET FVS contains a vertex $x_e \in V(S)$, then we can instead add the vertex x to X and delete x_e from X, because every cycle that contains vertex x_e also contains vertex x; hence we can always find an optimal solution that is disjoint from $V(S)$.

Let (G, S, k) be an instance of EDGE SUBSET FVS, such that G is a graph with the above properties. Analogous to the paper of Cygan et al. [3] we consider a solution Z of the EDGE SUBSET FVS, with the difference that our solution is an 8-approximation of the problem, to reduce the size of S. Even et al. [6] show that there exists an 8-approximation algorithm for SUBSET FVS. Since SUBSET FVS and EDGE SUBSET FVS are equivalent (cf. [3]), we can compute in polynomial time an 8-approximation for EDGE SUBSET FVS and we can assume that $Z \cap V(S) = \emptyset$. If $|Z| > 8k$, then we can stop immediately because no solution of size at most k can exist. On the other hand, if $|Z| \leq k$, then Z is a solution and we are done.

The set Z is a feasible solution to EDGE SUBSET FVS on $(G, S, |Z|)$. This implies that every edge $e \in S$ is a bridge in $G - Z$. In a next step we also remove all edges in S from $G - Z$. Every connected component in $G - Z - S$ contains no edge from S and, following Cygan et al. [3], we call such a component a *bubble*. We denote the set of bubbles by $\mathcal{D}_Z$ and define a graph $H_Z = (\mathcal{D}_Z, E_{\mathcal{D}_Z})$ whose vertices are bubbles and with bubbles I and J being adjacent, i.e., $\{I, J\} \in E_{\mathcal{D}_Z}$, if and only if the components I and J are connected by an edge from S. The graph H_Z is a forest, because Z is a solution for $(G, S, |Z|)$ and a cycle in H_Z would give rise to an S-cycle in $G - Z$. Similarly, no two bubbles can be connected by more than one edge of S. By V_I we denote the vertices that are contained in bubble I. Since $|E(V_I, V_J) \cap S| \leq 1$ for all $I, J \in \mathcal{D}_Z$ and equality holds if and only if $\{I, J\} \in E_{\mathcal{D}_Z}$, we can associate an edge $e = \{I, J\} \in E_{\mathcal{D}_Z}$ with the one edge $e_S = \{v_I, v_J\}$ in $E(V_I, V_J) \cap S$. If we add the vertex set Z and all edges $\{z, I\}$ with the property that $z \in Z, I \in \mathcal{D}_Z$ and $E(z, V_I) \neq \emptyset$ to the graph H_Z we obtain a graph H_Z^+ that contains S-cycles. Note that every S-cycle must contain a vertex of the set Z. We partition the set of bubbles according to the number of bubbles they are connected with.

▶ **Definition 9.** A bubble $I \in \mathcal{D}_Z$ is called (i) solitary, if $\deg_{H_Z}(I) = 0$; (ii) leaf, if $\deg_{H_Z}(I) = 1$; and (iii) inner, if $\deg_{H_Z}(I) \geq 2$. By $\mathcal{D}_Z^s, \mathcal{D}_Z^l, \mathcal{D}_Z^i$ we denote the corresponding sets of of bubbles.

Let $X \subseteq V \setminus V(S)$ be a superset of Z. We define H_X, $H_X^+, \mathcal{D}_X$ and $E_{\mathcal{D}_X}$ analogously to $H_Z, H_Z^+, \mathcal{D}_Z$ and $E_{\mathcal{D}_Z}$. Observe that the number of edges in S is at most $|\mathcal{D}_Z \setminus \mathcal{D}_Z^s|$, because H_Z is a forest, any two bubbles are connected by at most one S-edge, and $V(S) \cap Z = \emptyset$.

So far our setup is essentially the same as the one used by Cygan et al. [3]. However, instead of an 8-approximate solution they use the framework of iterative compression, which provides a solution Z of size $k + 1$ and leaves them with the task of reducing the number of S-edges for the problem of finding a solution Z^* that is *disjoint* from Z. Moreover, it suffices for them to consider the case that every feasible solution (if one exists) is disjoint from Z. In this setting they are able to reduce to an equivalent instance (or find that some

assumption was violated) with only $\mathcal{O}(k^3)$ edges in S. Thus, while many relevant structures like z-flowers or parallel x-y paths containing S-edges are the same, many things have to be handled differently. In particular, if we find that at least one out of two vertices $x, y \in Z$ must be in the solution then we cannot stop (using the maximality condition) but need to continue and use this information in a more direct way.

During the reduction we detect certain pairs $\{x, y\}$ of different vertices with the property that each solution of size at most k must contain at least one of the vertices (if one exists). We store this fact as a *pair-constraint*. We keep and enforce this information in the final instance, unless we decide earlier to delete x or y. By $\mathcal{P}$ we denote the set of pair-constraints that we have found so far. We can interpret this set as a set of edges and by $V(\mathcal{P})$ we denote all vertices that are contained in a pair-constraint. Note that vertices from the set $V(S)$ are never contained in a pair-constraint from $\mathcal{P}$, because there always exists a solution that is disjoint from $V(S)$. We need the set $\mathcal{P}$ to detect edges in S that may be safely deleted. To this end, we generalize the EDGE SUBSET FVS problem by adding a set of pair-constraints $\mathcal{P}$ to the input; we call this problem PAIR-CONSTRAINED EDGE SUBSET FVS.

PAIR-CONSTRAINED EDGE SUBSET FEEDBACK VERTEX SET **Parameter:** k
Input: An undirected graph G, a set $S \subseteq E$ of edges, a set $\mathcal{P}$ of pair-constraints and an integer k.
Question: Does there exist a set $X \subseteq V$ of size at most k such that $G - X$ contains no S-cycle and such that for each pair-constraint $\{x, y\} \in \mathcal{P}$ we have $x \in X$ or $y \in X$?

Clearly, instances (G, S, k) of EDGE SUBSET FVS and $(G, S, \emptyset, k)$ of PAIR-CONSTRAINED EDGE SUBSET FVS are equivalent. Our goal is to reduce the size of S by detecting S-edges that we can delete from S without changing the outcome. This leads to the following definition:

▶ **Definition 10.** Let $(G, S, \mathcal{P}, k)$ be an instance of PAIR-CONSTRAINED EDGE SUBSET FVS. We call an edge $e \in S$ *irrelevant*, if $X \subseteq V(G)$ is a solution for $(G, S, \mathcal{P}, k)$ if and only if X is a solution for $(G, S \setminus \{e\}, \mathcal{P}, k)$.

Note that if two different S-edges e and e' are irrelevant in $(G, S, \mathcal{P}, k)$, then e' is not necessarily irrelevant in $(G, S \setminus \{e\}, \mathcal{P}, k)$. Also, we do not expect to find all irrelevant edges.

The reduction rules. We now present our reduction rules; we assume that always the lowest numbered applicable rule is applied first. Correctness and efficiency of the overall reduction process are deferred to the full version. Let $(G, S, \mathcal{P} = \emptyset, k)$ be an instance for PAIR-CONSTRAINED EDGE SUBSET FVS and let Z be an 8-approximation of this problem with $k < |Z| \leq 8k$ that is disjoint from $V(S)$. In the following the graphs $G - Z$, $G - Z - S$, H_Z, and H_Z^+ are always defined with respect to the current instance $(G, S, \mathcal{P}, k)$ of PAIR-CONSTRAINED EDGE SUBSET FVS. Note that $Z \subseteq V$ and we delete a vertex from Z if we delete the corresponding vertex in V.

Rule 1: If $k < 0$, or if $k = 0$ and there exists an S-cycle, then reduce $(G, S, \mathcal{P}, k)$ to some trivial false instance, i.e. $G' := (\{x\}, \{e = \{x, x\}\})$, $S' := \{e\}$, $\mathcal{P}' = \emptyset$ and $k' := 0$.
Rule 2: Delete all bridges and all connected components not containing any edge from S.
Rule 3: If edge $e \in S$ is a bridge in $(V, E \setminus (S \setminus \{e\}))$, then reduce to $S' = S \setminus \{e\}$.
Rule 4: If vertex $v \in V(\mathcal{P})$ is contained in at least $k + 1$ pair-constraints of $\mathcal{P}$, then we reduce to $G' = G - v$ and $k' = k - 1$.
Rule 5: If $|\mathcal{P}| > k^2$, then reduce $(G, S, \mathcal{P}, k)$ to some trivial false instance.
Rule 6: If there exists a z-flower of order $k + 1$ in G for a vertex $z \in Z$, then we reduce to $G' := G - z$ and $k' := k - 1$.

Rules 2 and 3 ensure that each bubble $I \in \mathcal{D}_Z$ is adjacent to a vertex in Z in graph H_Z^+ and Rules 4 and 5 make sure that $\mathcal{P}$ remains small. For the next rules we need a maximal matching M in H_Z that covers all inner bubbles $\mathcal{D}_Z^i$ in H_Z. Note that two adjacent leaf bubbles I_1, I_2 form a K_2 in H_Z, hence the edge $\{I_1, I_2\} \in E_{\mathcal{D}_Z}$ is contained in every maximal matching in H_Z. We use this matching to detect pair-constraints in Z. To this end we introduce the following definition: Let $e = \{I, J\}$ be an edge in the matching M. We say e *sees* the pair $\{x, y\}$ of different vertices $x, y \in Z$ respectively the vertex $x \in Z$, if $\{I, x\}, \{J, y\} \in E(H_Z^+)$ or $\{I, y\}, \{J, x\} \in E(H_Z^+)$ respectively $\{I, x\}, \{J, x\} \in E(H_Z^+)$.

Rule 7: If at least $(k + 2)$ edges in M see a pair $\{x, y\}$ of different vertices in Z, then we add $\{x, y\}$ to the set of pair-constraints $\mathcal{P}$.

Rule 8: If there exists an edge $e \in M$ such that e sees no single vertex $z \in Z$ and for every pair $\{x, y\}$ seen by e the pair $\{x, y\}$ is a pair-constraint in $\mathcal{P}$, then remove e_S from S and e from M. (Recall: If $e = \{I, J\} \in E(H_Z)$, then e_S is the unique edge in $E(V_I, V_J) \cap S$.)

The matching M is always recomputed if, through application of rules, it does no longer cover every inner bubble or is not maximal when testing whether Rules 7 or 8 apply.

Let $L = \mathcal{D}_Z^l \setminus V(M)$ be the set of leaf bubbles that are not covered by M. Because the matching covers at least all inner bubbles, we know that $|S| \leq 2|M| + |L|$. Therefore we have to find a reduction rule that reduces the number of leaf bubbles in L. Every leaf bubble in L is adjacent to an inner bubble in H_Z, because M covers all leaf bubbles that are not adjacent to an inner bubble. To bound the number of leaf bubbles in L we define for each $z \in Z$ a graph G_z with the help of the following two sets. The first one, $L_z = N_{H_Z^+}(z) \cap L$, is the set of all leaf bubbles I that are adjacent to z in H_Z^+. The other $V_z^i = \{v \in V \mid \exists J \in N_{H_Z^+}(L_z) \colon v \in V_J\}$ consists of all vertices that are contained in an inner bubble that is adjacent to a leaf bubble in L_z. Let $V(G_z) = \{z\} \cup L_z \cup V_z^i$ and

$$E(G_z) = E_{H_Z^+}(z, L_z) \cup \{\{I, w\} \mid \exists I \in L_z, v \in V_I, w \in V_z^i \colon \{v, w\} \in S\} \cup (E(G[V_z^i]) \setminus S).$$

In the graph G_z each leaf bubble $I \in L_z$ is a single vertex. We are not interested in the internal structure of leaf bubbles in L_z, whereas we are interested in the structure of the inner bubbles that are adjacent to the leaf bubbles in L_z. Thus we add the connected component that corresponds to an inner bubble which is adjacent to a bubble in L_z to G_z. In order to apply the concept of flowers and blocking sets in G_z, an edge $e \in E(G_z)$ is an S-edge in G_z if $e = \{I, w\}$ with $I \in L_z$ and $w \in V_z^i$. Note that e is an edge in G_z, because there exists an S-edge $e' = \{v, w\}$ in G with $v \in V_I$.

Since no previous rule is applicable and a z-flower in G_z gives rise to a z-flower in G of same order, one can show, using Gallai's A-path Theorem, that there exists a z-blocker $B_z \subseteq V_z^i \setminus V(S)$ of size at most $2k$ for every vertex $z \in Z$ in G_z. Let $B = \bigcup_{z \in Z} B_z$ be the union of all z-blockers B_z of size at most $2k$. Note that the set L is the union of all sets L_z with $z \in Z$, because every leaf bubble is adjacent to a vertex in Z, i.e., $L = \bigcup_{z \in Z} L_z$.

Because $B \subseteq V_Z^i \setminus V(S)$ we know that $L \subseteq \mathcal{D}_{Z \cup B}^l$; thus we can use $H_{Z \cup B}$ to bound the number of leaf bubbles in L. Let $\boldsymbol{I} = \{J \in \mathcal{D}_{Z \cup B}^i \mid E(L, J) \neq \emptyset\}$ be the set of inner bubbles in $H_{Z \cup B}$ that are adjacent to a leaf bubble in L. Clearly the number of edges between $\boldsymbol{I}$ and L in $H_{Z \cup B}$ equals the number $|L|$. Instead of again using a matching to reduce this number we consider more carefully the properties of these edges (more details in full version). For this we define the property of seeing a pair in a slightly different way. Let $e = \{I, J\}$ be an edge with $I \in \boldsymbol{I}$ and $J \in L$. We say that $e = \{I, J\}$ with $I \in \boldsymbol{I}$ and $J \in L$ sees the pair $\{x, y\}$ of different vertices $x \in Z \cup B$ and $y \in Z$, if $\{I, x\}, \{J, y\} \in E(H_{Z \cup B}^+)$. Observe that a bubble in L is never adjacent to a vertex in B in the graph $H_{Z \cup B}$, because $B \subseteq \bigcup_{z \in Z} V_z^i \setminus V(S)$.

Rule 9: If at least $(k+2)$ edges $\{I_1, J_1\}, \ldots, \{I_l, J_l\}$ with $l \geq k+2$, $I_i \in \boldsymbol{I}$ and $J_i \in L$ for $1 \leq i \leq l$ see a pair $\{x, y\}$ of different vertices, such that $x \in Z \cup B$ is adjacent to I_i, $y \in Z$ is adjacent to J_i for all $i \in \{1, \ldots, l\}$, then we add $\{x, y\}$ to the set of pair-constraints $\mathcal{P}$.

Rule 10: If there exists an edge $e = \{I, J\}$ with $I \in \boldsymbol{I}$ and $J \in L$ such that e sees no single vertex $z \in Z$ and for every pair $\{x, y\}$ seen by e the pair $\{x, y\}$ is a pair-constraint in $\mathcal{P}$, then remove e_S from S, delete J from L and replace I by $I \cup J$ in $\boldsymbol{I}$.

Note that if we delete an edge $e = \{I, J\}$ from S by applying Rule 10, then the consequence is that bubbles I and J are now merged into a single bubble.

If no reduction rule is applicable, then $|M| \in \mathcal{O}(k^3)$ and $|L| \in \mathcal{O}(k^4)$. (A proof of these results is deferred to the full version.) As mentioned above, $|S| = |\mathcal{D}_Z^i| + |\mathcal{D}_Z^l| \leq 2|M| + |L| \in \mathcal{O}(k^4)$, because H_Z is a forest, because there is at most one edge of S between any two bubbles, and because $V(S) \cap Z = \emptyset$.

Finding an equivalent instance for Edge Subset Feedback Vertex Set. Up to now we can only bound the number of edges in S for the PAIR-CONSTRAINED EDGE SUBSET FVS problem. As mentioned above the instance $(G, S, \mathcal{P} = \emptyset, k)$ for PAIR-CONSTRAINED EDGE SUBSET FVS is equivalent to the instance (G, S, k) of EDGE SUBSET FVS. Therefore we only have to show that we can find in polynomial time an instance of EDGE SUBSET FVS that is equivalent to the instance $(G, S, \mathcal{P}, k)$ of PAIR-CONSTRAINED EDGE SUBSET FVS and has at most $\mathcal{O}(k^4)$ S-edges. Let $\{x, y\} \in \mathcal{P}$ be a pair-constraint. If there are two edges between x and y of which at least one is contained in S, then x or y must be in any solution, because xy is an S-cycle. For this reason, the instance $(G', S' = S \cup \mathcal{P}, k)$ of EDGE SUBSET FVS is equivalent to the instance $(G, S, \mathcal{P}, k)$ of PAIR-CONSTRAINED EDGE SUBSET FVS, where G' is created from G by adding one edge $\{x, y\}$ between every two vertices x and y with $\{x, y\} \in \mathcal{P}$ when $\{x, y\} \notin E$ and by adding an edge $\{x, y\}$ between x and y that is also contained in S'; hence there are two edges between x and y with $\{x, y\} \in \mathcal{P}$ in graph G' and we add exactly one edge between x and y to S'. Because we cannot apply Rule 4 or 5 to $(G, S, \mathcal{P}, k)$, we know that $|\mathcal{P}| \leq k^2$. This leads to a bound of $|S| + |\mathcal{P}| \in \mathcal{O}(k^4)$ edges in S' for the EDGE SUBSET FVS problem after the reduction. Together with the kernel with $\mathcal{O}(|S|^2 k)$ vertices for EDGE SUBSET FVS parameterized by $|S|$ and k, we obtain a kernelized instance with $\mathcal{O}(k^9)$ vertices for EDGE SUBSET FVS parameterized by k.

Note that it is no problem that we use in Section 4 the existence of a solution disjoint from $V(S)$ and that we only preserve dominant solutions in Section 3, because the reduction rules in Section 4 as well as the kernelization in Section 3 lead to equivalent instances and because every instance has a dominant solution (if a solution exists).

5 Conclusions

We have shown that the SUBSET FVS problem has a randomized polynomial kernelization using the matroid-based tools of Kratsch and Wahlström [12], positively answering the question of Cygan et al. [3]. As in previous work [12] the error-probability can be made exponentially small without increasing the kernel size. Nevertheless, it would of course be very interesting whether the use of randomization and/or matroids can be avoided. Furthermore, there is quite a gap between $\mathcal{O}(k^9)$ vertices and a lower bound of *size* $\mathcal{O}(k^{2-\varepsilon})$ that is inherited from VERTEX COVER [4], conditioned on non-collapse of the polynomial hierarchy.

Other open problems regarding existence of polynomial kernels, possibly amenable to the matroid tools, are MULTIWAY CUT and DIRECTED FEEDBACK VERTEX SET (DFVS). There is also a directed version of SUBSET FVS, called DIRECTED SUBSET FEEDBACK VERTEX SET, but it generalizes DFVS, whose kernel status has remained open for quite some time now.

References

1 Kevin Burrage, Vladimir Estivill-Castro, Michael R. Fellows, Michael A. Langston, Shev Mac, and Frances A. Rosamond. The undirected feedback vertex set problem has a poly(k) kernel. In *IWPEC 2006*, volume 4169 of *LNCS*, pages 192–202. Springer, 2006. doi:10.1007/11847250_18.

2 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 2011*, pages 150–159. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.23.

3 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Subset feedback vertex set is fixed-parameter tractable. *SIAM J. Discrete Math.*, 27(1):290–309, 2013. doi:10.1137/110843071.

4 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.

5 Reinhard Diestel. Graph theory. *Graduate texts in mathematics*, 2005.

6 Guy Even, Joseph Naor, and Leonid Zosin. An 8-approximation algorithm for the subset feedback vertex set problem. *SIAM J. Comput.*, 30(4):1231–1252, 2000. doi:10.1137/S0097539798340047.

7 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *SODA 2014*, pages 142–151. SIAM, 2014. doi:10.1137/1.9781611973402.10.

8 Tibor Gallai. Maximum-minimum sätze und verallgemeinerte faktoren von graphen. *Acta Mathematica Hungarica*, 12(1-2):131–173, 1961.

9 Eva-Maria C. Hols and Stefan Kratsch. A randomized polynomial kernel for subset feedback vertex set. *CoRR*, abs/1512.02510, 2015. URL: http://arxiv.org/abs/1512.02510.

10 Ken-ichi Kawarabayashi and Yusuke Kobayashi. Fixed-parameter tractability for the subset feedback set problem and the s-cycle packing problem. *J. Comb. Theory, Ser. B*, 102(4):1020–1034, 2012. doi:10.1016/j.jctb.2011.12.001.

11 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Inf. Process. Lett.*, 114(10):556–560, 2014. doi:10.1016/j.ipl.2014.05.001.

12 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *FOCS 2012*, pages 450–459. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.46.

13 Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. Linear time parameterized algorithms for subset feedback vertex set. In *ICALP 2015*, volume 9134 of *LNCS*, pages 935–946. Springer, 2015. doi:10.1007/978-3-662-47672-7_76.

14 László Lovász. Flats in matroids and geometric graphs. *Combinatorial surveys*, pages 45–86, 1977.

15 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.

16 Dániel Marx, Barry O'Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Transactions on Algorithms*, 9(4):30, 2013. doi:10.1145/2500119.

17 Hazel Perfect. Applications of Menger's graph theorem. *J. Math. Anal. Appl.*, 22:96–111, 1968.

18 Alexander Schrijver. A short proof of Mader's sigma-paths theorem. *J. Comb. Theory, Ser. B*, 82(2):319–321, 2001. doi:10.1006/jctb.2000.2029.

19 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Transactions on Algorithms*, 6(2), 2010. doi:10.1145/1721837.1721848.

20 Po Tong, Eugene L. Lawler, and Vijay V. Vazirani. *Solving the weighted parity problem for gammoids by reduction to graphic matching.* Computer Science Division, University of California, 1982.

21 Magnus Wahlström. Half-integrality, lp-branching and FPT algorithms. In *SODA 2014*, pages 1762–1781. SIAM, 2014. doi:10.1137/1.9781611973402.128.

Periods and Borders of Random Words*

Štěpán Holub[1] and Jeffrey Shallit[2]

1 **Department of Algebra, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic**
holub@karlin.mff.cuni.cz

2 **School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1 Canada**
shallit@cs.uwaterloo.ca

Abstract

We investigate the behavior of the periods and border lengths of random words over a fixed alphabet. We show that the asymptotic probability that a random word has a given maximal border length k is a constant, depending only on k and the alphabet size ℓ. We give a recurrence that allows us to determine these constants with any required precision. This also allows us to evaluate the expected period of a random word. For the binary case, the expected period is asymptotically about $n - 1.641$. We also give explicit formulas for the probability that a random word is unbordered or has maximum border length one.

1998 ACM Subject Classification G.2.1 Combinatorics

Keywords and phrases random word, period, word border

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.44

1 Introduction and Notation

A *word* is a finite sequence of letter chosen from a finite alphabet Σ. The periodicity of words is a classical and well-studied topic in both discrete mathematics and combinatorics on words, starting with the classic paper of Fine and Wilf [4] and continuing with the works of Guibas and Odlyzko [6, 7, 5]. For more recent work, see, for example, [8, 15, 12].

We say that a word w has period p if $w[i] = w[i + p]$ for all i that make the equation meaningful. (If $|w| = n$ and one indexes beginning at position 1, this would be for $1 \leq i \leq n - p$.) Trivially every word of length n has all periods of length $\geq n$, so we restrict our attention to periods $\leq n$. The least period is sometimes called *the* period. For example, the French word `entente` has periods 3, 6, and 7.

Empirically, one quickly discovers that a randomly chosen word typically has a least period that is very close to its length. This readily follows from the fact that the number of words over a given alphabet grows exponentially as the length increases. It can also be seen as a particular case of the fact that most strings are not compressible.

In this paper, we quantify this basic observation and show that the expected least period of a string of length n over an ℓ-letter alphabet is $n - \alpha_\ell(n)$, where $\alpha_\ell(n)$ is $\mathcal{O}(1)$.

Another concept frequently studied in formal language theory is that of *border* of a word [13, 3, 14]. A word x has border w if w is both a prefix and a suffix of x. Normally we do not consider the trivial borders of length 0 or $n = |w|$. Thus, for example, the English word

* This work was partially supported by Czech Science Foundation grant number 13-01832S.

© Štěpán Holub and Jeffrey Shallit;
licensed under Creative Commons License CC-BY

`ionization` has one border: `ion`. Less trivially, the word `alfalfa` has two borders: `a` and `alfa`. A word with no borders is *unbordered*.

There is an obvious connection between periods of a word and its borders: if w has a period p, then it has a border of length $|w| - p$. For example, the English word `abracadabra`, of length 11, has periods 7, 10, and 11, while it has borders of length 1 and 4.

Consequently, the least period of a word corresponds to the length of the longest border (and an unbordered word corresponds to least period n, the length of the word). The reader should be constantly aware of this duality, since it is often useful and more natural to think about periods in terms of borders. This can be seen from the announced result: it is more compact to speak directly about the expected maximum border length, which is $\alpha_\ell(n)$.

If P is a set of integers, we shall write $n - P$ for $\{n - p \mid p \in P\}$, and $P - n$ for $\{p - n \mid p \in P\}$.

By $\mathrm{pref}_i(v)$, we mean the prefix of length i of the word v.

2 Multiperiodic Words and the Average Border Length

We shall obtain our results by counting words with a given length n and a given finite set of periods $P \subseteq \{1, 2, \dots n\}$, or equivalently, with a given set of border lengths $n - P$. For technical reasons, in order to be able to deal with unbordered words, we shall always suppose that $n \in P$, that is, we shall say that every word has a border length zero.

There are two basic types of requirements. Let

$$\mathcal{G}_\ell(P, n) = \{w \in \Sigma_\ell^n \mid \text{for each } p \text{ in } P,\ p \text{ is a period of } w\},$$

and let $G_\ell(P, n)$ be the cardinality of $\mathcal{G}_\ell(P, n)$. Similarly, let

$$\mathcal{F}_\ell(P, n) = \{w \in \mathcal{G}_\ell(P, n) \mid \min P \text{ is the least period of } w\},$$

and let $F(P, n)$ be the cardinality of $\mathcal{F}_\ell(P, n)$.

Words with many periods have been amply studied. In particular, there is a fast algorithm constructing a word of length n with periods P and maximal possible number of letters. Such a word, called an FW-word in the literature, is unique up to renaming of the letters. Let $\mathfrak{c}(P, n)$ denote the cardinality of the alphabet of the FW-word of length n and periods P.

▶ **Example 1.** Let $P = \{p, q\}$ and $d = \gcd(p, q)$. The well-known periodicity lemma (often called the Fine and Wilf theorem, which is the origin of the term FW-word) states that if a word of length at least $p + q - d$ has periods p and q, then it also has period d. Moreover, the bound $p + q - d$ is sharp; for all $p, q \geq 1$ there are words of length $p + q - d - 1$ with period p and q but not period d. This can be stated, using the just-introduced terminology, by the two assertions $\mathfrak{c}(\{p, q\}, p + q - d) = d$ and $\mathfrak{c}(\{p, q\}, p + q - d - 1) > d$.

The number $\mathfrak{c}(P, n)$ can be computed and the corresponding FW-word constructed using the algorithm of Tijdeman and Zamboni [16] (see [17] for an alternative presentation). The computation is summarized by the following formula:

$$\mathfrak{c}(P, n) = \begin{cases} 1, & \text{if } m = 1; \\ n, & \text{if } m \geq n; \\ \mathfrak{c}(Q, n - m), & \text{if } 2m \leq n; \\ \mathfrak{c}(Q, n - m) + 2m - n, & \text{if } m < n < 2m; \end{cases}$$

where $m = \min P$ and $Q = (P - m) \setminus \{0\} \cup \{m\}$.

Since each word having the periods in P (and possibly others) results from a coding (a letter-to-letter mapping) of the corresponding FW-word, we obtain

$$G_\ell(P,n) = \ell^{\mathfrak{c}(P,n)},$$

which is the starting point of our computation.

Note that $\mathcal{F}_\ell(\{p\},n)$ is the set of words from Σ_ℓ^n having least period p. Equivalently, $\mathcal{F}_\ell(\{n-r\},n)$ is the set of words with the longest border of length r. For $0 \le r < n$, let

$$\lambda_\ell(r,n) = \frac{F(\{n-r\},n)}{\ell^n}$$

denote the relative number of such words. Our goal is to compute

$$\alpha_\ell(n) = \sum_{r=0}^{n-1} r \cdot \lambda_\ell(r,n),$$

which is the expected maximum border length for words in Σ_ℓ^n. We first show that this quantity converges as n approaches infinity. This fact was recently independently proved in [1, Appendix].

▶ **Lemma 2.** *For each $\ell \ge 2$ and each $0 \le r < n$,*

$$|\lambda_\ell(r,n+1) - \lambda_\ell(r,n)| \le \frac{1}{\ell^{\lfloor n/2 \rfloor}}.$$

Proof. Case 1: $r \ge \lfloor n/2 \rfloor$. Then

$$|\lambda_\ell(r,n+1) - \lambda_\ell(r,n)| = \left| \frac{F_\ell(\{n+1-r\},n+1)}{\ell^{n+1}} - \frac{F_\ell(\{n-r\},n)}{\ell^n} \right|.$$

Recall that $F_\ell(\{n+1-r\},n+1)$ (resp., $F_\ell(\{n-r\},n)$) counts the words with longest border length r from Σ_ℓ^{n+1} (resp., Σ_ℓ^n). First, note that $F_\ell(\{p\},n) \le \ell^p$ for any p and n. This implies

$$|\lambda_\ell(r,n+1) - \lambda_\ell(r,n)| \le \frac{1}{\ell^r}$$

and we are done.

Case 2: $r < \lfloor n/2 \rfloor$. There is a useful correspondence between Σ_ℓ^n and Σ_ℓ^{n+1}, given by the insertion of a letter in the middle of the shorter word. The basic observation, already used in [9, 11], is that this insertion does not influence borders of length at most $\lfloor n/2 \rfloor$. Define

$$\mathcal{F} = \mathcal{F}_\ell(\{n+1-r\},n+1),$$
$$\mathcal{B} = \{w_1 a w_2 \mid a \in \Sigma_\ell,\ |w_1| = \lfloor n/2 \rfloor,\ |w_2| = \lceil n/2 \rceil,\ w_1 w_2 \in \mathcal{F}_\ell(\{n-r\},n)\}.$$

Then $|\mathcal{B}| = \ell \cdot F_\ell(\{n-r\},n)$. Let $w \in \mathcal{F} \setminus \mathcal{B}$ and write $w = w_1 a w_2$ with $a \in \Sigma_\ell$, $|w_1| = \lfloor n/2 \rfloor$, and $|w_2| = \lceil n/2 \rceil$. The words w and $w_1 w_2$ have the same borders up to length $\lfloor n/2 \rfloor$. Since $w_1 w_2 \notin \mathcal{F}_\ell(\{n-r\},n)$, we deduce that $w_1 w_2$ has a border of length at least $\lfloor n/2 \rfloor + 1$, that is, a period at most $\lceil n/2 \rceil - 1$. This implies

$$|\mathcal{F} \setminus \mathcal{B}| \le \ell \cdot \sum_{j=0}^{\lceil n/2 \rceil - 1} \ell^j < \ell^{\lceil n/2 \rceil + 1}. \tag{1}$$

Similarly, a word $w \in \mathcal{B} \setminus \mathcal{F}$ has period at most $\lceil n/2 \rceil$, and so

$$|\mathcal{B} \setminus \mathcal{F}| \leq \sum_{j=0}^{\lceil n/2 \rceil} \ell^j < \ell^{\lceil n/2 \rceil + 1}. \tag{2}$$

We thus obtain

$$|\lambda_\ell(r, n+1) - \lambda_\ell(r, n)| = \frac{1}{\ell^{n+1}} \Big| |\mathcal{B} \setminus \mathcal{F}| - |\mathcal{F} \setminus \mathcal{B}| \Big| < \frac{1}{\ell^{\lfloor n/2 \rfloor}}. \qquad \blacktriangleleft$$

▶ **Theorem 3.** *For each $\ell \geq 2$ and $r \geq 0$, the limits*

$$\alpha_\ell := \lim_{n\to\infty} \alpha_\ell(n) \quad \text{and} \quad \lambda_\ell(r) = \lim_{n\to\infty} \lambda_\ell(r, n)$$

exist. Furthermore, the convergence is exponential.

Proof. Follows directly from the definition of $\alpha_\ell(n)$ and Lemma 2. ◀

3 Recurrences

From the estimates of the previous section, we know that $\alpha_\ell(n)$ and $\lambda_\ell(r, n)$ both converge quickly to α_ℓ and $\lambda_\ell(r)$, respectively. Thus, they can be estimated to a few digits by explicit enumeration (see [1, Appendix] for α_2 and α_3).

In order to evaluate $\alpha_\ell(n)$ to dozens of decimal places, however, we need a more efficient way to calculate $F_\ell(\{p\}, n)$. This can be done using the recurrence formulas that we derive below. They are reformulations and generalizations of formulas given by Harborth [9] for sets of periods.

We first prove the following auxiliary claim.

▶ **Lemma 4.** *Let a word w have a period $p < |w|$ and let u be the prefix of w of length $|w| - p$. Then w has a period $q > p$ if and only if u has a period $q - p$.*

Proof. Note that u is a border of w. The following conditions are easily seen to be equivalent:

- w has a period q,
- w has a border of length $|w| - q$,
- u has a border of length $|w| - q$,
- u has a period $|u| - (|w| - q)$.

Since $|u| - (|w| - q) = (|w| - p) - (|w| - q) = q - p$, the proof is completed. ◀

▶ **Theorem 5.** *Let P be a set of periods with $m = \min P$ and $\max P < n$. Then*

$$F_\ell(P, n) = G_\ell(P, n) - \sum_{p=\lceil m/2 \rceil}^{m-1} H_\ell(P, p, n)\,, \tag{3}$$

where

$$H_\ell(P, p, n) := \begin{cases} F_\ell\big((P - p) \cup \{p\}, n - p\big), & \text{if } p < \lceil n/2 \rceil; \\ \ell^{2p-n} \cdot F_\ell(P - p, n - p), & \text{if } p \geq \lceil n/2 \rceil\,. \end{cases} \tag{4}$$

Proof. From $G_\ell(P, n)$ we have to subtract the number of words from Σ_ℓ^n that have periods P but also a period smaller than m. We define, for each $1 \leq p < m$, the set

$$\mathcal{H}_\ell(P, p, n) = \{w \in \Sigma_\ell^n \mid w \text{ has periods } P \cup \{p\}, \text{ and no period } p' \text{ with } p < p' < m\}\,.$$

If $p < \lceil m/2 \rceil$ then $\mathcal{H}_\ell(P,p,n)$ is empty, since a word $w \in \mathcal{H}_\ell(P,p,n)$ also has a period $2p$, and $p < 2p < m$ contradicts the definition of $\mathcal{H}_\ell(P,p,n)$. Moreover, the sets $\mathcal{H}_\ell(P,p,n)$ are pairwise disjoint, and

$$\mathcal{G}_\ell(P,n) \setminus \mathcal{F}_\ell(P,n) = \bigcup_{p=\lceil m/2 \rceil}^{m-1} \mathcal{H}_\ell(P,p,n)\,.$$

It remains to show that $H_\ell(P,p,n)$ is the cardinality of $\mathcal{H}_\ell(P,p,n)$ for each $\lceil m/2 \rceil \le p < m-1$.

Let $p < \lceil n/2 \rceil$. We claim that $w \mapsto \mathrm{pref}_{n-p} w$ is a one-to-one mapping of $\mathcal{H}_\ell(P,p,n)$ to $\mathcal{F}_\ell\big((P-p)\cup\{p\}, n-p\big)$. Let $w \in \mathcal{H}_\ell(P,p,n)$. By Lemma 4, the word $\mathrm{pref}_{n-p} w$ has periods $P-p$ and no period $p'-p$ with $p < p' < m$, that is, no period less than $m-p$. Since $m-p = \min\big((P-p)\cup\{p\}\big)$ and since $\mathrm{pref}_{n-p} w$ also has a period p, we have $\mathrm{pref}_{n-p} w \in \mathcal{F}_\ell\big((P-p)\cup\{p\}, n-p\big)$. Similarly, one can verify that if $v \in \mathcal{F}_\ell\big((P-p)\cup\{p\}, n-p\big)$, then $w_v := (\mathrm{pref}_p v)^{n/p} \in \mathcal{H}_\ell(P,p,n)$ and $\mathrm{pref}_{n-p} w_v = v$.

Let $p \ge \lceil n/2 \rceil$. Again, using Lemma 4, it is straightforward to verify that

$$\mathcal{H}_\ell(P,p,n) = \{ vuv \mid v \in \mathcal{F}_\ell(P-p, n-p), u \in \Sigma_\ell^{2p-n} \}\,. \qquad \blacktriangleleft$$

If $\min P$ is small, then we can formulate a more explicit formula that uses the Möbius μ-function.

▶ **Lemma 6.** *Let P be a set of periods with $m = \min P \le \lfloor n/2 \rfloor + 1$. Then*

$$F_\ell(P,n) = \sum_{d \mid m} \mu\left(\frac{m}{d}\right) G_\ell\left(P \cup \{d\}, n\right). \tag{5}$$

Proof. Let w be a word of length n with a period m and let p be the least period of w. Then, by the periodicity lemma, we have that p divides m, since $p < m$ implies $p + m - 1 \le n$. Therefore, for each divisor p of m,

$$G_\ell\left(P \cup \{p\}, n\right) = \sum_{d \mid p} F_\ell(P \cup \{d\}, n),$$

and the claim follows from Möbius inversion. $\blacktriangleleft$

4 Explicit Formulas

In this section we derive explicit formulas for $\lambda_\ell(0)$ and $\lambda_\ell(1)$, which are the asymptotic probabilities that a random word is unbordered, or has longest border of length one, respectively. These are two cases in which Theorem 5 yields a relatively simple expression, since $\lceil m/2 \rceil \ge \lfloor n/2 \rfloor$.

4.1 Unbordered Words

The number of unbordered words satisfies a well known recurrence formula (see, e.g., [9, p. 143, Eq. (34)] for the binary case and [11] for the general case). The formula can be verified using Theorem 5 but we shall give an elementary proof. In this section, let u_n denote $F_\ell(\{n\}, n)$, and let $t(n)$ denote $\lambda_\ell(0, n)$.

▶ **Theorem 7.**

$$u_n = \begin{cases} \ell, & \text{if } n = 1; \\ \ell(\ell-1) & \text{if } n = 2; \\ \ell \cdot u_{n-1}, & \text{if } n \geq 3 \text{ is odd}; \\ \ell \cdot u_{n-1} - u_{n/2}, & \text{if } n \geq 4 \text{ is even}. \end{cases}$$

Proof. For $k = 1, 2$, the verification is straightforward. Let x and y be nonempty words with $|x| = |y|$ and consider words xy, xay and $xaby$ where a and b are letters.

Since the shortest border of xay has length at most $|x|$, the word xy is unbordered if and only if xay is. This proves $u_n = \ell \cdot u_{n-1}$ if n is odd.

On the other hand, $xaby$ can have the shortest border of length $|x| + 1$. Therefore, $xaby$ is unbordered if and only if (i) xy is unbordered and (ii) $xa \neq by$. Since the shortest border is itself unbordered, we obtain $u_n = \ell^2 \cdot u_{n-2} - u_{n/2} = \ell \cdot u_{n-1} - u_{n/2}$ if n is even. ◀

Theorem 7 directly yields, for each $n \geq 1$,

$$t(2n+1) = t(2n) = t(2n-1) - t(n)\ell^{-n}\,.$$

Therefore

$$t(2n) = t(1) + \sum_{i=2}^{2n}(t(i) - t(i-1)) = 1 - \sum_{j=1}^{n} t(j)\ell^{-j}\,.$$

Defining the generating function $L_0(x) = \sum_{n\geq 1} t(n)x^n$, we get

$$\lim_{n\to\infty} \lambda_\ell(0, n) = 1 - L_0\left(\frac{1}{\ell}\right). \tag{6}$$

The next step is to obtain a functional equation for $L_0(x)$:

$$\begin{aligned} L_0(x)(1-x) &= t(1)x + \sum_{k\geq 2}(t(k) - t(k-1))x^k = \\ &= t(1)x + \sum_{j\geq 1}(t(2j) - t(2j-1))x^{2j} = \\ &= t(1)x - \sum_{j\geq 1} t(j)\ell^{-j}x^{2j} = x - L_0(x^2/\ell)\,. \end{aligned}$$

Therefore

$$L_0(x) = \frac{x}{1-x} - \frac{L_0(x^2/\ell)}{1-x}\,.$$

Successively substituting $x = 1/\ell$, $x = 1/\ell^3$, $x = 1/\ell^7$, …, we get

$$\begin{aligned} L_0\left(\frac{1}{\ell}\right) &= \frac{1}{\ell-1} - \left(1 + \frac{1}{\ell-1}\right) L_0\left(\frac{1}{\ell^3}\right), \\ L_0\left(\frac{1}{\ell^3}\right) &= \frac{1}{\ell^3-1} - \left(1 + \frac{1}{\ell^3-1}\right) L_0\left(\frac{1}{\ell^7}\right), \\ &\vdots \\ L_0\left(\frac{1}{\ell^{2^i-1}}\right) &= \frac{1}{\ell^{2^i-1}-1} - \left(1 + \frac{1}{\ell^{2^i-1}-1}\right) L_0\left(\frac{1}{\ell^{2^{i+1}-1}}\right). \end{aligned}$$

Since it is easy to see that

$$\lim_{n\to\infty} \frac{1}{\ell^{2^n-1}-1} \prod_{i=1}^{n-1} \left(1+\frac{1}{\ell^{2^i-1}-1}\right) L_0\left(\frac{1}{\ell^{2^n-1}}\right) = 0\,,$$

we obtain

$$L_0\left(\frac{1}{\ell}\right) = \sum_{n\geq 1} \left(\frac{(-1)^{n+1}}{\ell^{2^n-1}-1} \prod_{i=1}^{n-1} \left(1+\frac{1}{\ell^{2^i-1}-1}\right)\right).$$

A similar analysis was given previously by [2], although our analysis is slightly cleaner.

4.2 Words With Longest Border of Length One

There is also a relatively simple recurrence for $F_\ell(\{n-1\}, n)$, that is, for words with the longest border of length 1. The particular case $\ell = 2$ was previously given by Harborth [9, p. 143, Eq. (36)]. In this section, we let v_n denote $F_\ell(\{n-1\}, n)$, and let $s(n)$ denote $\lambda_\ell(1, n)$.

▶ **Theorem 8.**

$$v_n = \begin{cases} 0, & \text{if } n = 1; \\ \ell & \text{if } n = 2; \\ \ell \cdot v_{n-1} - v_{(n+1)/2}, & \text{if } n \geq 3 \text{ is odd}; \\ \ell \cdot v_{n-1} - (\ell-1)v_{n/2}, & \text{if } n \geq 4 \text{ is even}. \end{cases}$$

Proof. Verify that $v_1 = 0$ and $v_2 = \ell$, and let x and y be nonempty words with $|x| = |y|$. Consider words $cxyc$, $cxayc$ and $cxabyc$ where a, b, c are (not necessarily distinct) letters.

The letter c is the longest border of the word $cxayc$ if and only if (i) c is the longest border of $cxyc$ and (ii) $cxa \neq ayc$. Moreover, (i') c is the shortest border of $cxyc$, and (ii') $cxa = ayc$ $(= cxc)$ if and only if c is the shortest border of cxc. This implies $v_n = \ell \cdot v_{n-1} - v_{(n+1)/2}$ for $n \geq 3$ odd.

Similarly, c is the shortest border of $cxabyc$ if and only if (i) c is the longest border of $cxyc$ and (ii) $cxa \neq byc$. As above, we have to subtract the number of words cxc with the longest border c. It follows that $v_n = \ell^2 \cdot v_{n-2} - v_{n/2} = \ell v_{n-1} + (\ell-1)v_{n/2}$ for $n \geq 4$ even. ◀

From Theorem 8, we deduce

$$s(2n) - s(2n-2) = -s(n)\ell^{-n}, \qquad n \geq 2, \tag{7}$$

$$s(2n) - s(2n-1) = (\ell-1)s(n)\ell^{-n}, \qquad n \geq 2, \tag{8}$$

$$s(2n+1) - s(2n) = -s(n+1)\ell^{-n}, \qquad n \geq 1\,. \tag{9}$$

Using (7), we obtain

$$s(2n) = s(2) + \sum_{r=j}^{n} (s(2j) - s(2j-2)) = 1/\ell - \sum_{j=1}^{n} s(j)\ell^{-j}\,.$$

Defining the generating function $L_1(x) = \sum_{k\geq 1} s(k)x^k$, we then get

$$\lambda_\ell(1) = \frac{1}{\ell} - L_1\left(\frac{1}{\ell}\right).$$

A functional equation for L_1 is obtained as follows:

$$\begin{aligned} L_1(x)(1-x) &= s(1)x + \sum_{k\geq 2}(s(k)-s(k-1))x^k = \\ &= \frac{1}{\ell}x^2 + \sum_{i\geq 1}(s(2i+1)-s(2i))x^{2i+1} + \sum_{i\geq 2}(s(2i)-s(2i-1))x^{2i} = \\ &= \frac{1}{\ell}x^2 - \sum_{i\geq 1} s(i+1)\ell^{-i}x^{2i+1} + \sum_{i\geq 2}(\ell-1)s(i)\ell^{-i}x^{2i} = \\ &= \frac{1}{\ell}x^2 - \frac{\ell}{x}L_1\left(\frac{x^2}{\ell}\right) + (\ell-1)L_1\left(\frac{x^2}{\ell}\right). \end{aligned}$$

We have

$$L_1(x) = \frac{x^2}{\ell(1-x)} + \frac{\ell-1-\ell/x}{1-x}L_1\left(\frac{x^2}{\ell}\right),$$

and

$$L_1\left(\frac{1}{\ell^{2^i-1}}\right) = \frac{1}{\ell^{2^i}(\ell^{2^i-1}-1)} - \ell^{2^i-1}\frac{\ell^{2^i}-\ell+1}{\ell^{2^i-1}-1}L_1\left(\frac{1}{\ell^{2^{i+1}-1}}\right).$$

From here, we deduce

$$L_1\left(\frac{1}{\ell}\right) = \sum_{n\geq 1}\frac{(-1)^{n+1}}{\ell^{n+1}}\prod_{i=1}^{j}\frac{\ell^{2^{i-1}}-\ell+1}{\ell^{2^i-1}-1}.$$

We do not know how to obtain similar expressions for other border lengths.

5 Particular Values

Theorem 5, as well as explicit formulas from the previous section allow fast computer evaluation of $\alpha_\ell(n)$ and $\lambda_\ell(r,n)$ for large n, and therefore also evaluation of $\lambda_\ell(r)$ and α_ℓ with high precision. We list some rounded values in the following tables.

ℓ	α_ℓ
2	1.64116491178296695613
3	0.68587617299708343978
4	0.42195659003603599699
5	0.30201601806282253073
y 10	0.12233344445364555354
50	0.02081648979722449000

r	$\lambda_2(r)$
0	0.26778684021788911238
1	0.30042007151830329926
2	0.19891874779036456415
3	0.11216079483159432642
5	0.03044609816129782975
10	0.00097577734413168807

And some values of $\lambda_\ell(r)$ rounded to four decimal digits:

$\lambda_\ell(r)$	$\ell=3$	$\ell=4$	$\ell=5$	$\ell=10$
$r=0$	0.55698	0.68775	0.76006	0.89000
$r=1$	0.28270	0.23024	0.19034	0.09890
$r=2$	0.10547	0.06126	0.03961	0.00999
$r=3$	0.03641	0.01555	0.00798	0.00100

For example, we see that a long binary word chosen randomly has about 27% chance to be unbordered. A bit more probable, at 30%, is that such a word will have its longest border of

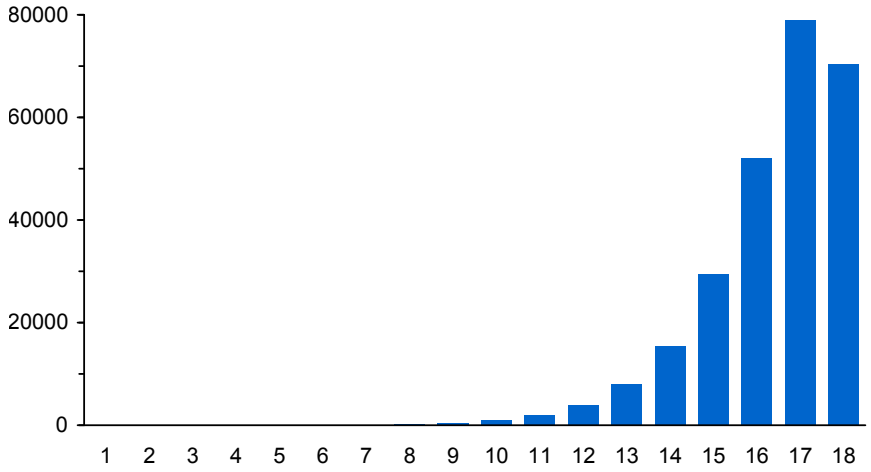


Figure 1 Distribution of lengths of shortest period for binary words of length 18.

length one. Over a five-letter alphabet, more than three words out of four are unbordered, on average.

Figure 1 shows the distribution of lengths of the shortest period for binary words of length $n = 18$.

Our original motivation was a question about the average period of a binary word. The answer is, that the border of a binary word has asymptotically constant expected length, namely

$$\alpha_2 \doteq 1.641164911782966956127744169400825540659536878257715 43\ldots .$$

6 Final Remarks

Recently there has been some interest in computing the expected value of the largest unbordered factor of a word [10]. This is a related, but seemingly much harder, problem.

References

1 A. Alatabbi, A. S. S. Islam, M. S. Rahman, J. Simpson, and W. F. Smyth. Enhanced covers of regular & indeterminate strings using prefix tables. *CoRR*, abs/1506.06793, 2015. URL: `http://arxiv.org/abs/1506.06793`.

2 G. Blom. Overlapping binary sequences (problem 94-20). *SIAM Review*, 37:619–620, 1995.

3 A. Ehrenfeucht and D. M. Silberger. Periodicity and unbordered segments of words. *Discrete Math.*, 26:101–109, 1979.

4 N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proc. Amer. Math. Soc.*, 16:109–114, 1965.

5 L. Guibas. Periodicities in strings. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume F12 of *NATO ASI*, pages 257–269. Springer-Verlag, 1985.

6 L. J. Guibas and A. M. Odlyzko. Periods in strings. *J. Combin. Theory. Ser. A*, 30:19–42, 1981.

7 L. J. Guibas and A. M. Odlyzko. String overlaps, pattern matching, and nontransitive games. *J. Combin. Theory. Ser. A*, 30:183–208, 1981.

8 V. Halava, T. Harju, and L. Ilie. Periods and binary words. *J. Combin. Theory. Ser. A*, 89:298–303, 2000.

9 H. Harborth. Endliche 0-1–Folgen mit gleichen Teilblöcken. *Journal für die reine und angewandte Mathematik*, 271:139–154, 1974.

10 A. Loptev, G. Kucherov, and T. Starikovskaya. On maximal unbordered factors. In F. Cicalese et al., editor, *CPM 2015*, volume 9133 of *Lecture Notes in Computer Science*, pages 343–354. Springer-Verlag, 2015.

11 P. T. Nielsen. A note on bifix-free sequences. *IEEE Trans. Inform. Theory*, IT-19:704–706, 1973.

12 E. Rivals and S. Rahmann. Combinatorics of periods in strings. *J. Combin. Theory. Ser. A*, 104:95–113, 2003.

13 D. M. Silberger. Borders and roots of a word. *Progress in Mathematics*, 30:191–199, 1971.

14 D. M. Silberger. How many unbordered words? *Comment. Math. Prace Mat.*, 22:143–145, 1980.

15 R. Tijdeman and L. Zamboni. Fine and Wilf words for any periods. *Indag. Math.*, 14:135–147, 2003.

16 R. Tijdeman and L. Q. Zamboni. Fine and Wilf words for any periods II. *Theor. Comput. Sci.*, 410(30-32):3027–3034, 2009.

17 Š. Holub. On an algorithm for multiperiodic words. *Acta Polytechnica*, 53(4):344–346, 2013.

Constrained Bipartite Vertex Cover: The Easy Kernel is Essentially Tight*

Bart M. P. Jansen

Eindhoven University of Technology, P. O. Box 513, Eindhoven, The Netherlands
b.m.p.jansen@tue.nl

Abstract

The Constrained Bipartite Vertex Cover problem asks, for a bipartite graph G with partite sets A and B, and integers k_A and k_B, whether there is a vertex cover for G containing at most k_A vertices from A and k_B vertices from B. The problem has an easy kernel with $2k_A \cdot k_B$ edges and $4k_A \cdot k_B$ vertices, based on the fact that every vertex in A of degree more than k_B has to be included in the solution, together with every vertex in B of degree more than k_A. We show that the number of vertices and edges in this kernel are asymptotically essentially optimal in terms of the product $k_A \cdot k_B$. We prove that if there is a polynomial-time algorithm that reduces any instance (G, A, B, k_A, k_B) of Constrained Bipartite Vertex Cover to an equivalent instance (G', A', B', k'_A, k'_B) such that $k'_A \in (k_A)^{\mathcal{O}(1)}$, $k'_B \in (k_B)^{\mathcal{O}(1)}$, and $|V(G')| \in \mathcal{O}((k_A \cdot k_B)^{1-\varepsilon})$, for some $\varepsilon > 0$, then NP $\subseteq$ coNP/poly and the polynomial-time hierarchy collapses. Using a different construction, we prove that if there is a polynomial-time algorithm that reduces any n-vertex instance into an equivalent instance (of a possibly different problem) that can be encoded in $\mathcal{O}(n^{2-\varepsilon})$ bits, then NP $\subseteq$ coNP/poly.

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

Keywords and phrases kernel lower bounds, constrained bipartite vertex cover

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.45

1 Introduction

Motivation. Vertex Cover is a classic problem in combinatorial optimization. It has served as a testbed for a myriad of different techniques in the field of parameterized algorithmics. In this paper we study a variant of this problem on bipartite graphs:

> Constrained Bipartite Vertex Cover
> **Input:** A bipartite graph G with partite sets A and B, and integers k_A and k_B.
> **Question:** Is there a vertex cover S for G such that $|S \cap A| \leq k_A$ and $|S \cap B| \leq k_B$?

While Vertex Cover is in P for bipartite graphs, this constrained variant is NP-complete [17]. It is motivated by work in reconfigurable VLSI, since it can be used to model the Sparse Allocation Problem. We refer to the recent paper by Bai and Fernau [1] for a detailed overview of the history of the problem and its applications. This paper deals with the limits of efficient preprocessing procedures for Constrained Bipartite Vertex Cover.

Let us call a vertex cover S of a bipartite graph (k_A, k_B)-*constrained* if it satisfies $|S \cap A| \leq k_A$ and $|S \cap B| \leq k_B$. Observe that if an instance contains a vertex $v \in A$ of degree more than k_B, then any (k_A, k_B)-constrained vertex cover includes a. If a is not used, then all

* This work was supported by NWO Veni grant "Frontiers in Parameterized Preprocessing" and NWO Gravitation grant "Networks".

© Bart M. P. Jansen;
licensed under Creative Commons License CC-BY

neighbors of a are in the cover. However, all such neighbors belong to B and there are more than k_B of them. This suggests a simple reduction rule for the problem: if there is a vertex $a \in A$ of degree more than k_B, then remove vertex a and decrease k_A by one. Symmetrically, if there is a vertex $b \in B$ of degree more than k_A, remove it and decrease k_B. If S is a (k_A, k_B)-constrained vertex cover of an exhaustively reduced graph, the k_A vertices in A cover at most $\max_{a \in A} \deg(a) \leq k_B$ edges each, and the k_B vertices in B each cover at most k_A edges each. To be able to cover all the edges, the number of edges must therefore be bounded by $2(k_A \cdot k_B)$ and the instance can be rejected if this is not the case. After removing isolated vertices (which do not affect the answer) from the graph, the number of vertices can be bounded by $4(k_A \cdot k_B)$, since each edge contributes at most two vertices.

This simple kernelization strategy for Constrained Bipartite Vertex Cover has been known for over thirty years [10]. It is notably less effective than the well-known kernelization schemes for the classic Vertex Cover problem, which can efficiently reduce any instance (G, k) to an equivalent one with at most $2k$ vertices [5] (cf. [11, Section 4]). It is therefore natural to ask whether a similar size bound can be attained for Constrained Bipartite Vertex Cover. This was posed as an open problem by Marcin Pilipczuk [18].

Our Results. We show that, under the assumption that $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$, neither the number of vertices nor the number of edges in the simple kernel for Constrained Bipartite Vertex Cover can be significantly improved below $\Theta(k_A \cdot k_B)$. The simple preprocessing procedure outlined above is therefore close to optimal in terms of the product $k_A \cdot k_B$.

Concretely, our first result shows that if there is an $\varepsilon > 0$ and a polynomial-time algorithm that reduces any instance (G, A, B, k_A, k_B) of Constrained Bipartite Vertex Cover to an equivalent instance (G', A', B', k'_A, k'_B) such that $k'_A \in (k_A)^{\mathcal{O}(1)}$, $k'_B \in (k_B)^{\mathcal{O}(1)}$, and $|V(G')| \in \mathcal{O}((k_A \cdot k_B)^{1-\varepsilon})$, then $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ and the polynomial-time hierarchy collapses to its third level [19]. This result is obtained using the complementary witness lemma of Dell and van Melkebeek [8]. The lemma shows that $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ follows if the following type of polynomial-time compression algorithm exists for some $c \in \mathbb{N}$: The input is a sequence $x_1, \ldots, x_{n^c}$ of size-n inputs to an NP-hard problem. The output is a single instance x^* of Constrained Bipartite Vertex Cover whose truth status is the logical or of the answers to the inputs, with $|x^*| \in \mathcal{O}(n^c \log n^c)$. We present a construction (Lemma 2) that allows us to obtain precisely such a compression algorithm from a kernelization procedure for Constrained Bipartite Vertex Cover that satisfies the constraints set out above. The key in this construction is to embed the n^c inputs x_i into a Constrained Bipartite Vertex Cover instance on a graph that is *lopsided*: one partite set has size $\mathcal{O}(n^2 c \log n)$, while the other set has size $\mathcal{O}(n^{c+1})$. The produced graph is fairly sparse as the number of edges is at most the number of vertices times the size of the smaller partite set, $\mathcal{O}(n^2 c \log n)$, whose dependence on the *number* n^c of embedded instances is minimal. For sufficiently large c, the number of edges in the graph is therefore roughly equal to the number of vertices. If a hypothetical kernel can reduce the order of the composed instance of Constrained Bipartite Vertex Cover substantially, then the relation between the vertex and edge count yields the following: after an application of the simple kernelization, the number of edges reduces substantially as well (since the parameter values k_A and k_B do not increase too much). A kernel with $\mathcal{O}((k_A \cdot k_B)^{1-\varepsilon})$ vertices would therefore give a compression algorithm satisfying the requirements of the complementary witness lemma and imply $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

Our second result deals with sparsification, i.e., reducing the number of edges in the graph to make it less dense. We prove that unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$, there is no polynomial-time algorithm that reduces any n-vertex instance of Constrained Bipartite Vertex Cover

to an equivalent instance, of a possibly different problem, that can be encoded in $\mathcal{O}(n^{2-\varepsilon})$ bits for $\varepsilon > 0$. For this result we can use the framework of cross-composition to avoid invoking the complementary witness lemma directly. The key is to find a construction that embeds a series of t inputs of an NP-hard problem, each of size at most n, into an instance of Constrained Bipartite Vertex Cover where both partite sets have roughly $n \cdot \sqrt{t}$ vertices. In sharp contrast to Lemma 2, this second construction produces a very *dense* graph whose partite sets are balanced in size.

After discarding isolated vertices (which play no role in this problem), the number of vertices in an instance is at most twice the number of edges. The lower bound on the number of vertices in the kernel given by the first result therefore implies a similar lower bound on the number of edges. This lower bound holds against kernelizations that incur a bounded increase in the budget values k_A and k_B. The sparsification lower bound yields a more general edge lower bound (Corollary 14): even the existence of a kernelization for Constrained Bipartite Vertex Cover with $\mathcal{O}((k_A \cdot k_B)^{1-\varepsilon})$ edges that increases the budget values k_A and k_B arbitrarily, implies NP $\subseteq$ coNP/poly.

Related Work. There is another problem in the literature, called Constrained Minimum Bipartite Vertex Cover, which is similar to ours but behaves differently. Its inputs also consist of a graph G with partite sets A and B, along with integers k_A and k_B. However, the question is now whether there is a (k_A, k_B)-constrained vertex cover that is also a *minimum* vertex cover in G, i.e., for which there is no (unconstrained) vertex cover of G that is strictly smaller. This minimality requirement can make it possible to infer that a vertex must belong to any valid solution, while this conclusion is not valid if a vertex cover is allowed whose size is not globally minimum. Chen and Kanj [4, Section 2] showed that the Dulmage-Mendelsohn decomposition of bipartite graphs can be exploited to reduce an instance of Constrained Minimum Bipartite Vertex Cover to an equivalent instance with at most $2(k_A + k_B)$ vertices. Their result does not carry over to the more general problem considered here.

Let us return to Constrained Bipartite Vertex Cover. It has received considerable attention and was studied using a number of different algorithmic paradigms. Fernau and Niedermeier [12] first used the framework of parameterized complexity to attack the Constrained Bipartite Vertex Cover problem. They developed a moderately exponential FPT branching algorithm, aided by the simple problem kernel. Bai and Fernau [1] simplified their algorithm a decade later and report on experimental results.

There are a handful of results concerning tight lower bounds for kernelizations. There is work by Dell and van Melkebeek [8] on Vertex Cover, by Dell and Marx [7] and Hermelin and Wu [14] on packing problems, by Kratsch et al. [16] on Point-Line Cover, and by Jansen [15] on Treewidth.

Organization. Section 2 contains preliminaries on parameterized complexity. In Section 3 we develop the lower bound on the number of vertices in kernels for Constrained Bipartite Vertex Cover. Section 4 uses a different construction to give a lower bound on the number of edges. We conclude in Section 5.

2 Preliminaries

A parameterized problem $\mathcal{Q}$ is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. The second component of a tuple $(x, k) \in \Sigma^* \times \mathbb{N}$ is called the *parameter* [6, 9]. A parameterized problem $\mathcal{Q}$ is (strongly uniformly) *fixed-parameter tractable* if there is an algorithm that

decides whether $(x, k) \in \mathcal{Q}$ that runs in time $f(k)|x|^{\mathcal{O}(1)}$ for some computable function f. The set $\{1, 2, \ldots, n\}$ is abbreviated as $[n]$. All logarithms are base 2. For a set S and integer k we denote by $\binom{S}{k}$ the collection of all size-k subsets of S.

We say that a set S *avoids* a set T if $S \cap T = \emptyset$. Two vertices x, y in a graph are *false twins* if they are not adjacent to each other, but have exactly the same (open) neighborhood. Two disjoint vertex sets A, B in a graph are *adjacent* if there is an edge of the form $\{a, b\}$ with $a \in A$ and $b \in B$. The sets are *fully adjacent* if all members of A are adjacent to all members of B.

▶ **Definition 1** (Generalized kernelization)**.** 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 kernelization 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:

- $|x'|$ and k' are bounded by $h(k)$.
- $(x', k') \in \mathcal{Q}'$ if and only if $(x, k) \in \mathcal{Q}$.

The algorithm is a *kernelization*, or in short a *kernel*, for $\mathcal{Q}$ if $\mathcal{Q}' = \mathcal{Q}$. It is a *polynomial (generalized) kernelization* if $h(k)$ is a polynomial.

The notion of generalized kernelization, a term first used by Bodlaender et al. [2], is closely related to the notion of *compression* (cf. [3]). In both cases an instance is reduced to a small but equivalent instance of a different problem. In generalized kernelization the output is a parameterized instance, whereas in compression the output is a classical instance.

3 Vertex Lower Bound

The goal of this section is to prove a lower bound on the number of vertices in kernels for Constrained Bipartite Vertex Cover. Lemma 2 is the key ingredient.

▶ **Lemma 2.** *Let $n \in \mathbb{N}$ be even and $t \in \mathbb{N}$ be a power of two. There is an algorithm that, given t graphs $G_1, \ldots, G_t$ with exactly n vertices each, outputs a bipartite graph G' with partite sets A' and B', along with integers k'_A and k'_B, such that:*

1. *There is an index $i \in [t]$ such that G_i contains a clique of size $n/2$ if and only if G' has a (k'_A, k'_B)-constrained vertex cover.*
2. *$k'_A \in \mathcal{O}(n^2 \log t)$ and $k'_B \in \mathcal{O}(n \cdot t)$.*

The running time is polynomial in t and n.

Proof. We describe the construction carried out by the algorithm. It will be easy to see that it can be done in polynomial time. Let the input consist of graphs $G_1, \ldots, G_t$ where t is a power of two, and let each graph have n vertices. For each graph G_i, we identify its vertex set with the integers in the range $[n]$. Construct the bipartite graph G', whose partite sets we will denote by A' and B', as follows.

1. Add a *canonical* set B'_c consisting of n vertices $b_1, \ldots, b_n$ to B'. Add a *canonical* set A'_c consisting of $\binom{n}{2}$ vertices $a_{i,j}$ for $1 \leq i < j \leq n$ to A'. Add an edge between b_ℓ and $a_{i,j}$ if $i = \ell$ or $j = \ell$. This ensures that the graph $G'[A'_c \cup B'_c]$ is the vertex-edge incidence graph of an n-vertex clique, which contains the vertex-edge incidence graphs of all n-vertex graphs as induced subgraphs. This is why the vertex sets are called canonical.
2. For $i \in [t]$ add a vertex set B'_i to B' consisting of n vertices. These vertices will be false twins in the graph, with identical open neighborhoods. The adjacency between B'_i and the canonical set A'_c encodes the structure of the input graph G_i. For each pair $1 \leq i < j \leq n$ such that $\{i, j\}$ is *not* an edge in G_i, make all vertices of B'_i adjacent to $a_{i,j}$.

3. For $i \in [\log t]$, add two vertex sets $A'_{0,i}$ and $A'_{1,i}$ to A', of $\binom{n}{2}$ vertices each. The vertices in $A'_{0,i}$ will be false twins, as will the vertices in $A'_{1,i}$. The adjacencies between the sets $A'_{0/1,j}$ and the sets B'_i are based on the binary encoding of the number i. As t is a power of two, the integers in the range $[t]$ can be uniquely identified by $(\log t)$-bit strings, treating the number t as the all-zero string. For each $j \in [\log t]$, for each $i \in [t]$, do the following. If the j-th bit of the number i is a zero, then make all vertices in B'_i adjacent to all vertices in $A'_{0,j}$. If the bit is a one, make B'_i fully adjacent to $A'_{1,j}$ instead.

To conclude the construction, set $k'_A := (\binom{n}{2} - \binom{n/2}{2}) + \binom{n}{2} \log t$ and $k'_B := \frac{n}{2} + (t-1) \cdot n$. It is easy to see that the construction can be carried out in time polynomial in n and t, and that k'_A and k'_B satisfy the claimed bounds. It remains to prove the connection between cliques in the input graphs and constrained bipartite vertex covers of G'.

▶ **Claim 3.** *If there is an index $i^* \in [t]$ such that G_{i^*} has a clique of size $n/2$, then G' has a (k'_A, k'_B)-constrained vertex cover.*

Proof. Suppose G_{i^*} contains a clique $D \subseteq [n]$ of size $n/2$. We construct a constrained vertex cover S of G' as follows.

- Add all vertices b_j to S for which $j \in D$. This contributes $n/2$ vertices to S.
- Add all vertices $a_{i,j}$ to S for which $i \notin D$ or $j \notin D$, contributing $\binom{n}{2} - \binom{n/2}{2}$ vertices to S.
- For each $i \in [t] \setminus \{i^*\}$, add all vertices in B'_i to S. This contributes $(t-1) \cdot n$ vertices to S.
- For each $j \in [\log t]$, if the j-th bit of i^* is a zero then add all vertices of $A'_{0,j}$ to S. Otherwise add all vertices of $A'_{1,j}$ to S. This contributes $\binom{n}{2} \log t$ vertices to S.

It is easy to verify that S contains k'_A vertices from A' and k'_B vertices from B'. It remains to check that S is a vertex cover of G'.

1. To see that all edges of $G'[A'_c \cup B'_c]$ are covered by S, consider an edge between b_ℓ and $a_{i,j}$, which exists only if $i = \ell$ or $j = \ell$. If $\ell \in D$ then $b_\ell \in S$ covers the edge. Otherwise, $a_{i,j}$ is contained in S by the second step, and covers the edge.
2. To see that the edges between sets B'_i and A'_c are covered, observe that this trivially holds for all $i \neq i^*$ since B'_i is contained entirely in S. For i^* note that B'_{i^*} is only adjacent to vertices $a_{i,j}$ with $1 \leq i < j \leq n$ if $\{i,j\}$ is not an edge of G_{i^*}. In this case, we know that i and j are not both contained in the clique D in graph G_{i^*}, and therefore $a_{i,j}$ was added to S to cover such edges during the construction of S above.
3. To see that the edges between sets B'_i and $A'_{0/1,j}$ are covered, observe that this trivially holds for all $i \neq i^*$ as B'_i is contained in S. For i^* note that the adjacency between B'_{i^*} and $A'_{0/1}$'s follows the binary encoding of the number i^*. As we added the sets $A'_{0/1,j}$ to S that match the bit values of i^*, all such edges are covered.

Since all edges of G' are covered by S, this proves Claim 3. ◀

The next claim establishes several properties of constrained vertex covers in G', leading to a proof that a (k'_A, k'_B)-constrained vertex cover implies the existence of a clique of size $n/2$ in one of the input graphs.

▶ **Claim 4.** *For any (k'_A, k'_B)-constrained vertex cover S of G', the following holds.*

1. *For every $j \in [\log t]$ the set S contains all vertices of $A'_{0,j}$ or all vertices of $A'_{1,j}$.*
2. *There is an index $i^* \in [t]$ such that S contains all vertices of B'_i for all $i \in [t] \setminus \{i^*\}$.*
3. *There are at least $\binom{n/2}{2}$ distinct vertices $a_{i,j} \in A'_c$ for which $a_{i,j} \notin S$.*
4. *Let D contain the integers $\ell \in [n]$ for which there is a vertex $a_{i,j} \notin S$ with $i = \ell$ or $j = \ell$. Then $|D| = n/2$.*
5. *For every $\{i,j\} \in \binom{D}{2}$ we have $a_{i,j} \notin S$.*
6. *The set D forms a clique of size $n/2$ in the graph G_{i^*}, with i^* as in (2).*

Proof. Let S be a vertex cover of G' with $|S \cap A'| \leq k'_A$ and $|S \cap B'| \leq k'_B$.

(1) Suppose there is a bit position $j^* \in [\log t]$ such that S avoids both a vertex in A'_{0,j^*} and in A'_{1,j^*}. Since every set B'_i for $i \in [t]$ is fully adjacent to one of the sets A'_{0,j^*} or A'_{1,j^*} it follows that to cover such edges the set S contains all vertices of B'_i for all $i \in [t]$. Hence $|S \cap B'| \geq t \cdot n > k'_B$, which is a contradiction.

(2) Suppose there are two indices $i_1, i_2 \in [t]$ such that S avoids both a vertex of B'_{i_1} and of B'_{i_2}. Since the numbers i_1 and i_2 differ, there is an index j^* where their binary representations differ. Since B'_{i_1} is fully adjacent to one of the sets (A'_{0,j^*}, A'_{1,j^*}), and B'_{i_2} is fully adjacent to the other set, the fact that S avoids a vertex from both B'_{i_1} and B'_{i_2} implies that the vertex cover S contains all vertices of both A'_{0,j^*} and of A'_{1,j^*}, contributing $2\binom{n}{2}$ vertices to $S \cap A'$. By (1), we know that for all $j \in [\log t] \setminus \{j^*\}$ the set S contains at least $\binom{n}{2}$ vertices from $A'_{0,j} \cup A'_{1,j}$. But then S contains at least $2\binom{n}{2} + ((\log t) - 1)\binom{n}{2} > k'_A$ vertices from A', a contradiction.

(3) Since $|S \cap A'| \leq k'_A = (\binom{n}{2} - \binom{n/2}{2}) + \binom{n}{2} \log t$ and (1) shows that S fully contains at least one of the sets $A'_{0,j}$, $A'_{1,j}$ for every $j \in [\log t]$, it follows that $|S \cap (A' \setminus A'_c)| \geq \binom{n}{2} \log t$ and therefore that S contains at most $\binom{n}{2} - \binom{n/2}{2}$ vertices from A'_c. As $|A'_c| = \binom{n}{2}$, set S avoids at least $\binom{n/2}{2}$ vertices from A'_c.

(4) Every pair $\{i, j\} \in \binom{[n]}{2}$ corresponds to an edge in the complete n-vertex graph. For every vertex $a_{i,j} \notin S$, corresponding to an edge $\{i, j\}$, the vertex cover S contains both vertices b_i and b_j, since those vertices are adjacent to $a_{i,j}$. Any $\binom{n/2}{2}$ edges span at least $n/2$ endpoints, and there are at least $\binom{n/2}{2}$ pairs represented by members of $A'_c \setminus S$ by (3). It follows that the set D defined in the claim statement has size at least $n/2$, and consequently that S contains at least $n/2$ vertices from B'_c. Assume for a contradiction that $|D| > n/2$, implying that S contains more than $n/2$ vertices from B'_c. Since S also contains at least $(t-1) \cdot n$ vertices from $\bigcup_{i=1}^{t} B'_i$, by (2), it follows that $|S \cap B'| > k'_B$, a contradiction. Hence $|D| = n/2$.

(5) The definition of D implies that for every $i \in [n] \setminus D$, we have $a_{i,j} \in S$ for all $i < j \leq n$. Put differently, for each $i \notin D$ we know that for each pair $\{i, j\}$ involving i the corresponding vertex $a_{i,j}$ is contained in S. Since $|D| = n/2$, there are $\binom{n}{2} - \binom{n/2}{2}$ unordered pairs over $[n]$ involving a vertex not in D, and the corresponding $a_{i,j}$ vertices are in S for all these pairs. Since at least $\binom{n/2}{2}$ vertices from A'_c are not in S by (3), it follows that for each pair $\{i, j\} \in \binom{D}{2}$ we must have $a_{i,j} \notin S$.

(6) Assume for a contradiction that $\{i, j\} \in \binom{D}{2}$ with $i < j$ is a pair that is not connected by an edge in G_{i^*}. By (5) we have $a_{i,j} \notin S$. Since $\{i, j\}$ is not an edge of G_{i^*}, the construction of G' has made all vertices in B'_{i^*} adjacent to vertex $a_{i,j}$. Since $a_{i,j}$ is not in S, all vertices of B'_{i^*} must be. But by the choice of i^*, all vertices B'_i for $i \in [t] \setminus \{i^*\}$ are also in S. Hence $|S \cap B'| \geq t \cdot n > k'_B$, a contradiction. It follows that every pair of vertices in D is connected by an edge in G_{i^*}. Since $|D| = n/2$, graph G_{i^*} contains a clique of size $n/2$. ◀

The two claims give the equivalence between the existence of an $n/2$-clique in an input graph and constrained bipartite vertex covers of G'. This completes the proof of Lemma 2. ◀

Using Lemma 2 we can prove a lower bound for the number of vertices in kernels for Constrained Bipartite Vertex Cover. We also need the following simplified version of the complementary witness lemma due to Dell and van Melkebeek [8, Lemma 4].

▶ **Lemma 5.** *Let $L, L' \subseteq \Sigma^*$ be two languages. If there is a constant c and a polynomial-time algorithm that, given a list of $t := s^c$ strings $x_1, \ldots, x_t$, each of length at most s, outputs a string x^* such that:*

- $x^* \in L' \Leftrightarrow \exists i \in [t] : x_i \in L$, *and*
- $|x^*| \in \mathcal{O}(t \log t)$,

then $L \in$ coNP/poly.

As our terminology differs from that of Dell and van Melkebeek, let us point out how the statement above follows from their complementary witness lemma. Dell and van Melkebeek formulate their lemma in terms of (possibly co-nondeterministic) oracle communication protocols. These are two-player protocols in which the player holding the inputs is restricted to polynomial-time computation and the other player is computationally unbounded but does not know the input. The goal is for the first player to correctly decide whether there is at least one input that belongs to L, using as little communication as possible to the second player. The connection comes from the fact that a polynomial-time algorithm as described in Lemma 5 easily gives such a protocol: the first player runs the algorithm on its inputs to obtain the instance x^* that expresses the logical OR of his inputs, sends this small instance to the oracle, which sends back one bit that tells whether or not x^* is contained in L'. Using Lemmata 2 and 5 we now prove the vertex lower bound for Constrained Bipartite Vertex Cover kernelization.

▶ **Theorem 6.** *Let $\varepsilon > 0$ be a real number. If there is a polynomial-time algorithm that reduces any instance (G, A, B, k_A, k_B) of* Constrained Bipartite Vertex Cover *to an equivalent instance (G', A', B', k'_A, k'_B) of the same problem, such that $k'_A \in (k_A)^{\mathcal{O}(1)}, k'_B \in (k_B)^{\mathcal{O}(1)}$, and $|V(G')| \in \mathcal{O}((k_A \cdot k_B)^{1-\varepsilon})$, then* NP $\subseteq$ coNP/poly.

Proof. Assume such a kernelization algorithm exists and call it $\mathcal{K}$. Let $\frac{n}{2}$-Clique be the problem of deciding whether a graph of even order has a clique containing exactly half of its vertices. An easy padding argument proves that $\frac{n}{2}$-Clique is NP-complete. We will show that, using $\mathcal{K}$ and the construction of Lemma 2, we can make an algorithm that compresses the logical OR of a series of $\frac{n}{2}$-Clique instances into an equivalent instance of Constrained Bipartite Vertex Cover whose size satisfies the requirements of Lemma 5. This will show that $\frac{n}{2}$-Clique is contained in coNP/poly and yield NP $\subseteq$ coNP/poly.

Let L be the language over alphabet $\{0, 1\}$ such that a string x is contained in L if and only if it encodes the adjacency matrix of a graph with an even number n of vertices that has a clique of size $n/2$. We construct an algorithm $\mathcal{R}$ that satisfies the conditions of Lemma 5 with this L, while L' is the (classical) language encoding Constrained Bipartite Vertex Cover. The value of c depends on ϵ and will be specified later. On input a list of strings $x_1, \ldots, x_t$, where each string has length at most s and $t = s^c$, algorithm $\mathcal{R}$ proceeds as follows. It first checks which strings encode adjacency matrices of undirected graphs and throws away the others. If the resulting number of instances is not a power of two, it duplicates one instance until the nearest power of two is reached. After this step the input consists of $t' \leq 2t = 2s^c$ strings $x_1, \ldots, x_{t'}$ that encode graphs $G_1, \ldots, G_{t'}$ of at most $n := \lfloor \sqrt{s} \rfloor$ vertices each. As the next step, we pad the instances to ensure they all have the same size. For each input, while it has less than n vertices, add both an isolated vertex and a universal vertex to the graph. This increases the maximum clique size by exactly one, and the graph size by two, so that the new graph has a clique of half its vertices if and only if the original graph has one. We obtain a series of t' graphs $G_1, \ldots, G_{t'}$, each on exactly n vertices, in which the goal is to detect a clique of size $n/2$.

We invoke the construction of Lemma 2 to the graphs $G_1, \ldots, G_{t'}$ and obtain an instance (G, A, B, k_A, k_B) of Constrained Bipartite Vertex Cover of size polynomial in t' and n, such that $k_A \in \mathcal{O}(n^2 \log t')$ and $k_B \in \mathcal{O}(n \cdot t')$, which is a yes-instance if and

only if one of the inputs contains a clique of size $n/2$. Now we apply the hypothetical kernel $\mathcal{K}$ on the instance (G, A, B, k_A, k_B) to obtain an equivalent instance (G', A', B', k'_A, k'_B) of Constrained Bipartite Vertex Cover in which the two parameters have grown only polynomially, i.e., $k'_A \in \mathcal{O}((n^2 \log t')^{\alpha_1})$ and $k'_B \in \mathcal{O}((n \cdot t')^{\alpha_2})$, such that $|V(G')|$ is bounded by $\mathcal{O}((k_A \cdot k_B)^{1-\varepsilon})$ for some fixed $\varepsilon > 0$. The crucial following step is to apply the reduction rule from the simple kernel on this reduced instance: while there is a vertex in B' of degree more than k'_A, we delete it from the graph and decrease the budget by one. Similarly, remove all isolated vertices. Let $(G^*, A^*, B^*, k^*_A, k^*_B)$ be the resulting exhaustively reduced instance. Since the number of vertices does not increase by this step, we know that $|V(G^*)| \in \mathcal{O}((k_A \cdot k_B)^{1-\varepsilon})$. More importantly, we also get a bound on the number of edges. The edges of a bipartite graph can be counted by summing the degrees of all vertices in one partite set. The reduction rule ensures that all vertices of B^* have degree at most $k^*_A \leq k'_A$. We therefore find that the number of edges in G^* is bounded by:

$$\begin{aligned}
\mathcal{O}(|V(G^*)| \cdot k'_A) &\in \mathcal{O}((k_A \cdot k_B)^{1-\varepsilon} \cdot (n^2 \log t')^{\alpha_1}) \\
&\in \mathcal{O}((n^2 \log t' \cdot n \cdot t')^{1-\varepsilon} \cdot (n^2 \log t')^{\alpha_1}) \\
&\in \mathcal{O}((s \log(2s^c) \cdot \sqrt{s} \cdot (2s^c))^{1-\varepsilon} \cdot (s \log(2s^c))^{\alpha_1}) \qquad t' \leq 2s^c, n \leq \sqrt{s}. \\
&\in \mathcal{O}((s^{1.5+c} \cdot c \cdot \log s)^{1-\varepsilon} \cdot (s^{\alpha_1} \cdot c^{\alpha_1} \cdot \log^{\alpha_1} s)) \\
&\in \mathcal{O}((s^{(1-\varepsilon)(1.5+c)+\alpha_1}) \cdot c^{1+\alpha_1} \cdot \log^{(1-\varepsilon)+\alpha_1} s).
\end{aligned}$$

The derivation shows that if we choose $c := \lceil (2.5 + \alpha_1)/\varepsilon \rceil$ (implying that $s^{(1-\varepsilon)(1.5+c)+\alpha_1} < s^{c-1}$), the number of edges in the final graph is $\mathcal{O}(s^{c-1} \cdot \log^{\mathcal{O}(1)} s)$. As G^* has no isolated vertices, the same bound applies to the number of vertices. The instance of Constrained Bipartite Vertex Cover that results from the procedure can be encoded as a string x^* using an adjacency list, which requires $\mathcal{O}(|E(G^*)| \cdot \log |V(G^*)|)$ bits. The string x^* is given as the output. Tracing back the chain of equivalences, we know that x^* is a yes-instance of Constrained Bipartite Vertex Cover if and only if there is string x_i that is a yes-instance of $\frac{n}{2}$-Clique. It is easy to verify that the suggested algorithm $\mathcal{R}$ takes polynomial time. Using the bounds obtained above we find that $|x^*| \in \mathcal{O}(s^{c-1}(\log s)^{\mathcal{O}(1)})$, which is $\mathcal{O}(s^c)$. Hence our choice of c makes algorithm $\mathcal{R}$ satisfy all requirements of Lemma 5, proving that $\frac{n}{2}$-Clique is in coNP/poly and therefore that NP $\subseteq$ coNP/poly. Theorem 6 follows. ◀

4 Sparsification Lower Bound

We establish a sparsification lower bound for the parameterization of Constrained Bipartite Vertex Cover by the total number of vertices in the graph. From this, a general lower bound on the number of edges (which also holds against kernels that increase the budget values arbitrarily) will follow as an easy corollary. We employ the cross-composition framework by Bodlaender et al. [3], which builds on earlier work by several authors [2, 8, 13]. The following two definitions form the core of the framework.

▶ **Definition 7** (Polynomial equivalence relation)**.** An equivalence relation $\mathcal{R}$ on Σ^* is called a *polynomial equivalence relation* if the following conditions hold:

1. There is an algorithm that, given two strings $x, y \in \Sigma^*$, decides whether x and y belong to the same equivalence class in time polynomial in $|x| + |y|$.
2. For any finite set $S \subseteq \Sigma^*$ the equivalence relation $\mathcal{R}$ partitions the elements of S into a number of classes that is polynomially bounded in the size of the largest element of S.

▶ **Definition 8** (Cross-composition). Let $L \subseteq \Sigma^*$ be a language, let $\mathcal{R}$ be a polynomial equivalence relation on Σ^*, let $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem, and let $f : \mathbb{N} \to \mathbb{N}$ be a function. An OR-*cross-composition of* L *into* $\mathcal{Q}$ (with respect to $\mathcal{R}$) *of cost* $f(t)$ is an algorithm that, given t instances $x_1, x_2, \ldots, x_t \in \Sigma^*$ of L belonging to the same equivalence class of $\mathcal{R}$, takes time polynomial in $\sum_{i=1}^{t} |x_i|$ and outputs an instance $(y, k) \in \Sigma^* \times \mathbb{N}$ such that:

- The parameter k is bounded by $\mathcal{O}(f(t) \cdot (\max_i |x_i|)^c)$, where c is some constant independent of t.
- $(y, k) \in \mathcal{Q}$ if and only if there is an $i \in [t]$ such that $x_i \in L$.

The following theorem shows how these concepts give kernelization lower bounds.

▶ **Theorem 9** ([3, 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.

We use a restricted version of Constrained Bipartite Vertex Cover as the source problem in a cross-composition:

Equally Constrained Bipartite Vertex Cover
Input: A bipartite graph G with partite sets A and B such that $|A| = |B|$ is even.
Question: Is there a vertex cover S for G such that $|S \cap A| \leq |A|/2$ and $|S \cap B| \leq |B|/2$?

▶ **Proposition 10.** Equally Constrained Bipartite Vertex Cover *is* NP*-complete.*

Proposition 10 follows from a simple padding argument.

▶ **Theorem 11.** Constrained Bipartite Vertex Cover *parameterized by the number of vertices n does not have a generalized kernel of bitsize $\mathcal{O}(n^{2-\varepsilon})$, for any $\varepsilon > 0$, unless* NP $\subseteq$ coNP/poly.

Proof. With the aim of giving a cross-composition, we start by defining a polynomial equivalence relation $\mathcal{R}$ on inputs of Equally Constrained Bipartite Vertex Cover. We define any two strings that do not encode valid instances of Equally Constrained Bipartite Vertex Cover to be equivalent. Two well-formed instances (G_1, A_1, B_1) and (G_2, A_2, B_2) are equivalent if and only if $|A_1| = |B_1| = |A_2| = |B_2|$. It is easy to verify that this is a polynomial equivalence relation. We proceed to give a cross-composition of cost $\mathcal{O}(\sqrt{t} \cdot \log t) \in \mathcal{O}(t^{\frac{1}{2}+o(1)})$ from Equally Constrained Bipartite Vertex Cover into Constrained Bipartite Vertex Cover parameterized by the number of vertices n. Strings that do not encode valid inputs can be recognized in polynomial time, and can be reduced to a trivial NO-instance. In the remainder, we focus on the case that the input encodes a list of instances $(G_1, A_1, B_1), \ldots, (G_t, A_t, B_t)$ such that all partite sets of all input graphs have the same number of n vertices. If t is not equal to 2^{2r} for some integer r, then we can repeatedly duplicate an instance until this holds. This only blows up the size of the input by a constant factor and does not change whether there is at least one YES-instance in the inputs. In the remainder we can assume that $t = 2^{2r}$ for some r, implying that both $\sqrt{t}$ and $\log \sqrt{t}$ are integers. We can therefore index the inputs as $(G_{i,j}, A_{i,j}, B_{i,j})$ for $i, j \in [\sqrt{t}]$. For ease of presentation, the vertices in each partite set are identified with the integers in the range $[n]$. We can assume $n \geq 3$, as the instances are trivially solvable otherwise. We construct an instance (G', A', B', k'_A, k'_B) of Constrained Bipartite Vertex Cover that expresses the logical OR of the input instances, as follows.

1. For each $i \in [\sqrt{t}]$, add a vertex set A'_i consisting of n vertices $a_{i,1}, \ldots, a_{i,n}$ to A'.
2. For each $i \in [\sqrt{t}]$, add a vertex set B'_i consisting of n vertices $b_{i,1}, \ldots, b_{i,n}$ to B'.
3. The adjacency information of the input graphs is embedded into G'. For $i, j \in [\sqrt{t}]$, for $p, q \in [n]$, do the following. If $\{p, q\} \in E(G_{i,j})$, then make vertex $a_{i,p}$ adjacent to $b_{j,q}$. Afterward we have for every $i, j \in [\sqrt{t}]$ that the graph $G'[A'_i \cup B'_j]$ is isomorphic to $G_{i,j}$.
4. For each $i \in [\sqrt{t}]$, add a vertex set C'_i consisting of $n/2-1$ *checking vertices* to the partite set A'. Make all vertices of C'_i adjacent to all vertices of B'_i.
5. Define $w := \sqrt{t} \cdot n^2$. For each $j \in [\log \sqrt{t}]$, for each $x \in \{0, 1\}$, add a vertex set $B'_{x,j}$ to the partite set B' consisting of w vertices. For all $i \in [\sqrt{t}]$, make all vertices of $B'_{x,j}$ adjacent to all vertices of A'_i if the j-th bit of the binary representation of number i is an x.

This concludes the description of the graph G'. It is easy to verify that $n' := |V(G')| = 2n\sqrt{t} + (n/2-1)\sqrt{t} + 2w \log \sqrt{t} \in \mathcal{O}(n^2 \sqrt{t} \log t)$. The construction can be performed in polynomial time. Define $k'_A := n\sqrt{t} - 1$ and $k'_B := n/2 + (\sqrt{t}-1) \cdot n + w \cdot \log \sqrt{t}$. We will prove that (G', A', B', k'_A, k'_B) is a YES-instance of Constrained Bipartite Vertex Cover if and only if one of the inputs is a YES-instance.

▶ **Claim 12.** *If there are indices $i^*, j^* \in [\sqrt{t}]$ such that G_{i^*,j^*} has an $(n/2, n/2)$-constrained vertex cover, then G' has a (k'_A, k'_B)-constrained vertex cover.*

Proof. Suppose there are sets $A^* \subseteq A_{i^*,j^*}$ and $B^* \subseteq B_{i^*,j^*}$, each of size at most $n/2$, which together form a vertex cover of G_{i^*,j^*}. We build a (k'_A, k'_B)-constrained vertex cover S for G'.

- Add all vertices $a_{i^*,\ell}$ to S for which $\ell \in A^*$. This contributes $n/2$ vertices to $S \cap A'$.
- Add all vertices $b_{j^*,\ell}$ to S for which $\ell \in B^*$. This contributes $n/2$ vertices to $S \cap B'$.
- Add all vertices of all sets A'_i with $i \in [\sqrt{t}] \setminus \{i^*\}$ to S. This contributes $(\sqrt{t}-1)n$ vertices to $S \cap A'$.
- Add all vertices of all sets B'_j with $j \in [\sqrt{t}] \setminus \{j^*\}$ to S. This contributes $(\sqrt{t}-1)n$ vertices to $S \cap B'$.
- Add all vertices C'_{j^*} to S. This contributes $n/2-1$ vertices to $S \cap A'$.
- For each $j \in [\log \sqrt{t}]$, if the j-th bit of i^* is a zero then add all vertices of $B'_{0,j}$ to S. Otherwise add all vertices of $B'_{1,j}$ to S. This contributes $w \log \sqrt{t}$ vertices to $S \cap B'$.

It is easy to verify that S contains k'_A vertices from A' and k'_B vertices from B'. It remains to check that S is a vertex cover of G'. We discuss the edges of G' in the order in which they were added to the graph by the construction.

- The edges of the induced subgraph $G'[A'_{i^*} \cup B'_{j^*}]$ are covered since S includes the vertices corresponding to A^* and B^*, which form a vertex cover for G_{i^*,j^*}. Recall that G_{i^*,j^*} is isomorphic to $G'[A'_{i^*} \cup B'_{j^*}]$. Edges between A'_i and B'_j for $i \neq i^*$ or $j \neq j^*$ are covered because S includes all vertices of A'_i for $i \neq i^*$, and all vertices of B'_j for $j \neq j^*$.
- The edges between the checking vertices C'_{j^*} and B'_{j^*} are covered because S includes C'_{j^*}. The edges between C'_j and B'_j for $j \neq j^*$ are covered because S contains B'_j.
- The edges between A'_{i^*} and sets $B'_{x,j}$ for $x \in \{0, 1\}$, $j \in [\log \sqrt{t}]$ are covered because S contains all sets $B'_{x,j}$ whose bit value matches that of the binary representation of i^*. For $i \neq i^*$, the edges between A'_i and sets $B'_{x,j}$ are covered because S contains A'_i.

As S covers all edges of G', the claim follows. ◀

▶ **Claim 13.** *For any inclusionwise-minimal (k'_A, k'_B)-constrained vertex cover S of G', the following holds.*

1. *For every $j \in [\log \sqrt{t}]$ the set S contains all vertices of $B'_{0,j}$ or all vertices of $B'_{1,j}$.*
2. *For every $j \in [\log \sqrt{t}]$ we have $S \cap B'_{0,j} = \emptyset$ or $S \cap B'_{1,j} = \emptyset$.*
3. *There is an index $\ell \in [\sqrt{t}]$ such that $B'_\ell \setminus S \neq \emptyset$.*
4. *The set S avoids at least $n/2$ vertices from the set $\bigcup_{i \in [\sqrt{t}]} A'_i$.*

5. *There is an index $i^* \in [\sqrt{t}]$ such that S contains A'_i for all $i \in [\sqrt{t}] \setminus \{i^*\}$.*
6. *There is an index $j^* \in [\sqrt{t}]$ such that S contains B'_j for all $j \in [\sqrt{t}] \setminus \{j^*\}$.*
7. *Let i^* and j^* be as defined in (5) and (6). Then the graph G_{i^*,j^*} has a vertex cover containing at most $n/2$ vertices from each partite set.*

Proof. Let S be an inclusionwise-minimal (k'_A, k'_B)-constrained vertex cover of G'.

(1) Assume for a contradiction that there is an index $j \in [\log \sqrt{t}]$ such that S avoids both a vertex of $B'_{0,j}$ and of $B'_{1,j}$. To cover all the edges between $B'_{0,j}$ and sets A'_i for $i \in [\sqrt{t}]$, the set S must contain all sets A'_i for which the j-th bit of i is a zero. Similarly, S contains all sets A'_i for which the j-th bit is a one, to cover the edges between $B'_{0,j}$ and the A'_i's. So $S \supseteq \bigcup_{i \in [\sqrt{t}]} A'_i$, showing that $|S \cap A'| \geq n \cdot \sqrt{t} > k'_A$, a contradiction.

(2) Assume for a contradiction that there is an index $j \in [\log \sqrt{t}]$ such that $S \cap B'_{0,j} \neq \emptyset$ and $S \cap B'_{1,j} \neq \emptyset$. Observe that all vertices in $B'_{x,j}$ are false twins for $x \in \{0,1\}$. Hence if one vertex v of $B'_{x,j}$ is contained in the inclusionwise-minimal vertex cover S, then there is a neighbor u of v that is not in S, implying that all other vertices of $B'_{x,j}$ are also adjacent to u. Therefore all vertices of $B'_{0,j}$ and $B'_{1,j}$ are contained in S. Together with the fact that S contains at least one of the sets $B'_{0,j'}, B'_{1,j'}$ fully for all $j' \in [\sqrt{t}] \setminus \{j\}$, by (1), we find that $|S \cap B'| \geq (\log \sqrt{t} - 1)w + 2w = w + w \cdot \log \sqrt{t} = n^2\sqrt{t} + w \cdot \log \sqrt{t} > k'_B$; a contradiction.

(3) Suppose that $B'_\ell \subseteq S$ for all $\ell \in [\sqrt{t}]$, contributing $n \cdot \sqrt{t}$ vertices to $S \cap B'$. By (1) we know that S contains at least $w \cdot \log \sqrt{t}$ vertices from the sets $B'_{x,i}$. Hence $|S \cap B'| \geq n \cdot \sqrt{t} + w \cdot \log \sqrt{t} > k'_B$, a contradiction.

(4) Suppose that $|\bigcup_{i \in [\sqrt{t}]} A'_i \setminus S| < n/2$. Then $|\bigcup_{i \in [\sqrt{t}]} A'_i \cap S| > n \cdot \sqrt{t} - n/2$. By (3) we know that there is an index $i^* \in [\sqrt{t}]$ such that S avoids at least one vertex from B'_{i^*}. Since all vertices of C'_{i^*} are adjacent to all vertices of B'_{i^*}, this implies that S contains all $n/2 - 1$ vertices of C'_{i^*}. Since C'_{i^*} also belongs to the A' partite set, this shows that $|S \cap A'| > (n\sqrt{t} - n/2) + (n/2 - 1) = n\sqrt{t} - 1 = k'_A$, a contradiction.

(5) Consider the number i^* whose j-th bit is a one if S avoids $B'_{0,j}$ and is a zero otherwise. By (2), in the second case S avoids $B'_{1,j}$. For each $i \neq i^*$, the binary representation of i differs from that of i^* in at least one bit. Suppose that the j-th bit of i is a zero, and the j-th bit of i^* is a one. Then S avoids $B'_{0,j}$ and A'_i is adjacent to $B'_{0,j}$ by construction. Consequently, all vertices of A'_i are contained in S. Similarly, if the j-th bit of i is a one and the j-th bit of i^* is a zero, then S avoids $B'_{1,j}$ while $B'_{1,j}$ is adjacent to A'_i; hence A'_i is fully contained in S. It follows that for every index $i \neq i^*$ the set A'_i is contained in S.

(6) Suppose that there are two indices j', j'' such that S avoids a vertex of both $B'_{j'}$ and $B'_{j''}$. Then S contains all vertices of $C'_{j'}$ and $C'_{j''}$. By (5) we know that there is an index $i^* \in [\sqrt{t}]$ such that S contains $\bigcup_{i \in [\sqrt{t}] \setminus \{i^*\}} A'_i$. Hence $|S \cap A'| \geq 2(n-1) + (\sqrt{t} - 1) \cdot n > k'_A$, a contradiction. The last step uses the fact that $n \geq 3$.

(7) Let i^* and j^* be as defined in (5) and (6), implying that S contains all sets A'_i for $i \in [\sqrt{t}] \setminus \{i^*\}$ and all sets B'_j for $j \in [\sqrt{t}] \setminus \{j^*\}$. By (4) the set S avoids at least $n/2$ vertices from $\bigcup_{i \in [\sqrt{t}]} A'_i$, which implies that $|S \cap A'_{i^*}| \leq n/2$, since S fully contains the other sets A'_i. We proceed to show that $|S \cap B'_{j^*}| \leq n/2$. To see that, observe that S fully contains $(\sqrt{t} - 1) \cdot n$ vertices from $\bigcup_{j \in [\sqrt{t}] \setminus \{j^*\}} B'_j$. In addition, S contains at least $w \cdot \log \sqrt{t}$ vertices from $\bigcup_{j \in [\log \sqrt{t}], x \in \{0,1\}} B'_{x,j}$, by (1). Since the total size of $S \cap B'$ is at most $k'_B = n/2 + (\sqrt{t} - 1) \cdot n + w \cdot \log \sqrt{t}$ we find that $|S \cap B'_{j^*}| \leq n/2$. Now define $S^* := S \cap (A'_{i^*} \cup B'_{j^*})$. It follows that S^* is a vertex cover of the graph $G'[A'_{i^*} \cup B'_{j^*}]$ containing at most $n/2$ vertices from each partite set. Since $G'[A'_{i^*} \cup B'_{j^*}]$ is isomorphic to the input graph G_{i^*,j^*} by construction of G', the claim follows. ◀

The last item of Claim 13 shows that if there is a (k'_A, k'_B)-constrained vertex cover of G', then one of the input instances has answer YES: if a vertex cover with such size constraints exists, then there is also an inclusionwise-minimal vertex cover satisfying the same size bounds, causing one of the input graphs to have a vertex cover containing at most $n/2$ vertices from each partite set. By the definition of Equally Constrained Bipartite Vertex Cover, this certifies the YES-answer for that input. Together with the first claim, we have therefore established that the composed instance acts as the logical OR of the inputs. The construction thus satisfies all requirements of a cross-composition of Equally Constrained Bipartite Vertex Cover into Constrained Bipartite Vertex Cover parameterized by the number of vertices. The parameter of the composed instance is $n' \in \mathcal{O}(n^2\sqrt{t}\log t) \in \mathcal{O}(n^{O(1)} \cdot t^{\frac{1}{2}+o(1)})$, showing that the cross-composition has cost $f(t) \in \mathcal{O}(t^{\frac{1}{2}+o(1)})$. As the starting problem is NP-complete (Proposition 10), Theorem 11 now follows from Theorem 9. ◀

▶ **Corollary 14.** *Let $\varepsilon > 0$ be a real number. If there is a polynomial-time algorithm that reduces any instance (G, A, B, k_A, k_B) of* Constrained Bipartite Vertex Cover *to an equivalent instance of the same problem with $\mathcal{O}((k_A \cdot k_B)^{1-\varepsilon})$ edges, then* NP $\subseteq$ coNP/poly.

Proof. Suppose such a kernelization algorithm exists and call it $\mathcal{K}$. Using $\mathcal{K}$ we create a generalized kernelization $\mathcal{A}$ of subquadratic size for Constrained Bipartite Vertex Cover parameterized by the number of vertices n. Presented with an instance (G, A, B, k_A, k_B), the algorithm does the following. Let n be the number of vertices in G. If $k_A \geq n$ or $k_B \geq n$, then the answer is trivially YES as we may take all of A or all of B to form the desired constrained vertex cover. We can therefore output a constant-size YES-instance as the output of the compression. In the remaining cases we know $k_A, k_B < n$. The algorithm then invokes $\mathcal{K}$ on the input, obtaining an equivalent instance (G', A', B', k'_A, k'_B) where $|E(G')| \in \mathcal{O}((k_A \cdot k_B)^{1-\varepsilon}) \in \mathcal{O}((n \cdot n)^{1-\varepsilon}) \in \mathcal{O}(n^{2-2\varepsilon})$. After removing isolated vertices from the graph, which do not affect the answer, the number of vertices in G' is at most twice the number of edges, which is $\mathcal{O}(n^{2-2\varepsilon})$. This instance is encoded using an adjacency list representation. In general, an adjacency list encoding of a graph uses $\mathcal{O}(|V| + |E| \log |V|)$ bits. In this case, we find that G' can be encoded in $\mathcal{O}(n^{2-2\varepsilon} \log(n^{2-2\varepsilon})) \in \mathcal{O}(n^{2-\varepsilon})$ bits. After encoding the values of k'_A and k'_B in binary, which does not exceed this space bound, the algorithm outputs the resulting instance. Since it is equivalent to the input instance, this is a generalized kernel for Constrained Bipartite Vertex Cover. As the size is $\mathcal{O}(n^{2-\varepsilon})$ for some positive ε, by Theorem 11 we now obtain NP $\subseteq$ coNP/poly. ◀

5 Conclusion

In this paper we presented two kernelization lower bounds for Constrained Bipartite Vertex Cover. We proved that it is unlikely that the $\Theta(k_A \cdot k_B)$ bound on the number of vertices and edges from the easy kernel can be significantly improved. The easy kernel is therefore close to optimal when bounding the kernel size in terms of the product $k_A \cdot k_B$. Our results do not rule out the existence of kernels for Constrained Bipartite Vertex Cover with $\mathcal{O}(k_A + k_B)$ vertices, which we leave as an open problem.

Acknowledgments. The author is grateful to Marcin and Michał Pilipczuk for bringing the problem to his attention.

References

1 Guoqiang Bai and Henning Fernau. Constraint bipartite vertex cover: simpler exact algorithms and implementations. *J. Comb. Optim.*, 23(3):331–355, 2012. doi:10.1007/s10878-010-9289-7.

2 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.

3 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.

4 Jianer Chen and Iyad A. Kanj. Constrained minimum vertex cover in bipartite graphs: complexity and parameterized algorithms. *J. Comput. Syst. Sci.*, 67(4):833–847, 2003. doi:10.1016/j.jcss.2003.09.003.

5 Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001. doi:10.1006/jagm.2001.1186.

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 Holger Dell and Dániel Marx. Kernelization of packing problems. In *Proc. 23rd SODA*, pages 68–81, 2012. doi:10.1137/1.9781611973099.6.

8 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.

9 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

10 R.C. Evans. Testing repairable RAMs and mostly good memories. In *Proc. IEEE International Test Conference*, pages 49–55, 1981.

11 Michael R. Fellows, Bart M. P. Jansen, and Frances Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *European J. Combin.*, 34(3):541–566, 2013. doi:10.1016/j.ejc.2012.04.008.

12 Henning Fernau and Rolf Niedermeier. An efficient exact algorithm for constraint bipartite vertex cover. *J. Algorithms*, 38(2):374–410, 2001. doi:10.1006/jagm.2000.1141.

13 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.

14 Danny Hermelin and Xi Wu. Weak compositions and their applications to polynomial lower bounds for kernelization. In *Proc. 23rd SODA*, pages 104–113, 2012. doi:10.1137/1.9781611973099.9.

15 Bart M. P. Jansen. On sparsification for computing treewidth. *Algorithmica*, 71(3):605–635, 2015. doi:10.1007/s00453-014-9924-2.

16 Stefan Kratsch, Geevarghese Philip, and Saurabh Ray. Point line cover: The easy kernel is essentially tight. In *Proc. 25th SODA*, pages 1596–1606. SIAM, 2014. doi:10.1137/1.9781611973402.116.

17 S.-Y. Kuo and W.K. Fuchs. Efficient spare allocation for reconfigurable arrays. *IEEE Design Test of Computers*, 4(1):24–31, 1987. doi:10.1109/MDT.1987.295111.

18 Marcin Pilipczuk. Banach's algorithmic corner. http://corner.mimuw.edu.pl/?p=539, October 2013.

19 Chee-Keng Yap. Some consequences of non-uniform conditions on uniform classes. *Theor. Comput. Sci.*, 26:287–300, 1983. doi:10.1016/0304-3975(83)90020-8.

Separation Between Read-once Oblivious Algebraic Branching Programs (ROABPs) and Multilinear Depth Three Circuits

Neeraj Kayal[1], Vineet Nair[2], and Chandan Saha[3]

1 **Microsoft Research India**
`neeraka@microsoft.com`
2 **Indian Institute of Science, India**
`vineet.nair@csa.iisc.ernet.in`
3 **Indian Institute of Science, India**
`chandan@csa.iisc.ernet.in`

Abstract

We show an exponential separation between two well-studied models of algebraic computation, namely read-once oblivious algebraic branching programs (ROABPs) and multilinear depth three circuits. In particular we show the following:

1. There exists an explicit n-variate polynomial computable by linear sized multilinear depth three circuits (with only two product gates) such that every ROABP computing it requires $2^{\Omega(n)}$ size.
2. Any multilinear depth three circuit computing $\mathrm{IMM}_{n,d}$ (the iterated matrix multiplication polynomial formed by multiplying d, $n \times n$ symbolic matrices) has $n^{\Omega(d)}$ size. $\mathrm{IMM}_{n,d}$ can be easily computed by a $\mathrm{poly}(n, d)$ sized ROABP.
3. Further, the proof of 2 yields an exponential separation between multilinear depth four and multilinear depth three circuits: There is an explicit n-variate, degree d polynomial computable by a $\mathrm{poly}(n, d)$ sized multilinear depth four circuit such that any multilinear depth three circuit computing it has size $n^{\Omega(d)}$. This improves upon the quasi-polynomial separation result by Raz and Yehudayoff [2009] between these two models.

The hard polynomial in 1 is constructed using a novel application of expander graphs in conjunction with the evaluation dimension measure used previously in Nisan [1991], Raz [2006,2009], Raz and Yehudayoff [2009], and Forbes and Shpilka [2013], while 2 is proved via a new adaptation of the dimension of the partial derivatives measure used by Nisan and Wigderson [1997]. Our lower bounds hold over any field.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases multilinear depth three circuits, read-once oblivious algebraic branching programs, evaluation dimension, skewed partial derivatives, expander graphs, iterated matrix multiplication

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.46

1 Introduction

Proving lower bounds and separating complexity classes lie at the heart of complexity theory. In algebraic complexity, separating classes VP and VNP (the algebraic analogues of P and NP) equates to proving super-polynomial lower bounds for arithmetic circuits. Another prominent and pertinent problem is polynomial identity testing (PIT). Here we are given an arithmetic circuit computing a multivariate polynomial over some field and the problem is

© Neeraj Kayal, Vineet Nair, and Chandan Saha;
licensed under Creative Commons License CC-BY

to determine whether the polynomial is identically zero. Polynomial time randomized PIT follows easily from [6, 33, 37]. PIT is one of the very few natural problems in BPP (in fact, in co-RP) not known to be in P. Showing arithmetic circuit lower bounds and derandomizing PIT are closely related: [17] showed that a polynomial time PIT over integers implies a super-polynomial arithmetic circuit lower bound for the family of permanent polynomials or NEXP $\not\subseteq$ P/poly. [15, 1] showed that a polynomial time blackbox PIT (meaning, we are only allowed to evaluate the circuit at points from $\mathbb{F}^n$, where n is the number of inputs and $\mathbb{F}$ the underlying field) implies exponential lower bounds for circuits computing polynomials whose coefficients can be computed in PSPACE. Conversely, [17] also showed that a super-polynomial (exponential) circuit lower bound for any family of exponential-time computable multilinear polynomials implies a sub-exponential (quasi-polynomial) time algorithm for PIT, in fact blackbox PIT, using Nisan-Wigderson generators [24] and Kaltofen's [18] polynomial factorization algorithm. [8] showed a similar connection between lower bounds and PIT for low depth circuits. They proved lower bounds for bounded depth circuits imply efficient PIT for bounded depth circuits computing polynomials with low individual degree. So, in this certain sense the complexity of proving strong lower bounds and devising efficient PIT algorithms are quite similar. Derandomizing PIT is also interesting in its own right. It is well-known that such a derandomization would imply the problem of checking existence of a perfect matching in a given graph is in NC [35].

Research over the the past several years has made notable progress on both lower bounds and PIT for interesting special cases of arithmetic circuits and helped identify the frontiers of our current knowledge. In particular, we understand better the reason why super-polynomial lower bounds and poly-time PIT have remained elusive even for depth three circuits: An exponential lower bound (similarly, a poly-time blackbox PIT) for depth three circuits over fields of characteristic zero implies an exponential lower bound (similarly, quasi-polynomial-time PIT) for general circuits [12]. For more on arithmetic circuit lower bounds and PIT refer to the surveys [34], [4], [30, 21], [31, 32].

A potentially useful and interesting restriction to consider at depth three is multilinearity (meaning, every product gate computes a multilinear polynomial). Most of the hard polynomials used in the literature are multilinear, e.g. determinant, permanent, iterated matrix multiplication, Nisan-Wigderson polynomials etc. So, it is worthwhile to develop a fuller understanding of multilinear models [27, 26, 28, 29, 7]. We do know of strong lower bounds for multilinear depth three circuits due to [29] and also this paper (Theorem 9), but as yet no efficient (meaning, quasi-polynomial) PIT is known for this model. One reason for this is the absence of hardness versus randomness tradeoff results for bounded depth multilinear circuits. Recently, [5] has given a sub-exponential time blackbox PIT algorithm for multilinear depth three circuits using recently found quasi-polynomial blackbox PIT for another model, namely read-once oblivious algebraic branching programs (ROABPs) [10, 2] (Definition 2), thereby connecting these two interesting models of computation. Could there be a more efficient reduction from multilinear depth three circuits to ROABPs? If so then that would readily imply an efficient PIT algorithm for multilinear depth three circuits. This question has lead us to this work.

Related work and motivation. The model ROABP (see Definition 2) has been studied intensely in the recent years in the context of black-box PIT, equivalently *hitting-set generators* (Definition 20). This has resulted in deterministic, quasi-polynomial time hitting-set generators for ROABPs [2, 10] and other associated models like set-multilinear algebraic branching programs [9, 10] (a special case of which is set-multilinear depth three circuits [3, 10], see

also Definition 4), non-commutative algebraic branching programs [10] and diagonal depth-3 circuits [3, 10]. Quite recently, [5] has given a $2^{\tilde{O}(n^{\frac{2}{3}(1+\delta)})}$ time hitting-set generator for multilinear depth three circuits of size at most 2^{n^δ} by 'reducing' a multilinear depth three circuit to a collection of ROABPs and 'putting together' the hitting-sets of the ROABPs. This 'putting together' process raises the hitting-set complexity from quasi-polynomial (for a single ROABP) to sub-exponential (for a composition of several ROABPs). Had it been the case that a multilinear depth three circuit can be directly reduced to a single small size ROABP, an efficient hitting set for the former would have ensued immediately from [2, 10]. One of the results in the paper (Theorem 7), rules out this possibility. In fact, Theorem 7 shows something stronger as described below.

A closer look at [2] and [5] reveals an interesting, and potentially useful, intermediate model that we call *superposition of (two or more) set-multilinear depth three circuits* (see Definition 5). An example of superposition of two set-multilinear depth three circuits is,

$$C(X,Y) = (1 + 3x_1 + 5y_2)(4 + x_2 + y_1) + (6 + 9x_1 + 4y_1)(2 + 5x_2 + 3y_2).$$

The variable sets $X = \{x_1, x_2\}$ and $Y = \{y_1, y_2\}$ are completely disjoint and are called the *base sets* of $C(X,Y)$. When projected on X variables (i.e after putting the Y variables to zero), $C(X,Y)$ is a set-multilinear depth three circuit in the X variables. A similar thing is true for the Y variables. Thus, every base set is associated with a set-multilinear depth three circuit and vice versa. Any multilinear depth three circuit can be trivially viewed as a superposition of n set-multilinear depth three circuits with single variable in every base set, where n is the number of variables. A crucial observation in [5] is that every multilinear depth three circuit is *"almost"* a superposition of n^ϵ set-multilinear depth three circuits for some $\epsilon < 1$, and further the associated n^ϵ base sets can be found in sub-exponential time using k-wise independent hash functions. Once we know the $r = n^\epsilon$ base sets corresponding to r set-multilinear depth three circuits whose superposition forms a circuit of size s, finding a hitting set for the circuit in time $s^{r.\log s}$ follows easily by taking a direct product of hitting sets for r many ROABPs (in fact, set-multilinear depth three circuits). We think a useful model to consider at this juncture is superposition of constantly many set-multilinear depth three circuits with *unknown* base sets. In this case knowing the $r = O(1)$ base sets readily gives us a quasi-polynomial time hitting set, but finding these base sets from a given circuit is NP-hard for $r \geq 3$ (as we show in Observation 6), which rules out the possibility of knowing the base-sets even if we are allowed to see the circuit (as in the *white-box* case). Indeed, even in this special case where the given multilinear depth three circuit is promised to be a superposition of constantly many (say, 2) set-multilinear depth three circuits, the algorithm in [5] finds and works with many base sets and the resulting hitting set complexity grows to roughly $\exp(\sqrt{n})$. Could it be that superposition of constantly many set-multilinear depth three circuits efficiently reduce to ROABPs? Unfortunately, the answer to this also turns out to be negative as Theorem 7 gives an explicit example of a superposition of *two* set-multilinear depth three circuit computing an n-variate polynomial f such that any ROABP computing f has width $2^{\Omega(n)}$.

While comparing two models (here multilinear depth three circuits and ROABPs), it is desirable to show a separation in both directions whenever an efficient reduction from one to the other seems infeasible. In this sense, we show a complete seperation between the models under consideration by giving an explicit polynomial computable by a polynomial sized ROABP such that every multilinear depth three circuit computing it requires exponential size. In fact, this explicit polynomial is simply the Iterated Matrix Multiplication $\text{IMM}_{n,d}$ – the $(1,1)$-th entry of a product of d $n \times n$ symbolic matrices (Theorem 9). $\text{IMM}_{n,d}$ can

be easily computed by a polynomial-sized ROABP (see Observation 10). Although, a $2^{\Omega(d)}$ lower bound for multilinear depth three circuit computing Det_d is known [29], this does not imply a lower bound for $\mathrm{IMM}_{n,d}$ (despite the fact that Det and IMM are both complete for algebraic branching programs (ABPs) [22]) as the projection from IMM to Det can make the circuit non-multilinear. Another related work by [7] showed a separation between multilinear ABPs and multilinear formulas by exhibiting an explicit polynomial (namely, an *arc-full-rank* polynomial) that is computable by a linear size multilinear ABP but requires super-polynomial size multilinear formulas. But again multilinearity of a circuit can be lost when IMM is projected to arc-full-rank polynomials, and hence this result too does not imply a lower bound for IMM. An extension of Theorem 9 to a super-polynomial lower bound for multilinear formulas computing IMM will have interesting consequences in separating noncommutative formulas and noncommutative ABPs. In a contemporary work [20], some of the authors of this work and Sébastien Tavenas have been able to show an $n^{\Omega(\sqrt{d})}$ lower bound for multilinear depth four circuits computing $\mathrm{IMM}_{n,d}$ by significantly extending a few of the ideas present in this work and building upon (thereby improving) the work of [11]. Thus, in summary the models poly-sized ROABPs and poly-sized multilinear depth three ciruits have provably different computational powers, although they share a non-trivial intersection as poly-sized set-multilinear depth three circuits is harbored in both.

An interesting outcome of the proof of the lower bound for multilinear depth three circuits computing IMM is an exponential separation between multilinear depth three and multilinear depth four circuits. Previously, [29] showed a super-polynomial separation between multilinear constant depth h and depth $h+1$ circuits, which when applied to the depth three versus four setting gives a quasi-polynomial seperation between the two models. In comparison, Theorem 11 gives an exponential separation.

The models and our results. We define the relevant models and state our results now.

▶ **Definition 1** (Algebraic Branching Program)**.** An Algebraic Branching Program(ABP) in the variables $X = \{x_1, x_2, ..., x_n\}$ is a directed acyclic graph with a source vertex s and a sink vertex t. It has $(d+1)$ sets or layers of vertices $V_1, V_2, ..., V_{d+1}$, where V_1 and V_{d+1} contain only s and t respectively. The width of an ABP is the maximum number of vertices in any of the $(d+1)$ layers. All the edges in an ABP are such that an edge starts from a vertex in V_i and is directed to a vertex in V_{i+1}, where V_i belongs to the set $\{V_1, V_2, ..., V_d\}$. The edges in an ABP are labelled by polynomials over a base field $\mathbb{F}$. The weight of a path between any two vertices u and v in an ABP is computed by taking the product of the edge labels on the path from u to v. An ABP computes the sum of the weights of all the paths from s to t.

Note that in another standard definition of an ABP, the edges are labeled by linear polynomials over a base field $\mathbb{F}$. A special kind of ABP, namely ROABP, is defined in [10].

▶ **Definition 2** (Read-Once Oblivious Algebraic Branching Program)**.** A Read-Once Oblivious Algebraic Branching Program(ROABP) over a field $\mathbb{F}$ has an associated permutation $\pi : [n] \to [n]$ of the variables in X. The number of variables is equal to the number of layers of vertices minus one, i.e. $n = (d+1) - 1 = d$. The label associated with an edge from a vertex in V_i to a vertex in V_{i+1} is an univariate polynomial over $\mathbb{F}$ in the variable $x_{\pi(i)}$.

▶ **Definition 3** (Multilinear depth 4 and depth 3 circuits)**.** A circuit $C = \sum_{i=1}^{s} \prod_{j=1}^{d_i} Q_{ij}(X_j^i)$ is a multilinear depth four ($\Sigma\Pi\Sigma\Pi$) circuit in X variables over a field $\mathbb{F}$, if $X = \uplus_{j=1}^{d_i} X_j^i$ and $Q_{ij} \in \mathbb{F}[X_j^i]$ is a multilinear polynomial for every $i \in [s]$ and $j \in [d_i]$. If Q_{ij}'s are linear polynomials then C is a multilinear depth three ($\Sigma\Pi\Sigma$) circuit. The parameter s is the *top fan-in* of C.

▶ **Definition 4** (Set-multilinear depth three circuit)**.** A circuit $C = \sum_{i=1}^{s} \prod_{j=1}^{d} l_{ij}(X_j)$ is a set-multilinear depth three ($\Sigma\Pi\Sigma$) circuit in X variables over a field $\mathbb{F}$, if $X = \uplus_{j=1}^{d} X_j$ and $l_{ij} \in \mathbb{F}[X_j]$ is a linear polynomial for every $i \in [s]$ and $j \in [d]$. The sets $X_1, X_2, ..., X_d$ are called the *colors* of X. If $|X_j| = 1$ for every $j \in [d]$ then we say X has singleton colors and C is a set-multilinear depth three circuit with singleton colors.

As a bridge between multilinear and set-multilinear depth three circuits we define a model called superposition of set-multilinear depth three circuits.

▶ **Definition 5** (Superposition of set-multilinear depth three circuits)**.** A multilinear depth three ($\Sigma\Pi\Sigma$) circuit C over a field $\mathbb{F}$ is a superposition of t set-multilinear depth three circuits over variables $X = \uplus_{i=1}^{t} Y_i$, if for every $i \in [t]$, C is a set-multilinear depth three circuit in Y_i variables over the field $\mathbb{F}(X \setminus Y_i)$. The sets $Y_1, ..., Y_t$ are called the *base sets* of C. Further, we restrict the Y_i to have singleton colors for every $i \in [t]$.

Although the notion of superposition makes sense even if Y_i's do not have singleton colors, we restrict to singletons as this model itself captures multilinear depth three circuits. We make the following observation for superposition of set-multilinear depth three circuits.

▶ **Observation 6.** *Given a circuit C which is a superposition of t set-multilinear circuits on unknown base sets $Y_1, Y_2, ..., Y_t$, finding t base sets $Y_1', Y_2', ..., Y_t'$ such that C is a superposition of t set-multilinear circuits on base sets $Y_1', Y_2', ..., Y_t'$ is NP-hard when $t > 2$.*

Due to space constraint the proof of Observation 6 is omitted. It can be found in an extended version of this paper [19]. We now state the main results of this paper.

▶ **Theorem 7** (Main Theorem 1)**.**

1. *There is an explicit family of $2n$-variate polynomials $\{g_n\}_{n\geq 1}$ over any field $\mathbb{F}$ such that the following hold: g_n is computable by a multilinear depth three circuit C over $\mathbb{F}$ with top fanin three and C is also a superposition of two set-multilinear depth three circuits. Any ROABP over $\mathbb{F}$ computing g_n has width $2^{\Omega(n)}$.*
2. *There is an explicit family of $3n$-variate polynomials $\{g_n\}_{n\geq 1}$ over any field $\mathbb{F}$ such that the following hold: g_n is computable by a multilinear depth three circuit C over $\mathbb{F}$ with top fanin two and C is also a superposition of three set-multilinear depth three circuits. Any ROABP over $\mathbb{F}$ computing f_n has width $2^{\Omega(n)}$.*

We prove Theorem 7 in Section 3. The tightness of the theorem is shown by this observation.

▶ **Observation 8.** *A polynomial computed by a multilinear $\Sigma\Pi\Sigma$ circuit with top fan-in two and at most two variables per linear polynomial can also be computed by an ROABP with constant width.*

In the interest of space the proof of Observation 8 is omitted, see [19]. Thus, it follows from Theorem 7 that if we increase either the top fan-in or the number of variables per linear polynomial from two to three in multilinear depth three circuits then there exist polynomials computed by such circuits such that ROABPs computing these polynomials have exponential width. We now state the "converse" of Theorem 7.

▶ **Theorem 9** (Main Theorem 2)**.** *Any multilinear depth three circuit (over any field) computing $\mathrm{IMM}_{n,d}$, the $(1,1)$-th entry of a product of d $n \times n$ symbolic matrices, has top fan-in $n^{\Omega(d)}$ for $n \geq 11$. (Note: This also implies a lower bound for determinant, see Corollary 38.)*

We prove Theorem 9 in Section 4. It is not hard to observe the following.

▶ **Observation 10.** *$\mathrm{IMM}_{n,d}$ can be computed by an n^2 width ROABP.*

The proof of Observation 10 is omitted, see [19]. Thus, Theorem 7, Theorem 9 and Observation 10 together imply a complete separation between multilinear depth three circuits and ROABPs. As a consequence of the proof of Theorem 9 we also get an exponential separation between multilinear depth three and multilinear depth four circuits (proof in Section 4).

▶ **Theorem 11.** *There is an explicit family of $O(n^2d)$-variate polynomials of degree d, $\{f_d\}_{d\geq 1}$, such that f_d is computable by a $O(n^2d)$-sized multilinear depth four circuit with top fan-in one (i.e. a $\Pi\Sigma\Pi$ circuit) and every multilinear depth three circuit computing f_d has top fan-in $n^{\Omega(d)}$ for $n \geq 11$.*

Observe that the hard polynomials used in Theorem 7 belong to a special class of multilinear depth three circuits – they are both superpositions of constantly many set-multilinear depth three circuits and simultaneously a sum of constantly many set-multilinear depth three circuits. Here is an example of a circuit from this class.

$$\begin{aligned} C(X,Y) = &(1+3x_1+5y_2)(4+x_2+y_1)+(9+6x_1+4y_2)(3+2x_2+5y_1) \\ &+(6+9x_1+4y_1)(2+5x_2+3y_2)+(3+6x_1+9y_1)(5+8x_2+2y_2) \end{aligned}$$

$C(X,Y)$ is a superposition of two set-multilinear depth three circuits with base sets $X = \{x_1\} \cup \{x_2\}$ and $Y = \{y_1\} \cup \{y_2\}$. But $C(X,Y)$ is also a sum of two set-multilinear depth three circuits with $\{x_1, y_2\}$, $\{x_2, y_1\}$ being the colors in the first set-multilinear depth three circuit (corresponding to the first two products) and $\{x_1, y_1\}$, $\{x_2, y_2\}$ the colors in the second set-multilinear depth three circuit (corresponding to the last two products). For such a subclass of multilinear depth three circuits, we give a quasi-polynomial time hitting set by extending the proof technique of [3].

▶ **Theorem 12.** *Let $\mathcal{C}_{n,m,l,s}$ be a subclass of multilinear depth three circuits computing n-variate polynomials such that every circuit in $\mathcal{C}_{n,m,l,s}$ is a superposition of at most m set-multilinear depth three circuits and simultaneously a sum of at most l set-multilinear depth three circuits, and has top fan-in bounded by s. There is a hitting-set generator for $\mathcal{C}_{n,m,l,s}$ running in $(ns)^{O(lm\log s)}$ time.*

When m and l are bounded by $\mathsf{poly}(\log ns)$, we get quasi-polynomial time hitting sets. The proof of Theorem 12, which extends the shift and rank concentration technique of [3], is omitted, see [19]. To our understanding, even if m and l are constants, [5]'s algorithm yields an $exp(\sqrt{n})$ hitting set complexity. Also, [13] has recently given a $(ndw)^{O(l2^l\log(ndw))}$ time hitting set generator for n-variate, individual (variable) degree d polynomials computed by sum of l ROABPs each of width less than w. Sum of l set-multilinear depth three circuits reduces to sum of l ROABPs as set-multilinear depth three circuits readily reduce to poly-sized ROABPs. But, observe the doubly exponential dependence on l in their result. On the contrary, in Theorem 12 the dependence is singly exponential in l. So, the hitting-set complexity remains quasi-polynomial for $l = (\log n)^{O(1)}$ whereas [13] gives an exponential time hitting-set generator when applied to the model in Theorem 12. However, it is also important to note that the model considered in Theorem 12 is somewhat weaker than the sum of ROABPs model in [13] because of the additional restriction that our model is also a superposition of m set-multilinear depth three circuits.

2 Preliminaries

Measures. We have used two complexity measures, namely evaluation dimension and a novel variant of the dimension of the space of partial derivatives, to prove Theorem 7 and 9

respectively. Evaluation dimension was first defined in [10][1]. Let X be a set of variables.

▶ **Definition 13** (Evaluation Dimension)**.** The evaluation dimension of a polynomial $g \in \mathbb{F}[X]$ with respect to a set $S \subseteq X$, denoted as $\text{Evaldim}_S(g)$, is defined as

$$\dim(\text{span}_{\mathbb{F}}\{g(X)|_{\forall x_j \in S \ x_j=\alpha_j} : \forall x_j \in S \ \alpha_j \in \mathbb{F}\}).$$

Evaluation dimension is a nearly equivalent variant of another measure, the *rank of the partial derivatives matrix*, defined and used earlier in [27, 26, 28, 29, 7] to prove lower bounds and separations for several multilinear models. These two measures are identical over fields of characteristic zero (or sufficiently large), but the former is well defined over any field.

The partial derivatives measure was introduced in [23, 25]. The following is a simple variant of this measure that is also inspired by the measure used in [27].

▶ **Definition 14** ("Skewed" partial derivatives)**.** Let $f \in \mathbb{F}[X, Y]$, where X and Y are disjoint sets of variables, and $\mathcal{Y}_k$ be the set of all monomials in Y variables of degree $k \in \mathbb{N}$. Define the measure $\mathsf{PD}_{\mathcal{Y}_k}(f)$ as

$$\dim\left(\text{span}_{\mathbb{F}}\left\{\left[\frac{\partial f(X,Y)}{\partial m}\right]_{\forall y \in Y \ y=0} : m \in \mathcal{Y}_k\right\}\right).$$

In proving Theorem 9, we apply the above measure with a significant difference (or *skew*) between the number of X and Y variables – it is this imbalance that plays a crucial role in the proof. It is easy to see that both the above measures obey subadditivity.

▶ **Lemma 15** (Subadditivity)**.**

1. *Let* $g_1, g_2 \in \mathbb{F}[X]$ *and* $S \subseteq X$*. Then*

$$\text{Evaldim}_S(g_1 + g_2) \leq \text{Evaldim}_S(g_1) + \text{Evaldim}_S(g_2).$$

2. *Let* $f_1, f_2 \in \mathbb{F}[X, Y]$*. Then* $\mathsf{PD}_{\mathcal{Y}_k}(f_1 + f_2) \leq \mathsf{PD}_{\mathcal{Y}_k}(f_1) + \mathsf{PD}_{\mathcal{Y}_k}(f_2)$.

Expander Graphs. A vital ingredient that helps us construct the hard polynomials in Theorem 7 is a family of explicit 3-regular expanders. We recall a few definitions from [16].

▶ **Definition 16** (Edge expansion and family of expanders)**.** Let $G = (V, E)$ be an undirected d-regular graph. For $S \subseteq V$, let $E(S, \overline{S})$ be the set of edges with one end incident on a vertex in S and the other incident on a vertex in $\overline{S} = V \backslash S$. The *edge expansion* of G denoted $h(G)$ is defined as:

$$h(G) = \min_{S:\ |S| \leq \frac{|V|}{2}} \frac{|E(S, \overline{S})|}{|S|}.$$

A sequence of d-regular graphs $\{G_i\}_{i \in \mathbb{N}}$ of size increasing with i is a *family of d-regular expanders* if there exists an $\epsilon > 0$ such that $h(G_i) > \epsilon$ for every i.

▶ **Definition 17** (Mildly explicit expanders)**.** Let $\mathcal{G} = \{G_i\}_{i \in \mathbb{N}}$ be a family of d-regular expanders such that the number of vertices in G_i is bounded by a polynomial in i. $\mathcal{G}$ is *mildly explicit* if there exists an algorithm that takes input i and constructs G_i in time polynomial in the size of G_i.

[1] They attributed the notion to Ramprasad Saptharishi.

A family of mildly explicit expanders. [16] mentions a family of mildly explicit 3-regular p-vertex expanders $\{G_p\}_{p \text{ prime}}$ such that for every graph G_p in the family: $h(G_p) > \frac{2+10^{-4}}{2}$. The vertices of G_p correspond to elements in $\mathbb{Z}_p$. A vertex x in G_p is connected to $x+1$, $x-1$ and to its inverse x^{-1} (operations are modulo p and inverse of 0 is defined as 0). We refer the reader to [16], section 11.1.2, for more details. Denote this family of 3-regular p-vertex expanders by $\mathcal{S}$.

Double Cover. The proof of Theorem 7 works with bipartite expanders. It is standard to transform a d-regular expander graph to a d-regular bipartite expander graph by taking its double cover.

▶ **Definition 18** (Double Cover)**.** The double cover of a graph $G = (V, E)$ is the bipartite graph $\tilde{G} = (L \uplus R, \tilde{E})$ where $|L| = |R| = |V|$. Corresponding to a vertex $u \in V$ we have two vertices $u_L \in L$ and $u_R \in R$. Edges (u_L, v_R) and $(u_R, v_L) \in \tilde{E}$ if and only if there is an edge $(u, v) \in E$.

▶ **Lemma 19.** *Let $\mathcal{S} = \{G_p\}_{p \text{ prime}}$ be the family of expanders as described above, and $\tilde{\mathcal{S}} = \{\tilde{G_p}\}_p$ the family of double covers of graphs in $\mathcal{S}$. Then $h(\tilde{G_p}) > \frac{2+10^{-4}}{2}$ for every p.* [In the interest of space the proof is omitted.]

Hitting-set generators. In Theorem 12, we give a quasi-polynomial time hitting-set generator for a subclass of multilinear depth three circuits.

▶ **Definition 20** (Hitting-set generators)**.** A hitting-set generator for a class of circuits $\mathcal{C}$ is a Turing machine $\mathcal{H}$ that takes $(1^n, 1^s)$ as input and outputs a set $\{a_1, \ldots, a_m\} \subseteq \mathbb{Z}^n$ such that for every circuit $C \in \mathcal{C}$ of size bounded by s and computing a nonzero n-variate polynomial over a field $\mathbb{F} \supset \mathbb{Z}$, there is an $i \in [m]$ for which $C(a_i) \neq 0$. Complexity of $\mathcal{H}$ is its running time. Hitting-set generators can be defined similarly over finite fields by considering field extensions.1

Technical Lemmas. The following lemmas are used in Theorem 7. Lemma 21 follows from Hall's marriage theorem [14]. The proofs of Lemmas 22 and 23 are omitted, see [19].

▶ **Lemma 21.** *A d-regular graph can be split into d edge disjoint perfect matchings.*

▶ **Lemma 22.** *Suppose $g_1(X), g_2(X), ..., g_m(X) \in \mathbb{F}[X]$ are $\mathbb{F}$-linearly independent polynomials in the variables $X = \{x_1, x_2, ..., x_n\}$ where $m = 2^n$. If $Y = \{y_1, y_2, ..., y_n\}$ are n variables different from X then (by identifying an $i \in [m]$ with an $S \subseteq [n]$),*

$$\text{Evaldim}_Y\Big(\sum_{S \subseteq [n]} y_S \cdot g_S(X)\Big) = m, \quad \text{where for } S \subseteq [n],\ y_S := \prod_{j \in S} y_j.$$

▶ **Lemma 23.** *If R is a width-k ROABP that computes $g(X)$ then for every $i \in [0, |X|]$ there exists a set $S \subseteq X$ of size i such that* $\text{Evaldim}_S(g) \leq k$.

3 Lower bounds for ROABP: Proof of Theorem 7

Proof of part 1

Construction of the polynomial family. We construct a family of $2n$-variate multilinear polynomials $\{g_n\}_{n \geq 1}$ from the explicit family of 3-regular expander graphs $\mathcal{S}$ (described

in Section 2). From an n-vertex graph $G = (V, E)$ in $\mathcal{S}$, construct a polynomial $g(X, Y)$ in variables $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ as follows: Let $\tilde{G} = (L \uplus R, \tilde{E})$ be the double cover of G. By Lemma 19, $h(\tilde{G}) > \frac{2+10^{-4}}{2}$. With every vertex in L (similarly, R) associate a unique variable in X (respectively, Y), thus vertices in L and R are identified with the X and Y variables respectively. An edge between x_i and y_j is associated with the linear polynomial $(1 + x_i + y_j)$. By Lemma 21, $\tilde{G}$ can be split into three edge disjoint perfect matchings. Polynomial $g(X, Y)$ is a sum of three product terms corresponding to the three edge disjoint perfect matchings of $\tilde{G}$; a product term is formed by taking product of the linear polynomials associated with the edges of the corresponding matching. It is easy to show the following claim. We leave the proof to the reader.

▶ **Claim 24.** *Polynomial g (constructed above) is computed by a multilinear depth three circuit C of size $\Theta(n)$ and top fan-in three, and C is a superposition of two set-multilinear depth three circuits.*

High evaluation dimension of $g(X, Y)$. It turns out that the evaluation dimension of $g(X, Y)$ with respect to any subset of variables of size $n/10$ is large.

▶ **Lemma 25.** *For any set $S \subseteq X \uplus Y$ of size $n/10$, $\text{Evaldim}_S(g) \geq 2^{\epsilon n}$ where $\epsilon > 0$ is a constant.*

Proof. Consider any subset S of $n/10$ variables from $X \uplus Y$. With respect to set S we can classify the linear polynomials in the product terms of $g(X, Y)$ into three types: *untouched* – if none of the two variables in the linear polynomial belong to S, *partially touched* – if exactly one of the variables in the linear polynomial belongs to S, and *completely touched* – if both variables belong to S. Call the three product terms of g – P_1, P_2 and P_3. Proof of the next claim is omitted, see [19].

▶ **Claim 26.** *There exists a set $X_0 \subseteq X$ of $\left(\frac{7n}{10} - 4\right)$ X-variables such that every $x \in X_0$ appears in an untouched linear polynomial in every P_i (for $i \in [3]$), and further if $(1 + x + y_{j_1}), (1 + x + y_{j_2})$ and $(1 + x + y_{j_3})$ are the linear polynomials occurring in P_1, P_2 and P_3 respectively then $y_{j_1} \neq y_{j_2} \neq y_{j_3}$.*

For $i \in [3]$, let B_i be the set of partially touched linear polynomials in term P_i.

▶ **Claim 27.** *There is an $i \in [3]$ such that $|B_i| \geq \epsilon n$ where $\epsilon = 0.01$.*

Proof. Let T be such that, for all $i \in [3]$, $|B_i| \leq T$. Recall that g has been constructed from the bipartite expander $\tilde{G}$, and vertices in $\tilde{G}$ identified with the variable set $X \uplus Y$. We denote the vertices in $\tilde{G}$ corresponding to the variables in S also by S, and denote the set of edges going out from S to $\overline{S} = L \uplus R \backslash S$ in $\tilde{G}$ by $\tilde{E}(S, \overline{S})$. Using the expansion property of $\tilde{G}$,

$$|\tilde{E}(S, \overline{S})| \geq h(\tilde{G}) \cdot |S| \geq \frac{2 + 10^{-4}}{2} \cdot \left(\frac{n}{10}\right).$$

Every edge in $\tilde{E}(S, \overline{S})$ corresponds to a partially touched linear polynomial. Since $\tilde{G}$ is 3-regular, at least $\frac{|\tilde{E}(S,\overline{S})|}{3}$ of the edges correspond to distinct partially touched linear polynomials. By assumption, the number of such partially touched linear polynomials is at most $3T$; and so $T \geq 0.01n$. ◀

The next claim completes the proof of Lemma 25.

▶ **Claim 28.** *If there exists an $i \in [3]$ such that $|B_i| \geq \epsilon n$ for $\epsilon > 0$, then $\text{Evaldim}_S(g) \geq 2^{\epsilon n}$.*

Proof. Without loss of generality, assume $|B_1| \geq \epsilon n$. Pick two variables, say x and x', from the set X_0 (as described in Claim 26). Let $(1 + x + y_{j_2})$ and $(1 + x' + y'_{j_3})$ be the linear polynomials appearing in P_2 and P_3 respectively. By substituting $x = -(1 + y_{j_2})$ and $x' = -(1 + y'_{j_3})$ in g, the terms P_2 and P_3 vanish but P_1 does not (by Claim 26). Let $\hat{g}$ be the polynomial g after the substitution. Polynomial $\hat{g}$ has only one product term $\hat{P}_1$ (i.e. P_1 under the substitution), and $\hat{P}_1$ has as many partially touched linear polynomials as P_1. At this point, it is not difficult to prove the following observation.

▶ **Observation 29.** $\text{Evaldim}_S(g) \geq \text{Evaldim}_S(\hat{g}) = \text{Evaldim}_S(\hat{P}_1) \geq 2^{\epsilon n}$.

This completes the proof of Claim 28. ◀

From Lemma 23 and 25 we conclude that any ROABP computing $g(X, Y)$ has width at least $2^{\epsilon n}$. ◀

Proof of part 2

Construction of the polynomial family. Similar to part 1, we construct a family of $3n$-variate multilinear polynomials $\{g_n\}_{n\geq 1}$ from the explicit family of 3-regular expanders $\mathcal{S}$ – but this time edges will be associated with variables and vertices with linear polynomials. From an n-vertex graph $G = (V, E)$ in $\mathcal{S}$, construct a polynomial $g(X, Y, Z)$ in variables $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$ and $Z = \{z_1, \ldots, z_n\}$ as follows: Let $\tilde{G} = (L \uplus R, \tilde{E})$ be the double cover of G, and as before $h(\tilde{G}) > \frac{2+10^{-4}}{2}$. Edges of $\tilde{G}$ can be split into three edge disjoint perfect matchings (by Lemma 21). Label the edges of the first perfect matching by distinct X-variables, the edges of the second matching by distinct Y-variables, and the edges of the third by distinct Z-variables. Vertices of $\tilde{G}$ now correspond to linear polynomials naturally – if the three edges incident on a vertex are labelled x_i, y_j and z_k then associate the linear polynomial $(1 + x_i + y_j + z_k)$ with the vertex. Let P_1 be the product of the linear polynomials associated with the vertices of L, and P_2 the product of linear polynomials associated with the vertices of R. Polynomial $g(X, Y, Z)$ is the sum of P_1 and P_2. The following claim is easy to show (just like Claim 24).

▶ **Claim 30.** *Polynomial g (constructed above) is computed by a multilinear depth three circuit C of size $\Theta(n)$ and top fan-in two, and C is a superposition of three set-multilinear depth three circuits.*

High evaluation dimension of $g(X, Y)$. The proof of the following lemma is similar to that of Lemma 25, differences arise only due to the 'dual' nature of g.

▶ **Lemma 31.** *For any $S \subseteq X \uplus Y \uplus Z$ of size $n/10$, $\text{Evaldim}_S(g) \geq 2^{\epsilon n}$ where $\epsilon > 0$ is a constant.*

Proof. Let S be any set of $\frac{n}{10}$ variables from $X \uplus Y \uplus Z$. The definitions of untouched, partially touched and completely touched linear polynomials are almost the same as in the proof of Lemma 25. The difference is we have three variables instead of two in a linear polynomial in g. So, a linear polynomial is partially touched if at most two of the three variables belong to S. For $i \in [2]$, let B_i be the set of partially touched linear polynomials and C_i the set of completely touched linear polynomials in product term P_i of g.

▶ **Claim 32.** *There is an $i \in [2]$ such that $|B_i| \geq \epsilon n$ where $\epsilon = 0.01$.*

Proof. Let T be such that, for all $i \in [2]$, $|B_i| \leq T$. The proof of the next observation is omitted, see [19].

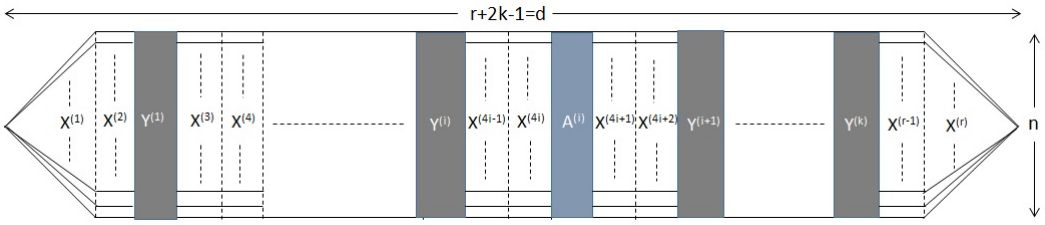


Figure 1 ABP $\mathcal{M}$.

▶ **Observation 33.** *$|C_1| + |C_2|$ is at least $\frac{n}{15} - \frac{8T}{3}$.*

Let C be the set of vertices in $\tilde{G}$ corresponding to the completely touched linear polynomials in both the product gates, thus $|C| = |C_1| + |C_2| \geq \frac{n}{15} - \frac{8T}{3}$. Each edge in $\tilde{E}(C, \overline{C})$ connects a vertex which corresponds to a completely touched linear polynomial to a vertex which corresponds to a partially touched linear polynomial. Using expansion of $\tilde{G}$,

$$|\tilde{E}(C, \overline{C})| \geq h(\tilde{G}) \cdot |C| \geq \frac{2 + 10^{-4}}{2} \cdot \left(\frac{n}{15} - \frac{8T}{3} \right).$$

Since edges in $\tilde{E}(C, \overline{C})$ are associated with variables in S, a vertex corresponding to a partially touched linear polynomial has at most two edges from $\tilde{E}(C, \overline{C})$ incident on it. Hence the number of vertices corresponding to partially touched linear polynomials is at least $\frac{\tilde{E}(C,\overline{C})}{2}$. But, by assumption, the number of such vertices is at most $2T$. Thus,

$$2T \geq \frac{|\tilde{E}(C, \overline{C})|}{2} \geq \frac{2 + 10^{-4}}{4} \cdot \left(\frac{n}{15} - \frac{8T}{3} \right) \quad \Rightarrow \quad T \geq 0.01n\,.$$

◀

The proof of the next claim is much like that of Claim 28 and is omitted.

▶ **Claim 34.** *If there exists an $i \in [2]$ such that $|B_i| \geq \epsilon n$ for $\epsilon > 0$, then* $\text{Evaldim}_S(g) \geq 2^{\epsilon n}$.

This completes the proof of Lemma 31. From Lemma 23 and 31 we conclude that any ROABP computing g has width at least $2^{\epsilon n}$. ◀

4 Lower bounds for multilinear depth three circuits

The proofs of Theorems 9 and 11 are inspired by a particular kind of *projection* of $\text{IMM}_{n,d}$ considered in [11]. We say a polynomial f is a *simple projection* of another polynomial g if f is obtained by simply setting some variables to field constants in g.

Proof of Theorem 9. The proof proceeds by constructing an ABP $\mathcal{M}$ of width n and with $d + 1$ layers of vertices such that (a) the polynomial computed by $\mathcal{M}$, say f, is a simple projection of $\text{IMM}_{n,d}$, and (b) any multilinear depth three circuit computing f has top fan-in $n^{\Omega(d)}$. Since an ABP can be viewed equivalently as a product of matrices, we will describe $\mathcal{M}$ using matrices. Figure 1 depicts the ABP $\mathcal{M}$.

Description of $\mathcal{M}$. The polynomial f, computed by $\mathcal{M}$, is defined over two disjoint sets of variables, X and Y. The Y variables are contained in k matrices, $\{Y^{(1)}, ..., Y^{(k)}\}$; the (u, v)-th entry in $Y^{(i)}$ is a formal variable $y^{(i)}_{u,v}$. There are $(k-1)$ matrices $\{A^{(1)}, ..., A^{(k-1)}\}$, such that all the entries in these matrices are ones. The X variables are contained in r matrices, $\{X^{(1)}, ..., X^{(r)}\}$. Matrices $X^{(1)}$ and $X^{(r)}$ are row and column vectors of size n respectively. The u-th entry in $X^{(1)}$ (similarly, $X^{(r)}$) is $x^{(1)}_u$ (respectively, $x^{(r)}_u$). All the remaining matrices $\{X^{(2)}, ..., X^{(r-1)}\}$ are diagonal matrices in the X variables, i.e. the (u, u)-th entry in $X^{(i)}$ is $x^{(i)}_u$ and all other entries are zero for $i \in [2, r-1]$. The matrices are placed as follows: Between two adjacent Y matrices, $Y^{(i)}$ and $Y^{(i+1)}$, we have five matrices ordered from left to right as $X^{(4i-1)}, X^{(4i)}, A^{(i)}, X^{(4i+1)}$ and $X^{(4i+2)}$ for every $i \in [1, k-1]$. Ordered from left to right, $X^{(1)}$ and $X^{(2)}$ are on the left of $Y^{(1)}$ and $X^{(r-1)}$ and $X^{(r)}$ are on the right of $Y^{(k)}$. Naturally, we have the following relation among k, r and d: $r = 4k$ and $d = r + 2k - 1$, i.e. $k = \frac{d+1}{6}$. Thus $|X| = nr = 4nk$ and $|Y| = n^2k$. This imbalance between the X and Y variables plays a vital role in the proof. As before, call the polynomial computed by this ABP $\mathcal{M}$ as $f(X, Y)$.

The following claim is easy to verify as f is a simple projection of $IMM_{n,d}$.

▶ **Claim 35.** *If* $\mathrm{IMM}_{n,d}$ *is computed by a multilinear depth three circuit having top fan-in* s *then* f *is also computed by a multilinear depth three circuit having top fan-in* s*.*

We show every multilinear depth three circuit computing f has top fan-in $n^{\Omega(d)}$ for $n \geq 11$.

Lower bounding $\mathsf{PD}_{\mathcal{Y}_k}(f)$. Let $\tilde{\mathcal{Y}}_k \subseteq \mathcal{Y}_k$ be the set of monomials formed by picking exactly one Y-variable from each of the matrices $Y^{(1)}, ..., Y^{(k)}$ and taking their product. Then, $|\tilde{\mathcal{Y}}_k| = n^{2k}$.

▶ **Claim 36.** $\mathsf{PD}_{\mathcal{Y}_k}(f(X, Y)) = |\tilde{\mathcal{Y}}_k| = n^{2k}$.

Proof. The derivative of f with respect to a monomial $m \in \mathcal{Y}_k$ is nonzero if and only if $m \in \tilde{\mathcal{Y}}_k$. Also, such a derivative $\frac{\partial f}{\partial m}$ is a multilinear degree-r monomial in X-variables. The derivatives of f with respect to two distinct monomials m and m' in $\tilde{\mathcal{Y}}_k$ give two distinct multilinear degree-r monomials in X-variables. Hence, $\mathsf{PD}_{\mathcal{Y}_k}(f) = |\tilde{\mathcal{Y}}_k|$. ◀

Upper bounding $\mathsf{PD}_{\mathcal{Y}_k}$ of a multilinear depth three circuit.

▶ **Lemma 37.** *Let* C *be a multilinear depth three circuit having top fan-in* s *computing a polynomial in* X *and* Y *variables. Then* $\mathsf{PD}_{\mathcal{Y}_k}(C) \leq s \cdot (k+1) \cdot \binom{|X|}{k}$ *if* $k \leq \frac{|X|}{2}$*.*

Proof. Let $C = \sum_{i=1}^{s} T_i$, where each T_i is a product of linear polynomials on disjoint sets of variables. From Lemma 15, $\mathsf{PD}_{\mathcal{Y}_k}(C) \leq s \cdot \max_{i \in [s]} \mathsf{PD}_{\mathcal{Y}_k}(T_i)$. We need to upper bound the dimension of the "skewed" partial derivatives of a term $T_i = T$ (say). Let $T = \prod_{j=1}^{q} l_j$, where l_j is a linear polynomial. Among the q linear polynomials at most $|X|$ of them contain the X variables. Without loss of generality, assume the linear polynomials $l_1, \ldots, l_p$ contain X-variables and the remaining $l_{p+1}, \ldots, l_q$ are X-free (here $p \leq |X|$). Let $Q = \prod_{j=p+1}^{q} l_j$. Then, $T = Q \cdot \prod_{j=1}^{p} l_j$. We take the derivative of T with respect to a monomial $m \in \mathcal{Y}_k$ and then substitute the Y variables to zero. Applying the product rule of differentiation and observing that the derivative of a linear polynomial with respect to a variable makes it a constant we have the following:

$$\left[\frac{\partial T}{\partial m}\right]_{Y=\overline{0}} = \sum_{\substack{S \subseteq [p] \\ |S| \leq k}} \alpha_S \prod_{j \in [p] \setminus S} [l_j]_{Y=\overline{0}}$$

where α_S's are constants from the field. Here m is a representative element of the set $\mathcal{Y}_k$. Hence every such derivative can be expressed as a linear combination of $\sum_{t=0}^{k}\binom{p}{t} \leq (k+1)\cdot\binom{|X|}{k}$ polynomials, where the last inequality is due to $k \leq \frac{|X|}{2}$ (if $t > p$ then $\binom{p}{t} \stackrel{\text{def}}{=} 0$). Therefore, $\mathsf{PD}_{\mathcal{Y}_k}(T) \leq (k+1)\cdot\binom{|X|}{k}$ and $\mathsf{PD}_{\mathcal{Y}_k}(C) \leq s\cdot(k+1)\cdot\binom{|X|}{k}$. ◀

It follows from Claim 36 and Lemma 37 that the top fan-in s of any multilinear depth three circuit computing $f(X,Y)$ is such that

$$s \geq \frac{n^{2k}}{(k+1)\cdot\binom{4nk}{k}} \geq \frac{n^{2k}}{(k+1)\cdot(4ne)^k} = n^{\Omega(d)},$$

as $n \geq 11$ and $k \leq |X|/2$ (required in Lemma 37). Claim 35 now completes the proof of Theorem 9. ◀

Theorem 9 implies the following corollary (already known due to [29]) as $IMM_{n,d}$ is a simple projection of Det_{nd}, the determinant of an $nd \times nd$ symbolic matrix [36].

▶ **Corollary 38** ([29]). *Any multilinear depth three circuit (over any field) computing* Det_d*, the determinant of a* $d \times d$ *symbolic matrix, has top fan-in* $2^{\Omega(d)}$*.*

Proof of Theorem 11. We now show that the polynomial $f(X,Y)$, computed by the ABP $\mathcal{M}$, can also be computed a multilinear depth four circuit of size $O(n^2d)$ and having top fan-in just one. ABP $\mathcal{M}$ has k matrices, $Y^{(1)},\ldots,Y^{(k)}$, containing the Y-variables. Associate with each matrix $Y^{(i)}$ four matrices containing the X-variables, two on the immediate left $X^{(4i-3)}$ and $X^{(4i-2)}$, and two on the immediate right $X^{(4i-1)}$ and $X^{(4i)}$. Every monomial in f is formed by picking exactly one variable from every matrix and taking their product. Once we pick $y^{(i)}_{u,v}$ from $Y^{(i)}$, this automatically fixes the variables picked from $X^{(4i-3)}, X^{(4i-2)}, X^{(4i-1)}$ and $X^{(4i)}$, as these are diagonal matrices. Moreover, any variable can be picked from $Y^{(i)}$ irrespective of which other Y-variables are picked from $Y^{(1)},\ldots,Y^{(i-1)},Y^{(i+1)},\ldots,Y^{(k)}$. This observation can be easily formalized to show that

$$f = \prod_{i=1}^{k}\sum_{u,v\in[n]} x_u^{(4i-3)}x_u^{(4i-2)}\cdot y^{(i)}_{u,v}\cdot x_v^{(4i-1)}x_v^{(4i)}.$$

The size of this multilinear ΠΣΠ circuit is $O(n^2k) = O(n^2d)$. ◀

Acknowledgements. VN and CS would like to thank Rohit Gurjar for helpful discussions on Observations 6 and 8. Thanks also to Nitin Saxena and Arpita Korwar for some early discussions at the onset of this work. NK and CS would like to thank Sébastien Tavenas for attending a presentation of this work and giving us some useful feedback. We also thank the anonymous reviewers for their helpful comments.

References

1 Manindra Agrawal. Proving lower bounds via pseudo-random generators. In *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International Conference, Hyderabad, India, December 15-18, 2005, Proceedings*, pages 92–105, 2005. doi:10.1007/11590156_6.

2 Manindra Agrawal, Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Hitting-sets for ROABP and sum of set-multilinear circuits. *SIAM J. Comput.*, 44(3):669–697, 2015. doi:10.1137/140975103.

3 Manindra Agrawal, Chandan Saha, and Nitin Saxena. Quasi-polynomial hitting-set for set-depth-Δ formulas. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 321–330, 2013.

4 Xi Chen, Neeraj Kayal, and Avi Wigderson. Partial Derivatives in Arithmetic Complexity and Beyond. *Foundations and Trends in Theoretical Computer Science*, 6(1-2):1–138, 2011.

5 Rafael Mendes de Oliveira, Amir Shpilka, and Ben Lee Volk. Subexponential size hitting sets for bounded depth multilinear formulas. In *30th Conference on Computational Complexity, CCC 2015, June 17-19, 2015, Portland, Oregon, USA*, pages 304–322, 2015. doi:10.4230/LIPIcs.CCC.2015.304.

6 Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978. doi:10.1016/0020-0190(78)90067-4.

7 Zeev Dvir, Guillaume Malod, Sylvain Perifel, and Amir Yehudayoff. Separating multilinear branching programs and formulas. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19-22, 2012*, pages 615–624, 2012. doi:10.1145/2213977.2214034.

8 Zeev Dvir, Amir Shpilka, and Amir Yehudayoff. Hardness-randomness tradeoffs for bounded depth arithmetic circuits. *SIAM J. Comput.*, 39(4):1279–1293, 2009. doi:10.1137/080735850.

9 Michael A. Forbes, Ramprasad Saptharishi, and Amir Shpilka. Hitting sets for multilinear read-once algebraic branching programs, in any order. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 – June 03, 2014*, pages 867–875, 2014.

10 Michael A. Forbes and Amir Shpilka. Quasipolynomial-time identity testing of noncommutative and read-once oblivious algebraic branching programs. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 243–252, 2013. doi:10.1109/FOCS.2013.34.

11 Hervé Fournier, Nutan Limaye, Guillaume Malod, and Srikanth Srinivasan. Lower bounds for depth 4 formulas computing iterated matrix multiplication. In *STOC*, pages 128–135, 2014.

12 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Arithmetic circuits: A chasm at depth three. In *Foundations of Computer Science (FOCS)*, pages 578–587, 2013.

13 Rohit Gurjar, Arpita Korwar, Nitin Saxena, and Thomas Thierauf. Deterministic identity testing for sum of read-once oblivious arithmetic branching programs. In *30th Conference on Computational Complexity, CCC 2015, June 17-19, 2015, Portland, Oregon, USA*, pages 323–346, 2015. doi:10.4230/LIPIcs.CCC.2015.323.

14 Philip Hall. On representatives of subsets. *J. London Math. Soc.*, 10(1):26–30, 1935.

15 Joos Heintz and Claus-Peter Schnorr. Testing polynomials which are easy to compute (extended abstract). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*, pages 262–272, 1980. doi:10.1145/800141.804674.

16 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.

17 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004. doi:10.1007/s00037-004-0182-6.

18 Erich Kaltofen. Factorization of Polynomials Given by Straight-Line Programs. In *Randomness and Computation*, pages 375–412. JAI Press, 1989.

19 Neeraj Kayal, Vineet Nair, and Chandan Saha. Separation between Read-once Oblivious Algebraic Branching Programs (ROABPs) and Multilinear Depth Three Circuits.

Electronic Colloquium on Computational Complexity (ECCC), 22:154, 2015. URL: http://eccc.hpi-web.de/report/2015/154.

20 Neeraj Kayal, Chandan Saha, and Sébastien Tavenas. On the size of homogeneous and of depth four formulas with low individual degree. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:181, 2015. URL: http://eccc.hpi-web.de/report/2015/181.

21 Neeraj Kayal and Ramprasad Saptharishi. A selection of lower bounds for arithmetic circuits. *Perspectives in Computational Complexity*, 2014.

22 Meena Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chicago J. Theor. Comput. Sci.*, 1997, 1997. URL: http://cjtcs.cs.uchicago.edu/articles/1997/5/contents.html.

23 Noam Nisan. Lower bounds for non-commutative computation (extended abstract). In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 410–418, 1991. doi:10.1145/103418.103462.

24 Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

25 Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity*, 6(3):217–234, 1997. Available at http://www.math.ias.edu/~avi/PUBLICATIONS/MYPAPERS/NW96/final.pdf.

26 Ran Raz. Separation of multilinear circuit and formula size. *Theory of Computing*, 2(1):121–135, 2006.

27 Ran Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. *J. ACM*, 56(2), 2009.

28 Ran Raz and Amir Yehudayoff. Balancing syntactically multilinear arithmetic circuits. *Computational Complexity*, 17(4):515–535, 2008.

29 Ran Raz and Amir Yehudayoff. Lower bounds and separations for constant depth multilinear circuits. *Computational Complexity*, 18(2):171–207, 2009.

30 Ramprasad Saptharishi. Recent progress on arithmetic circuit lower bounds. *Bulletin of the EATCS*, 114, 2014.

31 Nitin Saxena. Progress on polynomial identity testing. *Bulletin of the EATCS*, 99:49–79, 2009.

32 Nitin Saxena. Progress on polynomial identity testing – II. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:186, 2013.

33 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

34 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5:207–388, March 2010. doi:10.1561/0400000039.

35 W.T. Tutte. The Factorization of Linear Graphs. *J. London Math. Soc.*, 22:107–111, 1947.

36 L. G. Valiant. Completeness Classes in Algebra. In *STOC'79: Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 249–261, New York, NY, USA, 1979. ACM Press.

37 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation, EUROSAM'79, An International Symposiumon Symbolic and Algebraic Computation, Marseille, France, June 1979, Proceedings*, pages 216–226, 1979. doi:10.1007/3-540-09519-5_73.

Towards an Atlas of Computational Learning Theory

Timo Kötzing[1] **and Martin Schirneck**[2]

1 **Hasso Plattner Institute, Potsdam, Germany**
2 **Hasso Plattner Institute, Potsdam, Germany**

Abstract

A major part of our knowledge about Computational Learning stems from comparisons of the learning power of different learning criteria. These comparisons inform about trade-offs between learning restrictions and, more generally, learning settings; furthermore, they inform about what restrictions can be observed without losing learning power.

With this paper we propose that one main focus of future research in Computational Learning should be on a *structured approach* to determine the relations of different learning criteria. In particular, we propose that, for small sets of learning criteria, all pairwise relations should be determined; these relations can then be easily depicted as a *map*, a diagram detailing the relations. Once we have maps for many relevant sets of learning criteria, the collection of these maps is an *Atlas of Computational Learning Theory*, informing at a glance about the landscape of computational learning just as a geographical atlas informs about the earth.

In this paper we work toward this goal by providing three example maps, one pertaining to *partially set-driven* learning, and two pertaining to *strongly monotone* learning. These maps can serve as blueprints for future maps of similar base structure.

1998 ACM Subject Classification I.2.6 Learning

Keywords and phrases computational learning, language learning, partially set-driven learning, strongly monotone learning

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.47

1 Introduction

Computational Learning Theory, also called Inductive Inference, is a branch of (algorithmic) learning theory. This branch analyzes the problem of algorithmically learning a description for a formal language (a computably enumerable subset of the set of natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$) when presented successively the elements of that language. For example, a learner h might be presented more and more even numbers. After each new number, h outputs a description for a language as its conjecture. The learner h might decide to output a program for the set of all multiples of 4, as long as all numbers presented are divisible by 4. Later, when h sees an even number not divisible by 4, it might change this guess to a program for the set of all multiples of 2.

Many criteria for deciding whether a learner h is *successful* on a language L have been proposed in the literature. Gold, in his seminal paper [10], gave a first, simple learning criterion, **Txt**$\mathbf{G}$**Ex**-*learning*[1], where a learner is *successful* iff, on every *text* for L (listing of all and only the elements of L) it eventually stops changing its conjectures, and its final

[1] **Txt** stands for learning from a *text* of positive examples; **G** stands for Gold, who introduced this model, and is used to to indicate full-information learning; **Ex** stands for *explanatory*.

© Timo Kötzing and Martin Schirneck;
licensed under Creative Commons License CC-BY

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 47; pp. 47:1–47:13

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

conjecture is a correct description for the input sequence. Trivially, each single, describable language L has a suitable constant function as a **TxtGEx**-learner (this learner constantly outputs a description for L). Thus, we are interested in analyzing for which *classes of languages* $\mathcal{L}$ there is a *single learner* h learning *each* member of $\mathcal{L}$. This framework is also sometimes known as *language learning in the limit* and has been studied extensively, using a wide range of learning criteria similar to **TxtGEx**-learning (see, for example, the textbook [12]).

Recently, the notion of a learning criterion was formalized in [15] (see Section 2 for the formal notions relevant to this paper). This formalization defines learning criteria as a combination of several components. At the core of any learning criterion is the *interaction operator* which determines what information a learner can get about its target at any stage of the learning process. For example the learner might only see one new datum at a time, only remembering its very previous conjecture (*iterative learning*, **It**), or the learner might get the full information (**G**).

Second, there are many restrictions that can be imposed on learning. For example, **Ex** is the restriction that the learner converges to a single hypothesis, and this hypothesis correctly describes the target language. A relaxation to this is known as *behaviorally correct learning* (**Bc**), which does not require syntactic convergence, but still all but some finite initial hypothesis have to be correct. These restrictions can be combined with other restrictions, such as, for example, *strong monotonicity* (**SMon**), requiring that the languages described by the successive hypotheses to be monotonically non-decreasing. For any given interaction operator β and any learning restriction δ we will use **Txt**$\beta\delta$ to denote the learning criterion employing β as interaction operator and δ as learning restriction (in the setting of learning from text). Note that δ might be the combination, the conjunction, of several learning restrictions; we denote this conjunction of two restriction δ, δ' as $\delta\delta'$.

The main interaction operators from the literature (besides **It** and **G** there are also *set-driven* learning, **Sd**, and *partially set-driven learning*, **Psd**) exhibit a structure: for any two interaction operators β, β', we write $\beta \preceq \beta'$ iff every β-learner can be compiled into an equivalent β'-learner (see Section 2); intuitively, learners can always ignore additional information. In particular, we have the relations

$$\mathbf{Sd} \prec \mathbf{Psd} \prec \mathbf{G} \text{ and } \mathbf{It} \prec \mathbf{G}$$

and no other among these four operators. Note that, for any β, β' with $\beta \preceq \beta'$ and any restriction δ, any class **Txt**$\beta\delta$-learnable class is also **Txt**$\beta'\delta$-learnable: the existence of a β-learner implies the existence of an equivalent β'-learner (see Lemma 1). Note that the converse is not true: While there is no direct way to translate **Sd**-learners into equivalent **It**-learners, it is well known that every **TxtItEx**-learnable class is also **TxtSdEx**-learnable [14]. In other words, some comparisons of learning power of different learning criteria are *trivial* (following directly from very basic relations of the interaction operators), while others are *contingent* on the setting.

A similar observation holds for learning restrictions. These restrictions can be compared with $\subseteq$ (since we formally define them as sets of pairs for which the restriction holds); for example, we have $\mathbf{Ex} \subset \mathbf{Bc}$ in the sense that any sequence of conjectures which correctly **Ex**-identifies a target from a text T is also a sequence of conjectures which correctly **Bc**-identifies the same target from the same text T (but not vice versa). This again gives that some comparisons of learning power of different learning criteria are *trivial* (following directly from the $\subseteq$ relation on the restrictions), while others are *contingent* on the setting.

This observation holds for all parts of the learning criterion: a monotone change in a single component leads to a monotone change of the learning power of the learning criterion

(see Lemma 1 for the formal statement). In other words, for any set of learning criteria, the rough structure is visible from trivial inclusions, detailed structure requires specific analysis. Just as the exploration of a geographical landscape might begin with drawing a rough map of the area and then go explore the different parts in detail, a researcher comparing the learning power of different learning criteria might proceed by drawing the trivial inclusions among these criteria as a kind of "backbone" and then determine what further contingent relations hold. In this way the researcher draws a "map" of the collection of learning criteria.

For a full characterization it would be desirable to have a map of all (important) learning criteria. As this would require to determine the pairwise relations of several hundred learning criteria (using all possible combinations of different possible components of learning criteria), this is not easily feasible and more of a long term goal (just as the complete exploration of the earth is not feasible in one go). Furthermore, a large map of all criteria might not be very easy to understand; just as is done for the earth, giving a collection of maps, each giving details for some specific part, gives a much better idea. Thus, we want to propose that an important goal in Computational Learning Theory is *to give an atlas of insightful maps of learning criteria.*

The question which maps are insightful is of course the crucial and difficult part. The literature has given several examples; we proceed by giving three main kinds of maps that we propose to be insightful.

1.1 Partially Set-Driven Learning

A wealth of learning criteria can be derived from **TxtGEx**-learning by adding restrictions on the intermediate conjectures and how they should relate to each other and the data. For example, one could require that a conjecture which is consistent with the data must not be changed; this is known as *conservative* learning (**Conv**, [1]). Additionally to conservative learning, the following learning restrictions are considered frequently in the literature.

In *cautious* learning (**Caut**, [19]) the learner is not allowed to ever give a conjecture for a strict subset of a previously conjectured set. In *non-U-shaped* learning (**NU**, [2]) a learner may never *semantically* abandon a correct conjecture; in *strongly non-U-shaped* learning (**SNU**, [7]) not even syntactic changes are allowed after giving a correct conjecture. In *decisive* learning (**Dec**, [19]), a learner may never (semantically) return to a *semantically* abandoned conjecture; in *strongly decisive* learning (**SDec**, [16]) the learner may not even (semantically) return to *syntactically* abandoned conjectures. Finally, a number of monotonicity requirements are studied ([13, 24, 18]): in *strongly monotone* learning (**SMon**) the conjectured sets may only grow; in *monotone* learning (**Mon**) only incorrect data may be removed; and in *weakly monotone* learning (**WMon**) the conjectured set may only grow while it is consistent. A common property of these restrictions is *delayability* (see Definition 2).

Recently, [17] gave the map of **TxtG**δ**Ex** for all the learning restriction δ given above (plus **T**, denoting no restriction). The same paper also gave the map for **Sd** in place of **G**, while [11] gave the map for **It** in place of **G**. Thus, the only main interaction operator still missing is **Psd**, for which we give the map in this paper (depicted in Figure 1, see Section 3 for all relevant theorems). A solid black line indicates a trivial inclusion (the lower criterion is included in the higher); a dashed black line indicates an inclusion which is not trivial. A gray box around criteria indicates equality of (learning power of) these criteria. We will use this way of graphical representation for every map in this paper.

Thus, the solid black lines are the backbone of these learning criteria. It is interesting to note that the structure is exactly like for the interaction operator **G**, even though most of the proofs do not carry over, and **Psd**-learning is in general weaker than its **G**-variants (as,

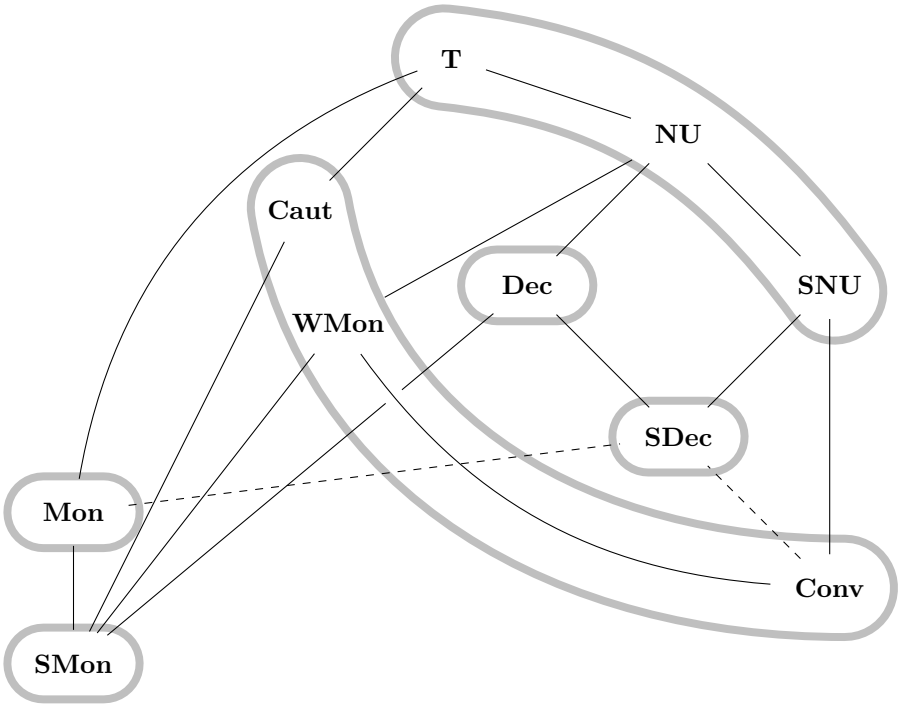


Figure 1 Relation of [**TxtPsd**δ**Ex**] for various learning restrictions δ. The backbone is given by the black solid lines (trivial inclusions bottom-to-top). The only previously known relations are the collapse of **T**, **NU** and **SNU**, as well as that **SMon** and **Mon** are each on their own.

for example, in the case of strongly monotone learning, see Section 1.2). Typically the proofs can use some ideas from the **G**-case, but need to add some specific ideas. In this context we developed some additional methods, such as a normal form for partially set-driven learners (see Lemma 3). Also, we showed that conservative, weakly monotone and cautious learning lie in between two other restrictions, which are then shown to be equivalent (see Theorem 6). This gives another interesting characterization of the important restriction of conservative learning.

1.2 Strongly Montone Learning and Interaction Operators

When comparing different interaction operators, it comes in handy to have a maximum (with respect to $\preceq$) and also a minimum. The maximum (for the four main operators) is **G**. As minimum we introduce *confluently iterative learning* (**CflIt**). It requires a learner to be just like an iterative learner, but with the added restriction of being confluent, that is, when a sequence of inputs is given in different order or quantity, the same output is produced by the learner. Intuitively, the learner has to be set-driven (thus **CflIt** $\preceq$ **Sd** and **CflIt** $\preceq$ **It**).

Consider now the restriction of strongly monotone learning (**SMon**). We can either require that the learner has to be strongly monotone only on relevant inputs (relevant to the current class of languages to be learned) or generally. This latter version is denoted τ(**SMon**) and written as a prefix to the learning criterion. Note that this requires the learner to be total (usually any element from $\mathcal{P}$, the set of all partial computable functions, can be used as a learner, as long as the learner produces output on relevant inputs). As totality of the learner can impede the learner in itself, it is interesting to compare the two different versions of a restrictions (on relevant or on all inputs) also with the version where the restriction is

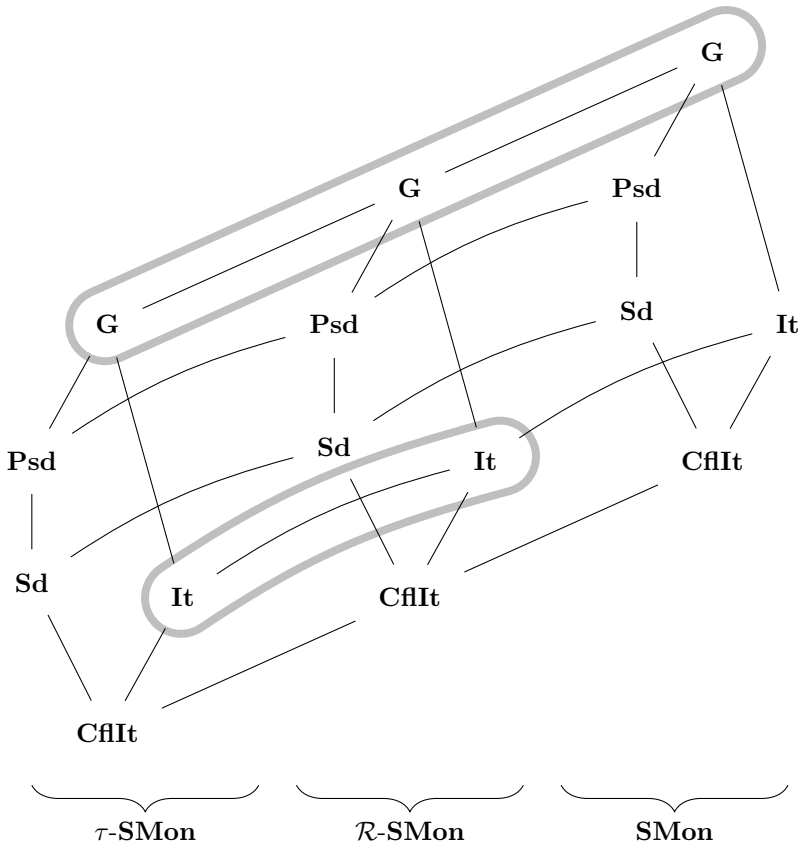


Figure 2 Relation of $[\tau(\mathbf{SMon})\mathbf{Txt}\beta\mathbf{Ex}]$, $[\mathcal{R}\mathbf{Txt}\beta\mathbf{SMonEx}]$ and $[\mathbf{Txt}\beta\mathbf{SMonEx}]$ for various interaction operators β. The backbone is given by the black solid lines (trivial inclusions bottom-to-top). The only previously known relation is $[\mathcal{R}\mathbf{TxtGSMonEx}] = [\mathbf{TxtGSMonEx}]$.

only on relevant inputs, but the learner is required to be total. We denote the restriction of totality by $\mathcal{R}$ (the symbol for the set of total computable functions) as a prefix.

Naturally we have that $\tau(\mathbf{SMon})\mathbf{Txt}\beta\mathbf{Ex}$ is trivially weaker in learning power than $\mathcal{R}\mathbf{Txt}\beta\mathbf{SMonEx}$, which is in turn trivially weaker in learning power than $\mathbf{Txt}\beta\mathbf{SMonEx}$. In this paper we give the map for these learning criteria (depicted in Figure 2, see Section 4 for all relevant theorems.

Again the solid black lines give the backbone. When replacing **SMon** with any other learning restriction, this backbone would stay the same. Noteworthy is the collapse of the three different criteria involving **G**, which has the same underlying idea as the collapse of the two criteria featuring **It**. As we can see, any strongly monotone **G**-learner can be assumed to be not only total but also strongly monotone on arbitrary inputs; this does not hold for the other interaction operators: the proof for **G** exploits first a standard delaying trick to make sure that the learner is total, and then make use of the knowledge of prior hypotheses to make sure that learner proceeds strongly monotonically.

Furthermore we also give the map for the criteria for the corresponding learning criteria with **Bc** in place of **Ex**. This map is trivial, as all learning criteria $\tau(\mathbf{SMon})\mathbf{Txt}\beta\mathbf{Bc}$, $\mathcal{R}\mathbf{Txt}\beta\mathbf{SMonBc}$ and $\mathbf{Txt}\beta\mathbf{SMonBc}$ for the five different β are equivalent in learning power.

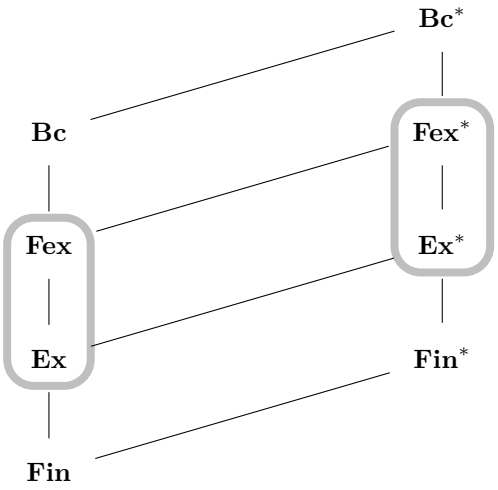


Figure 3 Relation of [**TxtGSMon**δ] for various δ. The backbone is given by the black solid lines (trivial inclusions bottom-to-top).

1.3 Strongly Monotone Learning and Convergence Criteria

In Section 1.1 we saw many different learning restrictions which are typically combined with **Ex**. As an alternative to **Ex** we saw **Bc**. Another alternative is **Fex**, allowing any *finite* number of limit conjectures, naturally in between **Ex** and **Bc**. Even more restrictive than **Ex** is **Fin**, allowing only one hypothesis different from ? overall (this is *finite learning*). Each of these four convergence restrictions can be relaxed by counting hypotheses for finite variants of the target also as correct hypotheses; this is denoted by an asterisk as suffix (e.g., **Ex***).

We give the map for the learning criteria **TxtGSMon**δ for the eight different choices of convergence criteria (depicted in Figure 3, see Section 5 for all relevant theorems).
As you can see, the **Ex**-variants and the corresponding **Fex**-variants collapse in the case of strongly monotone full-information learning. All other learning criteria stay separated, and no further inclusions exist.

1.4 Conclusions

We have given three different maps considering very different kinds of learning criteria. We believe that such maps increase our general understanding of learning and give a better picture of how different criteria relate. While establishing these maps we typically get structural insights, which can easily be explained with reference to the map. Most crucially, since the main result of the research is a map, it is very easy to communicate the results; especially other researchers who know the general backbone will be able to take in the results effortlessly. Furthermore, this approach points out gaps in our knowledge about learning criteria and thus focuses research efforts: maps which leave open problems are typically of much lesser value, since they only provide a partial picture.

We believe that giving similar maps will drive future research in an important direction: the development of an *Atlas of Computational Learning Theory*.

Next, in Section 2, we give a formal definition of learning criteria. Sections 3 to 5 conclude the paper by giving the formal theorems establishing the maps presented above. All proofs are omitted due to space restrictions; the main contribution of this paper lies elsewhere.

2 Learning Criteria

In this section we formally introduce our setting of learning in the limit and associated learning criteria. We follow the system in [15] in defining learning criteria. For background on computability, see [21]. In particular, we let W_e denote the recursively enumerable (r.e.) set enumerated by the program with (coded) number e. Thus, we can interpret a natural numbers e as hypothesis for the set W_e. We denote the set of all r.e. sets by $\mathcal{E}$.

A *learner* is a partial computable function $h \in \mathcal{P}$. We allow learners to output ? to denote that no conjecture is made yet. A *language* is an r.e. set $L \in \mathcal{E}$ of natural numbers. Any total function $T\colon \mathbb{N} \to \mathbb{N} \cup \{\#\}$ is a *text*, the collection of all texts is **Txt**. For any text (or sequence) T, we let $\mathrm{content}(T) = \mathrm{range}(T)\backslash\{\#\}$. For any given language L, a *text for* L is a text T such that $\mathrm{content}(T) = L$. The set of all texts for some language L is denoted $\mathbf{Txt}(L)$. For a given text $T \in \mathbf{Txt}$ and any n, we use $T[n]$ to denote the sequence $(T(0), \ldots, T(n-1))$ (the empty sequence λ when $n = 0$). Initial parts of this kind is what learners usually get as information.

An *interaction operator* is an operator β taking as arguments a function h (the learner) and a text T, and outputs a (possibly partial) function p. We call p the *learning sequence* (or *sequence of hypotheses*) of h given T. We define the interaction operators **G** (*Gold-style* or *full-information learning* [10]), **Psd** (*partially set-driven learning*, [22]), **Sd** (*set-driven learning*, [23]) and **It** (*iterative learning*, [23]) as follows. For all $h \in \mathcal{P}$, texts T and all i,

$$\begin{aligned}
\mathbf{G}(h,T)(i) &= h(T[i]);\\
\mathbf{Psd}(h,T)(i) &= h(\mathrm{content}(T[i]),i);\\
\mathbf{Sd}(h,T)(i) &= h(\mathrm{content}(T[i]));\\
\mathbf{It}(h,T)(i) &= \begin{cases} h(\lambda), & \text{if } i=0;\\ h(\mathbf{It}(h,T)(i-1),T(i-1)), & \text{otherwise.}\end{cases}
\end{aligned}$$

In set-driven learning, the learner has access to the set of all previous data, but not to the full sequence as in **G**-learning. In partially set-driven learning, the learner has the set of data and the current iteration number. **Psd**-learning is sometimes also called *rearrangement-independent learning* [4]. In iterative learning, the learner can access its last hypothesis as well as the most recent input data. Hereby, $h(\lambda)$ denotes the initial hypothesis of learner h. For two interaction operators β, β', we say *β-learners can be translated into β'-learners*, written $\beta \preceq \beta'$, if, for every learner h, there is some learner h' such that, for arbitrary texts T, the resulting sequence of hypotheses of h working on T is the same as that of h', i.e. $\forall T \in \mathbf{Txt}\colon \beta(h,T) = \beta'(h',T)$. For example, an **Sd**-learner can be translated into an **Psd**-learner by simply ignoring the additional information of the number of the current iteration. Clearly, all learners investigated in this paper can be translated into **G**-learners. For any β-learner h such that $\beta \preceq \mathbf{G}$, we let h^* (the *starred learner*) denote the **G**-learner to simulate h. A learner h is said to be *confluently iterative* just in case it is both set-driven and iterative. That is, its starred learner h^* satisfies the following two conditions. For any two finite sequences σ, τ and natural numbers x, we have $\mathrm{content}(\sigma) = \mathrm{content}(\tau) \Rightarrow h^*(\sigma) = h^*(\tau)$ and $h^*(\sigma) = h^*(\tau) \Rightarrow h^*(\sigma \diamond x) = h^*(\tau \diamond x)$. The interaction operator associated with confluent iterativeness is denoted **CflIt**, it is a $\preceq$-lower bound for both **Sd** and **It**.

Successful learning requires the learner to observe certain restrictions, for example convergence to a correct index. A *learning restriction* is a predicate δ on a learning sequence and a text. We give the important example of explanatory learning (**Ex**, [10]) defined such

that, for all sequences of hypotheses p and all texts T,

$$\mathbf{Ex}(p,T) \Leftrightarrow p \text{ total } \wedge [\exists n_0 \colon (\forall n \geq n_0 : p(n) = p(n_0)) \wedge W_{p(n_0)} = \text{content}(T)].$$

There are several other success criteria under investigation in this work, most notably in Section 5. We always require successful learning sequences p to be total, as in **Ex**-learning. These success criteria, as well as other learning restrictions discussed in Section 1, are given as follows.

$$\begin{aligned}
\mathbf{Ex}^*(p,T) &\Leftrightarrow \exists n_0 \colon (\forall n \geq n_0 : p(n) = p(n_0)) \wedge W_{p(n_0)} =^* \text{content}(T);\\
\mathbf{Fex}(p,T) &\Leftrightarrow \exists D\ \exists n_0 \colon (\forall n \geq n_0 \colon p(n) \in D) \wedge (\forall e \in D \colon W_e = \text{content}(T));\\
\mathbf{Fex}^*(p,T) &\Leftrightarrow \exists D\ \exists n_0 \colon (\forall n \geq n_0 \colon p(n) \in D) \wedge (\forall e \in D \colon W_e =^* \text{content}(T));\\
\mathbf{Bc}(p,T) &\Leftrightarrow \exists n_0\ \forall n \geq n_0 \colon W_{p(n)} = \text{content}(T);\\
\mathbf{Bc}^*(p,T) &\Leftrightarrow \exists n_0\ \forall n \geq n_0 \colon W_{p(n)} =^* \text{content}(T);\\
\mathbf{Fin}(p,T) &\Leftrightarrow \exists n_0 \colon (\forall n < n_0 \colon p(n) = ?) \wedge W_{p(n_0)} = \text{content}(T);\\
\mathbf{Fin}^*(p,T) &\Leftrightarrow \exists n_0 \colon (\forall n < n_0 \colon p(n) = ?) \wedge W_{p(n_0)} =^* \text{content}(T).\\
\mathbf{Conv}(p,T) &\Leftrightarrow \forall i : \text{content}(T[i+1]) \subseteq W_{p(i)} \Rightarrow p(i) = p(i+1);\\
\mathbf{Caut}(p,T) &\Leftrightarrow \forall i,j : W_{p(i)} \subset W_{p(j)} \Rightarrow i < j;\\
\mathbf{NU}(p,T) &\Leftrightarrow \forall i,j,k : i \leq j \leq k \ \wedge\ W_{p(i)} = W_{p(k)} = \text{content}(T) \Rightarrow W_{p(j)} = W_{p(i)};\\
\mathbf{Dec}(p,T) &\Leftrightarrow \forall i,j,k : i \leq j \leq k \ \wedge\ W_{p(i)} = W_{p(k)} \Rightarrow W_{p(j)} = W_{p(i)};\\
\mathbf{SNU}(p,T) &\Leftrightarrow \forall i,j,k : i \leq j \leq k \ \wedge\ W_{p(i)} = W_{p(k)} = \text{content}(T) \Rightarrow p(j) = p(i);\\
\mathbf{SDec}(p,T) &\Leftrightarrow \forall i,j,k : i \leq j \leq k \ \wedge\ W_{p(i)} = W_{p(k)} \Rightarrow p(j) = p(i);\\
\mathbf{SMon}(p,T) &\Leftrightarrow \forall i,j : i < j \Rightarrow W_{p(i)} \subseteq W_{p(j)};\\
\mathbf{Mon}(p,T) &\Leftrightarrow \forall i,j : i < j \Rightarrow W_{p(i)} \cap \text{content}(T) \subseteq W_{p(j)} \cap \text{content}(T);\\
\mathbf{WMon}(p,T) &\Leftrightarrow \forall i,j : i < j \wedge \text{content}(T[j]) \subseteq W_{p(i)} \Rightarrow W_{p(i)} \subseteq W_{p(j)}.
\end{aligned}$$

We combine any two learning restrictions δ and δ' by intersecting them; we denote this by juxtaposition. With $\mathbf{T}$ we denote the restriction which is always true (no restriction).

Now a *learning criterion* is a tuple $(\alpha, \mathcal{C}, \beta, \delta)$, where $\mathcal{C}$ is a set of learners (the admissible learners; typically $\mathcal{P}$ or $\mathcal{R}$), β is an interaction operator and α, δ are learning restrictions; we write $\tau(\alpha)\mathcal{C}\mathbf{Txt}\beta\delta$ to denote the learning criterion, omitting $\mathcal{C}$ in case of $\mathcal{C} = \mathcal{P}$ and the restriction in case it equals $\mathbf{T}$. We say that a learner $h \in \mathcal{C}$ $\tau(\alpha)\mathcal{C}\mathbf{Txt}\beta\delta$*-learns* a language L iff, on *arbitrary* texts $T \in \mathbf{Txt}$, $\alpha(\beta(h,T),T)$ and, for all texts $T \in \mathbf{Txt}(L)$, $\delta(\beta(h,T),T)$. The set of languages $\tau(\alpha)\mathcal{C}\mathbf{Txt}\beta\delta$-learned by $h \in \mathcal{C}$ is denoted by $\tau(\alpha)\mathcal{C}\mathbf{Txt}\beta\delta(h)$. We write $[\tau(\alpha)\mathcal{C}\mathbf{Txt}\beta\delta]$ to denote the set of all $\tau(\alpha)\mathcal{C}\mathbf{Txt}\beta\delta$-learnable classes.

Throughout this paper we are mostly concerned in showing separations between learning criteria. The reason is, that most of the inclusions are trivial in the sense that almost all of them follow from the next lemma. A formal proof can be found in [6].

▶ **Lemma 1.** *Let $\alpha \subseteq \alpha', \delta \subseteq \delta'$ be learning restrictions, $\mathcal{C} \subseteq \mathcal{C}'$ classes of admissible learners and $\beta \preceq \beta'$ two interaction operators. Then we have $[\tau(\alpha)\mathcal{C}\mathbf{Txt}\beta\delta] \subseteq [\tau(\alpha')\mathcal{C}'\mathbf{Txt}\beta'\delta']$.*

3 Delayable Partially Set-driven Language Learning

In this section we will investigate delayable **Psd**-learning. First, we establish a normal form for **Psd**-learners. Consider pairs (D,t) and (D',t') consisting of finite sets D, D' and numbers t, t', we write $(D,t) \to (D',t')$ just in case $t \leq t'$ and there is a text T such that

content$(T[t]) = D$ and content$(T[t']) = D'$. Note that $(D,t) \to (D',t')$ implies $D \subseteq D'$. Let h be some learner and L a language. We call a pair (D,t), with $D \subseteq L$ and $t \geq |D|$, such that $W_{h(D,t)} = L$ and for all (D',t') such that $D' \subseteq L$ and $(D,t) \to (D',t')$, $h(D,t) = h(D',t')$, a *locking information* for learner h on L. If we use interaction operator **G** instead a locking information is commonly referred to as a *locking sequence*. It is well known that, if h **Ex**-learns L, then *every* such pair can be extended to some locking information [4, 6]. Moreover, we call h *strongly locking* if, for each language $L \in \mathbf{TxtPsdEx}(h)$ and every text $T \in \mathbf{Txt}(L)$ for L, there is an n such that $(\text{content}(T[n]), n)$ serves as a locking information for h on L. We call learner h *order-independent* if, for all languages $L \in \mathbf{TxtPsdEx}(h)$ and any two texts $T, T' \in \mathbf{Txt}(L)$ for L, we have $\lim_{n\to\infty} \mathbf{Psd}(h,T)(n) = \lim_{n\to\infty} \mathbf{Psd}(h,T')(n)$. That is, h's final hypothesis only depends on the language L, not on the particular order in which its elements are presented in the text. We now turn to the notion of delayable learning.

▶ **Definition 2.** Let $\vec{R}$ be the set of all unbounded non-decreasing functions $r\colon \mathbb{N} \to \mathbb{N}$, i.e. for all m we have $\forall^\infty n : r(n) \geq m$. We call a learning restriction δ *delayable* if, for all texts T' and T with content(T') = content(T), all infinite sequences p and all $r \in \vec{R}$, if $(p,T') \in \delta$ and $\forall n : \text{content}(T'[r(n)]) \subseteq \text{content}(T[n])$, then $(p \circ r, T) \in \delta$.

Intuitively, as long as the learner has *at least as much* data as was used for a given conjecture, then this conjecture is permissible. The intersection of two delayable learning criteria is again delayable. *All* learning restrictions considered in this paper, including success criteria and **T**, are delayable. We now get a normal form for delayable **Psd**-learner.

▶ **Lemma 3.** *Let restrictions α, δ be delayable and $\mathcal{C} \in \{\mathcal{P}, \mathcal{R}\}$. Then $\tau(\alpha)\mathcal{C}\mathbf{TxtPsd}\delta$ allows for strongly locking and order-independent learning (simultaneously).*

We now proceed to prove the connections shown in Figure 1. The following theorem translates what is known about the respective criteria of delayable **G**-learning into the **Psd**-setting. The first part has already been proven by Case and Kötzing [6]. Furthermore, the separations in the second part (in the full-information case) are due to Baliga et al. [2], Kötzing and Palenta [17] as well as Osherson et al. [20]. For a collection of known separations see [17].

▶ **Theorem 4.** *Among the investigated criteria non-U-shapedness and strong non-U-shapedness are the only ones equally powerful to unrestricted* **Psd***-learning when paired with success criterion* **Ex**. *This is, we have:*

1. $[\mathbf{TxtPsdSNUEx}] = [\mathbf{TxtPsdNUEx}] = [\mathbf{TxtPsdEx}]$.
2. *For* $\delta \in \{\mathbf{Conv}, \mathbf{Caut}, \mathbf{Dec}, \mathbf{SDec}, \mathbf{Mon}, \mathbf{WMon}, \mathbf{SMon}\}$, $[\mathbf{TxtPsd}\delta\mathbf{Ex}] \subset [\mathbf{TxtPsdEx}]$.

The separation of decisive and strongly decisive learning follows just as for **G**, see [17].

▶ **Theorem 5.** *We have* $[\mathbf{TxtPsdSDecEx}] \subset [\mathbf{TxtPsdDecEx}]$.

For our work on conservative, weakly monotone and cautious learning, we first give two more learning restrictions. One of them, *witness-based* learning (**Wb**), is more restrictive than all three of conservative, weakly monotone and cautious learning; the other, *target-cautious learning* ($\mathbf{Caut_{Tar}}$), is less restrictive. That way those three learning restrictions are "sandwiched" between witness-based and target-cautious learning. We will then show that witness-based and target-cautious learning have equal learning power in the setting of partially set-driven learning, showing all three sandwiched restrictions to be equivalent.

We start by giving the definition of witness-based learning.

$$\mathbf{Wb}(p,T) \Leftrightarrow \forall i,j : (\exists k : i < k \leq j \wedge p(i) \neq p(k)) \Rightarrow (\text{content}(T[j]) \cap W_{p(j)}) \setminus W_{p(i)} \neq \emptyset.$$

Intuitively, any mind change has to be *witnessed* by some datum which was not included before, but is included now; this witness *justifies* the mind change.

Target-cautious learning was introduced in [17] to study the notion of cautious learning in more detail.

$$\mathbf{Caut_{Tar}}(p,T) \Leftrightarrow \forall i : \neg(\text{content}(T) \subset W_{p(i)})$$

Intuitively, the learner may never conjecture a superset of the actual target language; in other words, it is required to be cautious, but only with respect to the target.

We now get to the theorem establishing the equivalence of the conservative, weakly monotone and cautious learning.

▶ **Theorem 6.** *The following learning criteria are equivalent:* $\mathbf{TxtPsdCaut_{Tar}Ex}$*;* **TxtPsdCautEx***;* **TxtPsdConvEx***;* **TxtPsdWMonEx***;* **TxtPsdWbEx***.*

The following theorem solely reformulates a result well-known in **G**-learning in terms of **Psd**-learning. The first part uses a standard proof, see [12] for example. The second part has already been shown (for **G**-learning) by Kötzing and Palenta in [17], in turn based on a technique presented in [20].

▶ **Theorem 7.** *Monotone and weakly monotone* **Psd***-learning is incomparable. In particular, we have*

1. [**TxtSdWMonEx**]\[**TxtPsdMonEx**] $\neq \emptyset$*;*
2. [**TxtPsdMonSDecEx**]\[**TxtPsdWMonEx**] $\neq \emptyset$*;*

▶ **Corollary 8.** *For each learning restriction* $\delta \in \{\mathbf{Caut}, \mathbf{Caut_{Tar}}, \mathbf{Conv}, \mathbf{Wb}, \mathbf{WMon}\}$*, we have* [**TxtPsd**$\delta$**Ex**] $\subset$ [**TxtPsdSDecEx**]*.*

The upcoming lemma is based on an theorem due to Baliga et al. [2] stating that concept classes containing the set $\mathbb{N}$ of all natural numbers, if they are inferable at all, are decisively learnable. Kötzing and Palenta extended it to comprise strong decisiveness [17]. We show that the result still holds when restricted to **Psd**-learnable classes.

▶ **Lemma 9.** *Let* $\mathcal{L}$ *be a class of languages with* $\mathbb{N} \in \mathcal{L}$*. If class* $\mathcal{L}$ *is identifiable by a* **Psd***-learner at all, then it can in fact be so learned strongly decisive.*

We can now use Lemma 9 to show that every monotonically learnable class can be learned strongly decisively.

▶ **Theorem 10.** *Any monotonically* **Psd***-learnable class of languages can be so learned strongly decisively, while the converse does not hold. Thus, we have* [**TxtPsdMonEx**] $\subset$ [**TxtPsdSDecEx**]*.*

4 Strongly Monotone Language Learning

In this section we prove Figure 2 (in Section 4.1) as well as the equivalence of all corresponding learning criteria with **Bc** in place of **Ex** (in Section 4.2).

4.1 Ex-Learning

In this section we discuss strongly monotone language learning in the context of explanatory (**Ex**) convergence. In particular, we will prove the diagram shown in Figure 2. For any delayable learning restriction δ we can, without loss of generality, assume every **G**-learner with respect to δ to be *total* [17]. In particular, this holds for $\delta = \mathbf{SMonEx}$ (and **SMonBc** as well). Thus, we get a first connection between the investigated criteria, namely $[\mathcal{R}\mathbf{TxtGSMonEx}] = [\mathbf{TxtGSMonEx}]$. Interestingly enough, at least for strongly monotone explanatory learning, **G** is the *only* interaction operator for which this relation holds, as shown in the next theorem.

▶ **Theorem 11.** *For each interaction operator* $\beta \in \{\mathbf{CflIt}, \mathbf{It}, \mathbf{Sd}, \mathbf{Psd}\}$*, we have* $[\mathbf{TxtCflItSMonEx}] \setminus [\mathcal{R}\mathbf{Txt}\beta\mathbf{SMonEx}] \neq \emptyset$.

Recall that we use $\tau(\alpha)$ to denote that a learner observes learning restriction α on *arbitrary* texts, even on those for languages it cannot identify. In the discussion of $\tau(\mathbf{SMon})$-learners we can distinguish two major groups: On one hand, we have **G**- as well as **It**-learners which can be transposed into τ-learners without loss of learning power. On the other hand, there is the group of **Psd**-, **Sd**- and **CflIt**-learners for which their total variants are strictly more powerful than their globally strongly monotone matches. The main property discriminating these two groups is whether the learner has access to its previous conjecture.

▶ **Theorem 12.** *Total full-information and iterative* **SMon***-learning can w.l.o.g. be done by a learner being strongly monotone on arbitrary texts. Thus, we get the following equalities.*

1. $[\tau(\mathbf{SMon})\mathbf{TxtGEx}] = [\mathcal{R}\mathbf{TxtGSMonEx}] = [\mathbf{TxtGSMonEx}]$;
2. $[\tau(\mathbf{SMon})\mathbf{TxtItEx}] = [\mathcal{R}\mathbf{TxtItSMonEx}]$.

The target classes identifiable by τ-learners drawn from the aforementioned second group (interaction operators **Psd**, **Sd** and **CflIt**) share an interesting common trait in terms of recursiveness. It is stated in the following lemma. The technique used in its proof resembles that of a well-known proposition regarding globally *consistent* learning, cf. [1] and [12, Proposition 5.6].

▶ **Lemma 13.** *If a class* $\mathcal{L}$ *is identifiable by a* **Psd***-learner being globally strongly monotone and if* $\mathcal{L}$ *comprises at least all singleton sets, then* $\mathcal{L}$ *is a collection of recursive languages.*

▶ **Theorem 14.** *There is a class of languages identifiable by a total* **CflIt***-learner which cannot be learned by a globally strongly monotone* **Psd***-learner. That is, for all interaction operators* $\beta \in \{\mathbf{CflIt}, \mathbf{Sd}, \mathbf{Psd}\}$*, we have* $[\mathcal{R}\mathbf{TxtCflItSMonEx}] \setminus [\tau(\mathbf{SMon})\mathbf{Txt}\beta\mathbf{Ex}] \neq \emptyset$.

▶ **Theorem 15.** *There is a class of languages identifiable by a total strongly monotone iterative learner which cannot be so learned partially set-driven, this is,* $[\mathcal{R}\mathbf{TxtItSMonEx}] \setminus [\mathbf{TxtPsdSMonEx}] \neq \emptyset$.

▶ **Corollary 16.** **Psd***-learning with respect to* **SMon** *is strictly less powerful than its full-information counterpart. Hence, we have* $[\mathbf{TxtPsdSMonEx}] \subset [\mathbf{TxtGSMonEx}]$.

This corollary stands in sharp contrast to the abilities of partially set-driven functions in *unrestricted* language learning. A famous result due to Fulk [9] states that *any* class of languages, which can be learned at all, can be inferred by a total **Psd**-learner. To our knowledge, **SMon** is the first learning restriction in literature for which no equivalent of Fulk's Theorem holds.

▶ **Theorem 17.** *We have* $[\tau(\mathbf{SMon})\mathbf{TxtSdEx}] \setminus [\mathbf{TxtItSMonEx}] \neq \emptyset$*, i.e. there is a class of languages identifiable by a globally strongly monotone* **Sd***-learner which cannot be identified by any* **It***-learner.*

The last question to be covered in this section is that of the relation among the learners *within* the second group, namely, that between **Psd**- and **Sd**-learners.

▶ **Theorem 18.** *For all variants,* **Psd***-learners are strictly more powerful than their set-driven analogues. Particularly, we have* $[\tau(\mathbf{SMon})\mathbf{TxtPsdEx}]\setminus[\mathbf{TxtSdSMonEx}] \neq \emptyset$.

4.2 Bc-Learning

In this section we turn the discussion to behaviorally correct (**Bc**) language learning. It becomes apparent that all criteria in this setting possess the same learning power. We establish this in one step by showing that full-information **Bc**-learning with respect to **SMon** can be done confluently iteratively being globally strongly monotone. We conclude this section by proving that strongly monotone **Bc**-learning is strictly more powerful than strongly monotone **Ex**-learning.

▶ **Theorem 19.** *We have* $[\tau(\mathbf{SMon})\mathbf{TxtCflItBc}] = [\mathbf{TxtGSMonBc}]$*, i.e. every class of languages which is* **TxtGSMonBc***-identifiable can be learned by a* **CflIt***-learner being strongly monotone on arbitrary texts.*

▶ **Theorem 20.** *Strongly monotone* **Bc***-learning is strictly more powerful than its explanatory counterpart. So we have* $[\mathbf{TxtGSMonEx}] \subset [\mathbf{TxtGSMonBc}]$*. Even stronger, we have* $[\mathbf{TxtGSMonBc}] \setminus [\mathbf{TxtGEx}] \neq \emptyset$.

5 Anomalous and Vacillatory Language Learning

In this section we examine the behavior of different success criteria for learning when paired with the requirement of strong monotonicity. A result in the field of function learning states one does *not* gain additional learning power in allowing the learner to vacillate between finitely many correct hypotheses in the limit [3, 8]. However, in (unrestricted) language identification, **Fex**-learners can indeed infer strictly more classes of languages [5]. We begin our analysis in proving that, when paired with **SMon**, this advantage vanishes once again.

▶ **Theorem 21.** *For strongly monotone language learning, vacillating among finitely many hypotheses does not increase learning power. Thus, we have*
1. $[\mathbf{TxtGSMonEx}] = [\mathbf{TxtGSMonFex}]$*;*
2. $[\mathbf{TxtGSMonEx}^*] = [\mathbf{TxtGSMonFex}^*]$.

We conclude with two theorems establishing the separations given in Figure 3. Note that for finite learning (**Fin**) any learner is strongly monotone as it outputs only a single hypothesis besides "?", hence, $[\mathbf{TxtGSMonFin}^*] = [\mathbf{TxtGFin}^*]$.

▶ **Theorem 22.** *There is a class of languages which can only be inferred if the learner is allowed to make finite error. Thus, we have* $[\mathbf{TxtGSMonFin}^*]\setminus[\mathbf{TxtGSMonBc}] \neq \emptyset$*. Even stronger, we have* $[\mathbf{TxtGSMonFin}^*]\setminus[\mathbf{TxtGBc}] \neq \emptyset$

▶ **Theorem 23.** *The following two separations hold for anomalous and behaviorally correct strongly monotone language learning:*
1. $[\mathbf{TxtGSMonEx}]\setminus[\mathbf{TxtGSMonFin}^*] \neq \emptyset$*;*
2. $[\mathbf{TxtGSMonBc}]\setminus[\mathbf{TxtGSMonEx}^*] \neq \emptyset$.

References

1 D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.
2 G. Baliga, J. Case, W. Merkle, F. Stephan, and W. Wiehagen. When unlearning helps. *Information and Computation*, 206:694–709, 2008.
3 J. Bārzdiņš and K. Podnieks. The theory of inductive inference. In *Mathematical Foundations of Computer Science*, 1973.
4 L. Blum and M. Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.
5 J. Case. The power of vacillation in language learning. *SIAM Journal on Computing*, 28(6):1941–1969, 1999.
6 J. Case and T. Kötzing. Strongly non-U-shaped learning results by general techniques. In *Proc. of COLT (Conference on Learning Theory)*, pages 181–193, 2010.
7 J. Case and S. Moelius. Optimal language learning from positive data. *Information and Computation*, 209:1293–1311, 2011.
8 J. Case and C. Smith. Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science*, 25:193–220, 1983.
9 M. Fulk. Prudence and other conditions on formal language learning. *Information and Computation*, 85:1–11, 1990.
10 E. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
11 S. Jain, T. Kötzing, and F. Stephan. On the role of update constraints and text-types in iterative learning. In *Proc. of ALT (Algorithmic Learning Theory)*, 2014.
12 S. Jain, D. Osherson, J. Royer, and A. Sharma. *Systems that Learn: An Introduction to Learning Theory*. MIT Press, Cambridge, MA, second edition, 1999.
13 K. Jantke. Monotonic and non-monotonic inductive inference of functions and patterns. In *Proc. of Nonmonotonic and Inductive Logic*, pages 161–177, 1991.
14 E. Kinber and F. Stephan. Language learning from texts: Mind changes, limited memory and monotonicity. *Information and Computation*, 123:224–241, 1995.
15 T. Kötzing. *Abstraction and Complexity in Computational Learning in the Limit*. PhD thesis, University of Delaware, 2009. Available online at http://pqdtopen.proquest.com/#viewpdf?dispub=3373055.
16 T. Kötzing. A solution to Wiehagen's thesis. In *Proc. of STACS (Symposium on Theoretical Aspects of Computer Science)*, pages 494–505, 2014.
17 T. Kötzing and R. Palenta. A map of update constraints in inductive inference. In *Proc. of ALT (Algorithmic Learning Theory)*, 2014.
18 S. Lange and T. Zeugmann. Monotonic versus non-monotonic language learning. In *Proc. of Nonmonotonic and Inductive Logic*, pages 254–269, 1993.
19 D. Osherson, M. Stob, and S. Weinstein. Learning strategies. *Information and Control*, 53:32–51, 1982.
20 D. Osherson, M. Stob, and S. Weinstein. *Systems that Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists*. MIT Press, Cambridge, MA, 1986.
21 H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw Hill, New York City, NY, 1967. Reprinted by MIT Press, Cambridge, MA, 1987.
22 G. Schäfer-Richter. *Über Eingabeabhängigkeit und Komplexität von Inferenzstrategien*. PhD thesis, RWTH Aachen, 1984.
23 K. Wexler and P. Culicover. *Formal Principles of Language Acquisition*. MIT Press, Cambridge, MA, 1980.
24 R. Wiehagen. A thesis in inductive inference. In *Proc. of Nonmonotonic and Inductive Logic*, pages 184–207, 1991.

Quantum Query Complexity of Subgraph Isomorphism and Homomorphism

Raghav Kulkarni[1] **and Supartha Podder**[2]

1 **Centre for Quantum Technologies & Nanyang Technological University, Singapore**
kulraghav@gmail.com
2 **Centre for Quantum Technologies & National University of Singapore, Singapore**
supartha@gmail.com

Abstract

Let H be a (non-empty) graph on n vertices, possibly containing isolated vertices. Let $f_H(G) = 1$ iff the input graph G on n vertices contains H as a (not necessarily induced) subgraph. Let α_H denote the cardinality of a maximum independent set of H. In this paper we show:

$$Q(f_H) = \Omega\left(\sqrt{\alpha_H \cdot n}\right),$$

where $Q(f_H)$ denotes the quantum query complexity of f_H.

As a consequence we obtain lower bounds for $Q(f_H)$ in terms of several other parameters of H such as the average degree, minimum vertex cover, chromatic number, and the critical probability.

We also use the above bound to show that $Q(f_H) = \Omega(n^{3/4})$ for any H, improving on the previously best known bound of $\Omega(n^{2/3})$ [16]. Until very recently, it was believed that the quantum query complexity is at least square root of the randomized one. Our $\Omega(n^{3/4})$ bound for $Q(f_H)$ matches the square root of the current best known bound for the randomized query complexity of f_H, which is $\Omega(n^{3/2})$ due to Gröger. Interestingly, the randomized bound of $\Omega(\alpha_H \cdot n)$ for f_H still remains open.

We also study the Subgraph Homomorphism Problem, denoted by $f_{[H]}$, and show that $Q(f_{[H]}) = \Omega(n)$.

Finally we extend our results to the 3-uniform hypergraphs. In particular, we show an $\Omega(n^{4/5})$ bound for quantum query complexity of the Subgraph Isomorphism, improving on the previously known $\Omega(n^{3/4})$ bound. For the Subgraph Homomorphism, we obtain an $\Omega(n^{3/2})$ bound for the same.

1998 ACM Subject Classification F. Theory of Computation, F.1 Computation of Abstract Devices, F.1.1 Models of Computation, G. Mathematics of Computing, G.2 Discrete Mathematics, G.2.2 Graph Theory, Hypergraphs

Keywords and phrases quantum query complexity, subgraph isomorphism, monotone graph properties

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.48

1 Introduction

1.1 Classical and Quantum Query Complexity

The decision tree model (aka the query model), perhaps due to its simplicity and fundamental nature, has been extensively studied in the past and still remains a rich source of many

© Raghav Kulkarni and Supartha Podder;
licensed under Creative Commons License CC-BY

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 48; pp. 48:1–48:13

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

fascinating investigations. In this paper we focus on Boolean functions, i.e., the functions of the form $f : \{0,1\}^n \to \{0,1\}$. A deterministic decision tree T_f for f takes $x = (x_1, \ldots, x_n)$ as an input and determines the value of $f(x_1, \ldots, x_n)$ using queries of the form "is $x_i = 1$?" Let $C(T_f, x)$ denote the cost of the computation, that is the number of queries made by T_f on an input x. The *deterministic decision tree complexity* (aka the deterministic query complexity) of f is defined as

$$D(f) = \min_{T_f} \max_{x} C(T_f, x).$$

We encourage the reader to see an excellent survey by Buhrman and de Wolf [7] on the decision tree complexity of Boolean functions.

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.

One can also define the quantum version of the decision tree model as follows: Start with an N-qubit state $|0\rangle$ consisting of all zeros. We can transform this state by applying an unitary transformation U_0, then we can make a *quantum* query O, which essentially negates the amplitude of each basic state depending on whether the ith bit of the basic state is zero or one. A quantum algorithm with q queries looks like the following: $A = U_q O U_{q-1} \cdots O U_1 O U_0$. Here U_i's are fixed unitary transformation independent of the input x. The final state $A|0\rangle$ depends on the input x only via applications of O. We measure the final state outputing the rightmost qubit (WLOG there are no intermediate measurements). A bounded-error quantum query algorithm A computes f correctly if the final measurement gives the correct answer with probability at least $2/3$ for every input x. *The bounded-error quantum query complexity* of f, denoted by $Q(f)$, is the least q for which f admits a bounded-error quantum algorithm. We refer the reader to a survey by Buhrman and de Wolf [7] for more precise definition.

1.2 Subgraph Isomorphism Problem

Let H be a (non-empty) graph on n vertices, possibly containing isolated vertices and let G be an unknown input graph (on n vertices) given by query access to its edges, i.e, queries of the form "Is $\{i, j\}$ an edge in G?". We say $H \leq G$ if G contains H as a (not necessarily induced) subgraph. Let $f_H : \{0,1\}^{\binom{n}{2}} \to \{0,1\}$ be defined as follows:

$$f_H(G) = \begin{cases} 1 & \text{if } H \leq G \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The well-known Graph Isomorphism Problem asks whether a graph H is isomorphic to another graph G. The Subgraph Isomorphism Problem is a generalization of the Graph Isomorphism Problem where one asks whether H is isomorphic to a subgraph of G. Several central computational problems for graphs such as containing a clique, containing a Hamiltonian cycle, containing a perfect matching can be formulated as the Subgraph Isomorphism Problem by fixing the H appropriately. Given the generality and importance of the problem, people have investigated various restricted special cases of this problem in different models of computation [1] [14]. In the context of query complexity, in 1992 Gröger [9] studied this problem in the randomized setting and showed that $R(f_H) = \Omega(n^{3/2})$, which is the best

known bound to this date. In this paper we investigate this problem in the quantum setting. To the best of our knowledge, quantum query complexity for the Subgraph Isomorphism Problem has not been noted prior to this work when H is allowed to be any graph on n vertices. A special case of this problem when H is of a constant size has been investigated before for obtaining upper bounds [20].

1.3 Subgraph Homomorphism Problem

We also investigate a closely related Subgraph Homomorphism Problem.

A homomorphism from a graph H into a graph G is a function $h : V(H) \to V(G)$ such that: if $(u, v) \in E(H)$ then $(h(u), h(v)) \in E(G)$.

Let $f_{[H]}$ be the function defined as follows: $f_{[H]}(G) = 1$ if and only if H admits a homomorphism into G.

Note that unlike the isomorphism, the homomorphism need not be an injective function from $V(H)$ to $V(G)$. We study the query complexity of the Subgraph Homomorphism Problem towards the end of this paper. In the next section, we review the relevant literature.

1.4 Related Work

Understanding the query complexity of monotone graph properties has a long history. In the deterministic setting the Aanderaa-Rosenberg-Karp Conjecture asserts that one must query all the $\binom{n}{2}$ edges in the worst-case. The randomized complexity of monotone graph properties is conjectured to be $\Omega(n^2)$. Yao [19] obtained the first super-linear lower bound in the randomized setting using the graph packing arguments. Subsequently his bound was improved by King [12] and later by Hajnal [11]. The current best known bound is $\Omega(n^{4/3}\sqrt{\log n})$ due to Chakrabarti and Khot [8]. Moreover, O'Donnell, Saks, Schramm, and Servedio [15] also obtained an $\Omega(n^{4/3})$ bound via a more generic approach for monotone transitive functions. Friedgut, Kahn, and Wigderson [10] obtain an $\Omega(n/p)$ bound where the p is the critical probability of the property. In the quantum setting, Buhrman, Cleve, de Wolf and Zalka [6] were the first to study quantum complexity of graph properties. Santha and Yao [16] obtain an $\Omega(n^{2/3})$ bound for general properties. Their proof follows along the lines of Hajnal's proof.

Gröger [9] obtained an $\Omega(n^{3/2})$ bound for the randomized query complexity of the Subgraph Isomorphism. This is currently the best known bound for the Subgraph Isomorphism Problem. Until very recently[1], it was believed that the quantum query complexity is at least square root of the randomized one. In this paper we address the quantum query complexity of the Subgraph Isomorphism Problem and obtain the square root of the current best randomized bound.

The main difference between the previous work and this one is that all the previous work, including that of Santha and Yao [16], obtained the lower bounds based on an embedding of a *tribe* function [5] on a large number of variables in monotone graph properties[2]. Recall that the tribe function with parameters k and ℓ, is a function $T(k, \ell)$ on $k \cdot \ell$ variables defined as: $\bigvee_{i\in[k]} \bigwedge_{j\in[\ell]} x_{ij}$. This method yields a lower bound of $\Omega(k \cdot \ell)$ for the randomized query complexity and $\Omega(\sqrt{k \cdot \ell})$ for the quantum. We deviate from this line by embedding a threshold function T_n^t instead of a tribe. Recall that $T_n^t(z_1, \ldots, z_n)$ is a function on n

[1] Very recently this has been falsified by Ben-David [22].
[2] Similar tribe-embeddings were used for obtaining lower bounds for metroidal Boolean functions in [13]

variables that evaluates to 1 if and only if at least t of the z_i's are 1. Since the randomized complexity of T_n^t is $\Theta(n)$, this does not give us any advantage for obtaining super-linear randomized lower bounds. However, it *does* yield an advantage for the quantum lower bounds as the quantum query complexity of T_n^t is $\Theta(\sqrt{t(n-t)})$ [17], which can reach up to $\Omega(n)$ for large t. Since this technique works only in the quantum setting, the randomized versions of our bounds remain intriguingly open.

1.5 Our Results

Our main result is a lower bound on the quantum query complexity of the Subgraph Isomorphism Problem for H in terms of the maximum independence number of H.

▶ **Theorem 1.** *For any non-empty H,*

$$Q(f_H) = \Omega\left(\sqrt{\alpha_H \cdot n}\right),$$

where α_H denotes the size of a maximum independent set of H.

▶ **Corollary 2.** *For any non-empty H,*

1. $Q(f_H) = \Omega\left(\frac{n}{\sqrt{d_{avg}(H)}}\right)$,
2. $Q(f_H) = \Omega\left(\frac{n}{\sqrt{\chi_H}}\right)$,
3. $Q(f_H) = \Omega\left(\sqrt{\frac{n}{p}}\right)$,

where $d_{avg}(H)$ denotes the average degree of the vertices of H, χ_H denotes the chromatic number of H, and p denotes the critical probability [10] of H.

In particular, we get an $\Omega(n)$ bound when the graph H is sparse ($|E(H)| = O(n)$), or H has a constant chromatic number, or the critical probability of H is $O(1/n)$. Friedgut, Kahn, and Wigderson [10] show an $\Omega(n/p)$ bound for the randomized query complexity of general monotone properties. Quantization of this bound remains open. General monotone properties can be thought of as the Subgraph Isomorphism for a *family* of minimal subgraphs. The item 3 above, gives a quantization of [10] in the case when the family contains only a single subgraph.

▶ **Corollary 3.** *For any non-empty H,*

$$Q(f_H) = \Omega(n^{3/4}).$$

Prior to this work only an $\Omega(n^{2/3})$ bound was known from the work of Santha and Yao [16] on general monotone graph properties.

We extend our result to the 3-uniform hypergraphs. In particular, we show:

▶ **Theorem 4.** *Let H be a non-empty 3-uniform hypergraph on n vertices. Then,*

$$Q(f_H) = \Omega(n^{4/5}).$$

This improves the $\Omega(n^{3/4})$ bound obtained via the minimum certificate size.

The second part of this paper concerns the Subgraph Homomorphism Problem for H, denoted by $f_{[H]}$. Here we show the following two theorems:

▶ **Theorem 5.** *For any non-empty H,*

$$Q(f_{[H]}) = \Omega(n).$$

▶ **Theorem 6.** *For any non-empty 3-uniform hypergraph H on n vertices:*

$$Q(f_{[H]}) = \Omega(n^{3/2}).$$

Our proofs crucially rely on the duality of monotone functions and appropriate embeddings of tribe and threshold functions. All our lower bounds hold for the approximate degree $\widetilde{\deg}(f)$, which is known to be strictly smaller than the quantum query complexity [4].

Organization

Section 2 contains some preliminaries. Section 3 and Section 4 deal with the Subgraph Isomorphism Problem. Section 3 contains the proofs of Theorem 1, Corollary 2 and Corollary 3. Then Section 4 contains the proof of Theorem 4. The next two sections (Section 5 and Section 6) involve the Subgraph Homomorphism Problem and contains the proof of Theorem 5 and Theorem 6. Finally Section 7 contains conclusion and some open ends.

2 Preliminaries

In this section, we review some preliminary concepts and restate some previously known results.

Let $[n]$ denote the set $\{1, \ldots, n\}$.

▶ **Definition 7** (Dual of a Property)**.** The dual $\mathcal{P}$, denoted by $\mathcal{P}^*$, is:

$$\mathcal{P}^*(x) := \neg\mathcal{P}(\neg x),$$

where $\neg x$ denotes the binary string obtained by flipping each bit in x.

Note that $\mathcal{P}^{**} = \mathcal{P}$ and $Q(\mathcal{P}) = Q(\mathcal{P}^*)$.

A property $\mathcal{P}$ is said to be *monotone increasing* if for every $x \leq y$ we have $\mathcal{P}(x) \leq \mathcal{P}(y)$, where $x \leq y$ denotes $x_i \leq y_i$ for all i.

Note that if $\mathcal{P}$ is monotone, then so is $\mathcal{P}^*$.

A *minimal certificate* of size s for a monotone increasing property $\mathcal{P}$ is an input z such that (a) The hamming weight of z, i.e, $|z|$, is s, (b) $\mathcal{P}(z) = 1$, and (c) for any y with $|y| < s$, $\mathcal{P}(y) = 0$. Every minimal certificate z can be uniquely associated with the subset $S_z := \{i \mid z_i = 1\}$.

▶ **Lemma 8** (Minimal Certificate [7])**.** *If $\mathcal{P}$ has a minimal certificate of size s then*

$$Q(\mathcal{P}) \geq \Omega(\sqrt{s}).$$

We say that two minimal certificates z_1 and z_2 *pack* together, if $S_{z_1} \cap S_{z_2} = \emptyset$.

▶ **Lemma 9** (Packing Lemma [19])**.** *If z_1 is a minimal certificate of $\mathcal{P}$ and z_2 is a minimal certificate of $\mathcal{P}^*$ then z_1 and z_2* cannot *be packed together.*

▶ **Lemma 10** (Turán [2])**.** *If the average degree of a graph G is d then G contains an independent set of size at least $\Omega(n/d)$.*

▶ **Lemma 11** (Extended Turán [2])**.** *If the average degree of a k-uniform hypergraph G is d then G contains an independent set of size at least $\Omega(n/d^{\frac{1}{k-1}})$.*

A Boolean function $f(x_1, \ldots, x_n)$ is called *transitive* if there exists a group Γ acting transitively on the indices $\{1, \ldots, n\}$ such that f is invariant under the action, i.e., for every $\sigma \in \Gamma$ we have $f(x_{\sigma_1}, \ldots, x_{\sigma_n}) = f(x_1, \ldots, x_n)$.

Note that graph properties and hypergraph properties are transitive functions.

▶ **Lemma 12** (Transitive Packing [18]). *Let f be a monotone transitive function on n variables. If f has a minimal certificate of size s then every certificate of f^* must have size at least n/s.*

A *Threshold function* $T_n^t(z_1, \ldots, z_n)$ is a function on n variables such that T_n^t outputs 1 if and only if at least t variables are 1.

We are now ready to prove the quantum query complexity lower bound for the Subgraph Isomorphism Problem.

3 Subgraph Isomorphism for Graphs

Before proving Theorem 1 we first prove two lemmas.

Let S_d denote the star graph with d edges. Then f_{S_d} is the property of having a vertex of at least degree d. First we show:

▶ **Lemma 13.** *For $1 \leq d \leq n-1$,*

$$Q(f_{S_d}) = \Omega(n).$$

Proof. We divide the proof into two cases:

Case 1: $d > n/2$. Fix a clique on the vertices $1, \ldots, \lfloor n/2 \rfloor$ and fix an independent set on the vertices $\lfloor n/2 \rfloor + 1, \ldots, n$. Note that we still have $\lfloor n/2 \rfloor \times \lceil n/2 \rceil$ edge-variables that are not yet fixed. Now as soon as any vertex v from the clique has $(d - \lfloor n/2 \rfloor + 1)$ edges to the independent set present, we have a d-star. Thus f_{S_d} becomes an $OR_{\lfloor n/2 \rfloor} \circ T^{(d-\lfloor \frac{n}{2} \rfloor + 1)}_{\lceil n/2 \rceil}$ function, which has a lower bound of $\Omega(n)$ via the Composition Theorem for quantum query complexity [21].

Case 2: $d \leq n/2$. A minimum certificate of f_{S_d} is a d-star. Now by the Lemma 9 we know that this d-star cannot be packed with any minimal certificate of the dual $f^*_{S_d}$. Thus every vertex in the dual $f^*_{S_d}$ must have degree $> n - d$. Hence the minimal certificate size is at least $\Omega(n^2)$ and $Q(f^*_{S_d}) = Q(f_{S_d}) = \Omega(n)$. ◀

Let t denote the smallest integer such that $f^*_H(K_t) = 1$, where K_t denotes the complete graph on t vertices. Note that $t = \alpha_H + 1$.

▶ **Lemma 14.**

$$Q(f_H) \geq \Omega(\sqrt{t(n-t)}).$$

Proof. We *embed* T_n^t in f^*_H (on inputs of Hamming weight $t-1$ and t) via the following mapping: Let $x_{ij} := z_i \cdot z_j$ and let $f'(z_1, \ldots, z_n) := f^*_H(\{x_{ij}\})$. Note that $f' \equiv T_n^t$. Also note[3] that $Q(f_H) = Q(f^*_H)$ and $\widetilde{deg}(f') \leq 2 \cdot \widetilde{deg}(f^*_H)$. Since $Q(f) \geq \widetilde{deg}(f)$, it remains to prove the following:

[3] Since $x_{ij} = z_i.z_j$, every monomial of f^*_H of size d becomes a monomial of size at most $2d$ in f'.

▶ **Claim 1.** $\widetilde{deg}(f') = \Omega(\sqrt{t(n-t)})$

We need the following lemma due to Paturi [17]:

▶ **Lemma 15.** *Let g be a function on n variables such that $g(z) = 0$ for all z with $|z| = t-1$ and $g(z) = 1$ for all z with $|z| = t$. Then: $\widetilde{\deg}(g) = \Omega(\sqrt{t(n-t)})$.*

Proof of Claim 1. Note that f' $(\equiv T_n^t)$ satisfies the condition of the Lemma 15. ◀

This finishes the proof of the Lemma 14. ◀

Now we are in a position to prove the Theorem 1.

Proof of Theorem 1. Recall that t denotes the smallest integer such that $f_H^*(K_t) = 1$. We divide the proof into two cases:

Case 1: $t > n/2$. In this case, we reduce the f_H to f_{S_p} for some $p = \Omega(n)$. Let ν_H denote the minimum vertex cover size of H. Since $t > n/2$, we have $\nu_H \leq n/2$. When $\nu_H = 1$ the property is trivially a star property and from the Lemma 13 we already get $Q(f_H) = \Omega(n)$. Otherwise we restrict f_H by picking a clique on $\nu_H - 1$ vertices and joining all the other $n - \nu_H + 1$ remaining vertices to each vertex in this clique. The resulting function takes a graph on $p = n - \nu_H + 1$ vertices as input. Let's denote these vertices by S.

As the clique on $\nu_H - 1$ vertices cannot accommodate all the vertices in the minimum vertex cover of H, in order to satisfy the property f_H at least one vertex v in the vertex cover must occur among S. This vertex v may have some edges incident on the vertices of the clique and some edges incident on the vertices of S. In the restriction all the possible edges to the clique are already present. Thus as soon as we have the remaining edges to the vertices of S the property f_H is satisfied.

Hence the property is now reduced to finding a star graph with d edges, f_{S_d} where d is defined as follows: Let C be a vertex cover. Furthermore let $d_{out}(v)$ denote the number of neighbors of a vertex v in C that are outside C and $d_{out}(C)$ be the minimum over all such vertices v in C. Then d is the minimum $d_{out}(C)$ of a minimum vertex cover C of H (minimized over all the minimum vertex covers). Thus as soon as we have the star graph with d edges, our original restricted f_H is satisfied.

Now from the Lemma 13 we get $Q(f_H) = \Omega(n)$.

Case 2: $t \leq n/2$. Note that $t > \alpha_H$. And since $t \leq n/2$, we have $n - t = \Omega(n)$. Hence from the Lemma 14 we get the bound of $\Omega(\sqrt{t(n-t)})$, which is $\Omega(\sqrt{\alpha_H \cdot n})$. ◀

Proof of Corollary 2.

1. From Turán's theorem, we have: $\alpha_H \geq n/(2 \cdot d_{avg}(H))$.
2. Since $\alpha_H \cdot \chi_H \geq n$ we have $\alpha_H \geq n/\chi_H$.
3. Since the critical probability of H is p, the average degree of H is at most pn. Hence from Corollary 2(1), we get the $\Omega(n/p)$ bound. ◀

Proof of Corollary 3. When $d_{avg}(H) \geq \sqrt{n}$ the Lemma 8 gives an $\Omega(n^{3/4})$ bound. Otherwise when $d_{avg}(H) < \sqrt{n}$ we use the Corollary 2(1), which gives the same bound. ◀

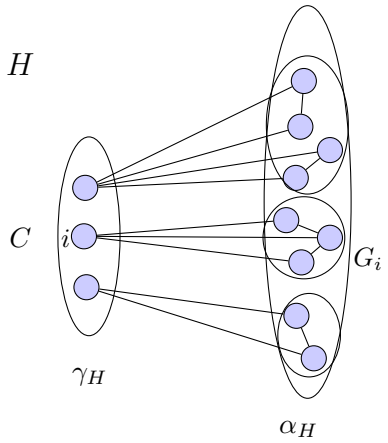


Figure 1 Structure of H.

4 Subgraph Isomorphism for 3-Uniform Hypergraphs

In this section we extend the $\Omega(n^{3/4})$ bound for the Subgraph Isomorphism for graphs to the 3-uniform hypergraphs. In particular, we obtain an $\Omega(n^{4/5})$ bound for the Subgraph Isomorphism for 3-uniform hypergraphs, improving upon the $\Omega(n^{3/4})$ bound obtained via the minimal certificate size.

Before going to the proof of Theorem 4, we extend the Lemma 14 to the 3-uniform hypergraphs. Let t be the smallest such that $f^*_H(K_t) = 1$. Note that $t = \alpha_H + 1$.

▶ **Lemma 16.** *Let H be a 3-uniform hypergraph on n vertices. Then:*

$$Q(f_H) \geq \Omega(\sqrt{t(n-t)}).$$

Proof. Let $T^t_n(z_1, \ldots, z_n)$ denote the threshold function on n variables that outputs 1 if and only if at least t variables are 1. We *embed* a T^t_n in f^*_H (on inputs of Hamming weight $t-1$ and t) via the following mapping: Let $x_{ijk} := z_i \cdot z_j \cdot z_k$. Let $f'(z_1, \ldots, z_n) := f^*_H(\{x_{ijk}\})$. Note that $f' \equiv T^t_n$. Also note[4] that the $\widetilde{deg}(f') \leq 3 \cdot \widetilde{deg}(f^*_H)$. Since $Q(f) \geq \widetilde{deg}(f)$, it remains to prove that $\widetilde{deg}(f') = \Omega(\sqrt{t(n-t)})$, which follows from the Lemma 15. ◀

Now we give a proof of the Theorem 4.

Proof of Theorem 4. We divide the proof into two main cases.

Case 1: $\alpha_H > n/2$. Let H be a 3-uniform hypergraph on n vertices. Let C denote a minimal vertex cover of H. Let $|C| = \nu_H$. Note that the hypergraph induced on $V - C$ is empty. For a vertex $i \in C$ let G_i denote the projection graph of the neighbors of i on $V - C$, i.e., $(i, u, v) \in E(H)$ (see Figure 1).

Let $\mathcal{P}_H$ denote the restriction of the f_H defined as follows: set the hyper-clique on $\nu_H - 1$ vertices to be present and add all the hyper-edges incident on the vertices of this clique. Let S denote the set of remaining $n - \nu_H + 1$ vertices. The hyper-edges among S are still undetermined. Note that $\mathcal{P}_H$ is a non-trivial property of $n - \nu_H + 1$ vertex hypergraphs,

[4] Since $x_{ijk} = z_i \cdot z_j \cdot z_k$, every monomial of f^*_H of size d becomes a monomial of size at most $3d$ in f'.

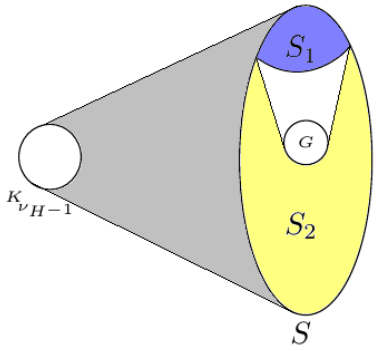


Figure 2 The Restriction $\mathcal{P}''$: the hyper-clique K_{ν_H-1} is present, all the hyper-edges in the gray area are present. All the hyper-edges in blue region are present, all the hyper-edges in yellow region are absent. G is fixed. White region symbolizes the hyper-edges with two-end points in S_1 and one in S_2 to be absent and one end point in S_1 and two end points in S_2 to be undertermined.

since H cannot be contained in the $\nu_H - 1$ hyper-clique and edges incident on it as the minimum vertex cover size of H is ν_H.

▶ **Lemma 17.** *If* $\exists C$, $\exists i : |E(G_i)| = O(n^{7/5})$, *then* $Q(f_H) = \Omega(n^{4/5})$.

Proof. In this case $\mathcal{P}_H$ has a certificate of size $O(n^{7/5})$. Hence from Lemma 12 the certificate size of $\mathcal{P}_H^*$ is $\Omega(\frac{n^3}{n^{7/5}}) = \Omega(n^{8/5})$. Now from the Lemma 8 we get $Q(f_H) = \Omega(n^{4/5})$. ◀

Hence from now onwards we assume that for all $i, |E(G_i)| = \Omega(n^{7/5})$. Moreover, we may also note that $\nu_H = O(n^{1/5})$, if not we have a minimal certificate for $\mathcal{P}_H$ of size $\Omega(n^{8/5})$. And hence from the Lemma 8 we already get the desired bound of $Q(f_H) = \Omega(n^{4/5})$.

Now we obtain a restriction $\mathcal{P}'$ of $\mathcal{P}_H$ as follows: divide S into two parts say S_1 and S_2 of size n_1 and n_2 respectively, where we choose $n_1 = \Theta(n^{1/5})$ and $n_2 = \Theta(n)$. Set all the hyper-edges within S_1 to be present and set all the hyper-edges within S_2 to be absent. Also set all the hyper-edges with two endpoints in S_1 and one in S_2 to be absent. Only possible undetermined hyper-edges are with one endpoint in S_1 and two in S_2. Note that even after setting all hyper edges in S_1 to be present we can safely assume that the property remains non-trivial. Otherwise we would have a certificate for $\mathcal{P}_H$ of size $O(n^{3/5})$, hence the dual will have large ($\gg \Omega(n^{8/5})$) certificates.

Let G be a projection graph among all the G_i's containing the least number of edges inside S_2. We further obtain a restriction $\mathcal{P}''$ by fixing a copy of G inside S_2 and allowing only potential hyper-edges with one endpoint in S_1 and the other two endpoints forming an edge of G (see Figure 2).

Let C be a vertex cover of H of minimum cardinality. Note that in order to satisfy $\mathcal{P}_H$, at least one of the vertices from C must move to S. Let us call a vertex of C that moves to S as pivot. Let k be the largest integer such that P_H has a minimal certificate with k pivots. Note that from Lemma 17 each pivot has $\Omega(n^{7/5})$ edges incident on it. Therefore if $k > n_1/2$ then we already have a minimal certificate whose size is $\Omega(n^{8/5})$. Otherwise: $k \leq n_1/2$. First we argue that any pivot must belong to S_1. If on the contrary, it were in S_2 then the only possible edges incident on such a pivot v are of the form (v, u, w) where $u \in S_1$ and $w \in S_2$. But there can be at most $O(n^{6/5})$ such edges, which contradicts the fact that any pivot supports at least $\Omega(n^{7/5})$ edges. Let the degree of a pivot be the number of edges inside S_2 that are adjacent to it. Next we choose a certificate for $\mathcal{P}_H$ with at most $k \leq n_1/2$

pivots such that the degree of the minimum degree pivot is minimum possible. Then we leave aside the minimum degree pivot in this certificate and fix the $k-1$ other pivots and their projection on S_2. From each of the remaining $n_1 - k + 1$ vertices we keep the projection of the minimum degree pivot on S_2 as the only possible edges.

Now from minimality of our choice at least one of these vertices must have all these $\Omega(n^{7/5})$ edges in order for the original graph to contain H. Thus we get an $\bigvee_{\Omega(n^{1/5})} \bigwedge_{\Omega(n^{7/5})}$ function as the restriction.

Since an $OR \circ AND$ on m variables admits an $\Omega(\sqrt{m})$ lower bound on the quantum query complexity we get $Q(f_H) = \Omega(n^{4/5})$.

Case 2: $\alpha_H \leq n/2$. In this case we use Lemma 16. Since $n - \alpha_H \geq n/2$, we immediately get $Q(f_H) = \Omega(\sqrt{\alpha_H \cdot n})$.

Now in order to prove Theorem 4, we need to show that the above bound always yields an $\Omega(n^{4/5})$ bound. Thus we further consider two cases based on the average degree. And in fact this gives us a larger $\Omega(n^{5/6})$ bound for the case 2.

Let d denote the average degree of H.

Case 2a: $d > n^{2/3}$. In this case $|E(H)| > \Omega(n^{5/3})$. Hence from Lemma 8 we get an $\Omega(n^{5/6})$ bound.

Case 2b: $d \leq n^{2/3}$. Here we use the extension of Turán's Theorem (see Lemma 11) to 3-uniform hypergraphs. Since the average degree is $O(n^{2/3})$, we get $\alpha_H \geq \Omega(n^{2/3})$. Therefore from Lemma 16 we get $Q(f_H) = \Omega(n^{5/6})$.

This completes the proof of Theorem 4. ◀

In the following two sections we study the Subgraph Homomorphism Problem. We first prove the quantum query complexity lower bounds for graphs and then for 3-uniform hypergraphs.

5 Subgraph Homomorphism for Graphs

Proof of Theorem 5. Let $\chi(H)$ denote the chromatic number of H. Note that H has a homomorphism into K_t for $t = \chi(H)$, i.e., $f_{[H]}(K_{t-1}) = 0$ and $f_{[H]}(K_t) = 1$.

We consider the following two cases.

Case 1: $t \geq n/2$. As K_{t-1} is a no instance and K_t is an yes instance for the property $f_{[H]}$, the minimum certificate size, $m(f_{[H]}) = \Omega(t^2) = \Omega(n^2)$. Hence from Lemma 8 we get an $\Omega(n)$ lower bound on the quantum query complexity.

Case 2: $t < n/2$. Consider the following restriction: We set a clique K_{t-2} on $t-2$ vertices to be present and we also set all the edges from the remaining $n - t + 2$ vertices to this clique to be present. Now notice that as soon as there is an edge between any two of the remaining $n-t+2$ vertices, we have a K_t. Hence the property $f_{[H]}$ has become the property of containing an edge among the $n - t + 2$ vertices. Since $t < n/2$, this is an OR function on $\Omega(n^2)$ variables. Thus $Q(f_{[H]}) = \Omega(n)$. ◀

▶ Remark. Our proof in fact shows that the minimum certificate size of either $f_{[H]}$ or $f^*_{[H]}$ is $\Omega(n^2)$. Hence we also obtain

$$R(f_{[H]}) = \Omega(n^2)$$

.

We now proceed to prove the quantum query complexity lower bound of the Subgraph Homomorphism Problem for 3-uniform hypergraph.

6 Subgraph Homomorphism for 3-Uniform Hypergraphs

Proof of Theorem 6. Proof of this theorem is similar to proof of Theorem 5.

Let $\chi(H)$ denote the chromatic number of H. Note that H has a homomorphism into K_t for $t = \chi(H)$, i.e., $f_{[H]}(K_{t-1}) = 0$ and $f_{[H]}(K_t) = 1$.

We consider the following two cases.

Case 1: $t \geq n/2$. Unlike the graph homomorphism case, we cannot claim the presence of a K_t in this case. However we can still use the following fact:

▶ **Fact 1** (Alon [3]). *If H is a 3-uniform hypergraph which is not k colorable then*

$$|E(H)| = \Omega(k^3).$$

Therefore, the minimum certificate size $m(f_{[H]}) = \Omega(t^3) = \Omega(n^3)$. Hence from Lemma 8 we get an $\Omega(n^{3/2})$ lower bound on the quantum query complexity.

Case 2: $t < n/2$. Consider the following restriction: We set a clique K_{t-3} on $t-3$ vertices to be present and we also set all the edges from remaining $(n-t+3)$ vertices to this clique to be present. Now notice that as soon as there is an edge between any three of the remaining $(n-t+3)$ vertices, we have a K_t. Hence the property $f_{[H]}$ has become the property of containing an edge among the $n-t+3$ vertices. Since $t < n/2$, this is an OR function on $\Omega(n^3)$ variables. Thus $Q(f_{[H]}) = \Omega(n^{3/2})$. ◀

7 Conclusion & Open Ends

We obtained an $\Omega(n^{3/4})$ lower bound for the quantum query complexity of Subgraph Isomorphism Problem for graphs, improving upon previously known $\Omega(n^{2/3})$ bound for the same. We extend our result to the 3-uniform hypergraphs by exhibiting an $\Omega(n^{4/5})$ bound, which improves on previously known $\Omega(n^{3/4})$ bound. Besides the obvious question of settling the randomized and quantum query complexity of the Subgraph Isomorphism problem, there are a few interesting questions that might be approachable. We list some of them below:

▶ **Question 1.** *Is it true that for any n-vertex graph H we have:*

(a) $R(f_H) = \Omega(\alpha_H \cdot n)$*?*

(b) $R(f_H) = \Omega(n^2/d^H_{avg}))$*?*

(c) $R(f_H) = \Omega(n^2/\chi_H)$*?*

▶ **Question 2.** *Is it true that for any 3-uniform hypergraph H we have:*

$$Q(f_H) = \Omega(n)?$$

Note that, in the proof of Theorem 4 we managed to get a slightly stronger $\Omega(n^{5/6})$ bound for the case 2. Thus an improved lower bound of $\Omega(n^{5/6})$ for the case 1 (when $\alpha_H > n/2$) would improve the overall bound.

Acknowledgement. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve this article. The authors also thank Biswaroop Maiti for the discussion in the beginning phase of this work.

References

1 Wikipedia – Subgraph Isomorphism. https://en.wikipedia.org/wiki/Subgraph_isomorphism_problem.

2 Miklós Ajtai, János Komlós, Janos Pintz, Joel Spencer, and Endre Szemerédi. Extremal Uncrowded Hypergraphs. *Journal of Combinatorial Theory, Series A*, 32(3):321–335, 1982.

3 Noga Alon. Hypergraphs With High Chromatic Number. *Graphs and Combinatorics*, 1(1):387–389, 1985.

4 Andris Ambainis. Polynomial Degree Vs. Quantum Query Complexity. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2003.*, pages 230–239. IEEE, 2003.

5 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, Ronald de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM (JACM)*, 48(4):778–797, 2001.

6 Harry Buhrman, Richard Cleve, Ronald de Wolf, Christof Zalka. Bounds for small-error and zero-error quantum algorithms. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1999*, pages 358–368. IEEE, 1999.

7 Harry Buhrman and Ronald de Wolf. Complexity Measures and Decision Tree Complexity: A Survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

8 Amit Chakrabarti and Subhash Khot. Improved Lower Bounds on the Randomized Complexity of Graph Properties. *Random Structures & Algorithms*, 30(3):427–440, 2007.

9 Gröger Hans Dietmar. On the Randomized Complexity of Monotone Graph Properties. *Acta Cybernetica*, 10(3):119–127, 1992.

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 Péter Hajnal. An $\Omega(n^{4/3})$ Lower Bound on the Randomized Complexity of Graph Properties. *Combinatorica*, 11(2):131–143, 1991.

12 Valerie King. Lower Bounds on the Complexity of Graph Properties. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC)*, pages 468–476. ACM, 1988.

13 Raghav Kulkarni and Miklos Santha. Query complexity of matroids. In *Algorithms and Complexity*, pages 300–311. Springer, 2013.

14 Yuan Li, Alexander Razborov, and Benjamin Rossman. On the AC^0 Complexity of Subgraph Isomorphism. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2014*, pages 344–353. IEEE, 2014.

15 Ryan O'Donnell, Michael Saks, Oded Schramm, and Rocco A Servedio. Every Decision Tree Has an Influential Variable. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2005*, pages 31–39. IEEE, 2005.

16 Unpublished Manuscript of Miklos Santha and Andrew Chi-Chih Yao.

17 Ramamohan Paturi. On the Degree of Polynomials That Approximate Symmetric Boolean Functions (Preliminary Version). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing (STOC), 1992*, pages 468–474. ACM, 1992.

18 Xiaoming Sun, Andrew C Yao, and Shengyu Zhang. Graph Properties and Circular Functions: How Low Can Quantum Query Complexity Go? In *Proceedings of the 19th IEEE Annual Conference on Computational Complexity (CCC), 2004*, pages 286–293. IEEE, 2004.

19 Andrew Chi-Chih Yao. Lower Bounds to Randomized Algorithms for Graph Properties In *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1987*, pages 393–400. IEEE, 1987.

20 Troy Lee, Frédéric Magniez and Miklos Santha. A Learning Graph Based Quantum Query Algorithm for Finding Constant-Size Subgraphs arXiv:1109.5135, 2011.

21 Troy Lee, Rajat Mittal, Ben W. Reichardt, Robert Spalek, and Mario Szegedy. Quantum Query Complexity of State Conversion In *Proceedings of the 52nd IEEE Annula Symposium on Foundations of Computer Science (FOCS), 2011* page 344-353. IEEE, 2011.

22 Shalev Ben-David. A Super-Grover Separation Between Randomized and Quantum Query Complexities Electronic Colloquium on Computational Complexity (ECCC), `http://eccc.hpi-web.de/report/2015/108`, 2015

Faster Exact and Parameterized Algorithm for Feedback Vertex Set in Tournaments

Mithilesh Kumar[1] **and Daniel Lokshtanov**[2]

1 **Department of Informatics, University of Bergen, Norway**
Mithilesh.Kumar@ii.uib.no
2 **Department of Informatics, University of Bergen, Norway**
daniello@ii.uib.no

Abstract

A *tournament* is a directed graph T such that every pair of vertices is connected by an arc. A *feedback vertex set* is a set S of vertices in T such that $T-S$ is acyclic. In this article we consider the Feedback Vertex Set problem in tournaments. Here the input is a tournament T and an integer k, and the task is to determine whether T has a feedback vertex set of size at most k. We give a new algorithm for Feedback Vertex Set in Tournaments. The running time of our algorithm is upper-bounded by $O(1.6181^k + n^{O(1)})$ and by $O(1.466^n)$. Thus our algorithm simultaneously improves over the fastest known parameterized algorithm for the problem by Dom et al. running in time $O(2^k k^{O(1)} + n^{O(1)})$, and the fastest known exact exponential-time algorithm by Gaspers and Mnich with running time $O(1.674^n)$. On the way to proving our main result we prove a strengthening of a special case of a graph partitioning theorem due to Bollobás and Scott. In particular we show that the vertices of any undirected m-edge graph of maximum degree d can be colored white or black in such a way that for each of the two colors, the number of edges with both endpoints of that color is between $m/4 - d/2$ and $m/4+d/2$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Parameterized algorithms, Exact algorithms, Feedback vertex set, Tournaments, Graph partitions

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.49

1 Introduction

A *feedback vertex set* in a graph G is a vertex set S such that $G-S$ is acyclic. For undirected graphs this means that $G-S$ is a forest, while for directed graphs this implies that $G-S$ is a directed acyclic graph (DAG). In the Feedback Vertex Set (FVS) problem we are given as input an *undirected* graph G and integer k, and asked whether there exists a feedback vertex set of size at most k. The corresponding problem for directed graphs is called Directed Feedback Vertex Set (DFVS). Both problems are NP-complete [12] and have been extensively studied from the perspective of approximation algorithms [1, 10], parameterized algorithms [4, 6, 14], exact exponential-time algorithms [16, 18] as well as graph theory [9, 17].

In this paper we consider a restriction of DFVS, namely the Feedback Vertex Set in Tournaments (TFVS) problem, from the perspective of parameterized algorithms and exact exponential-time algorithms. We refer to the textbooks of Cygan et al. [5] and Fomin and Kratsch [11] for an introduction to these fields. A *tournament* is a directed graph T such that every pair of vertices is connected by an arc, and TFVS is simply DFVS when the input graph is required to be a tournament.


© Mithilesh Kumar and Daniel Lokshtanov;
licensed under Creative Commons License CC-BY
33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 49; pp. 49:1–49:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


Even this restricted variant of DFVS has applications in voting systems and rank aggregation [8], and is quite well-studied [3, 8, 13, 15]. TFVS was shown to be fixed parameter tractable by Raman and Saurabh [15], who obtained an algorithm with running time $O(2.42^k \cdot n^{O(1)})$. In 2006, Dom et al. [7] (see also [8]) gave an algorithm for TFVS with running time $2^k n^{O(1)}$. Prior to our work this was the fastest known parameterized algorithm for the problem. The fastest exact exponential-time algorithm for TFVS was due to Gaspers and Mnich [13] and has running time $O(1.674^n)$.

Our main result is a new algorithm for TFVS. The running time of our algorithm is upper bounded by $O(1.6181^k + n^{O(1)})$ and by $O(1.466^n)$. Thus, we give a single algorithm that simultaneously significantly improves over the previously best known parameterized algorithm and exact exponential-time algorithm for the problem. It is worth noting that the algorithm of Gaspers and Mnich [13] also lists all inclusion minimal feedback vertex sets in the input tournament, while our algorithm can not be used for this purpose.

On the way to proving our main result we prove a balanced edge partition theorem for general undirected graphs. In particular we give a polynomial time algorithm that given an undirected m-edge graph G of maximum degree d, colors the vertices white or black in such a way that for each of the two colors, the number of edges with both endpoints of that color is between $m/4 - d/2$ and $m/4+d/2$. Our partition theorem is a sharper and constructive variant of a special case of a more general result due to Bollobás and Scott [2, Theorem 3.1].

Methods. As a preliminary step our algorithm applies the kernel of Dom et al. [8] to ensure that the number of vertices in the input tournament is upper bounded by $O(k^3)$. After this step, our algorithm has three phases.

In the first phase the algorithm finds, in sub-exponential-time, a "large enough" set M of vertices *disjoint* from the solution H sought for, such that M is evenly distributed in the topological ordering of $T - H$. From the set M we can infer a rough sketch of the unique topological ordering of $T - H$ without knowing the solution H. More concretely, every vertex v gets a tentative position in the ordering, and we know that if v is not deleted, then v's position in the topological order of $T - H$ is close to this tentative position.

We can now use this tentative ordering to identify *conflicts* between two vertices u and v. Two vertices u and v are in conflict if their tentative positions are so far apart that we know the order in which they have to appear in the topological sort of $T - H$, but the arc between u and v goes in the opposite direction. Thus, if u and v are in conflict then at least one of them has to be in the solution feedback vertex set H.

The second phase of the algorithm eliminates vertices that are in conflict with more than one other vertex. Suppose that u is in conflict with both v and w. If u is not deleted then both v and w have to be deleted. The algorithm finds the optimal solution by branching and recursively solves the instance where u is deleted, and the instance where u is not deleted but both v and w are deleted. This branching step is the bottleneck of the algorithm and gives rise to the $O(1.6181^k n^{O(1)})$ and the $O(1.466^n)$ running time bounds.

The third and last phase of the algorithm deals with the case where every vertex has at most one conflict. Here we apply a divide and conquer approach that is based on the partitioning theorem.

Organization of the paper. In Section 2 we set up definitions and notation, and state a few useful preliminary results. Section 3 describes and analyzes the first phase of the algorithm. Section 4 contains the second phase, as well as the final analysis of the correctness and running time of the entire algorithm, conditioned on the correctness and running time bound

of the third and last phase. In Section 5 we formally state and prove our new decomposition theorem for undirected graphs, while the description and analysis of the third phase of the algorithm is deferred to Section 6.

2 Preliminaries

In this paper, we work with graphs that do not contain any self loops. A *multigraph* is a graph that may contain more than one edge between the same pair of vertices. A graph is *mixed* if it can contain both directed and undirected edges. We will be working with mixed multigraphs; graphs that contain both directed and undirected edges, and where two vertices may have several edges between them.

When working with a mixed multigraph G we use $V(G)$ to denote the vertex set, $E(G)$ to denote the set of directed edges, and $\mathcal{E}(G)$ to denote the set of undirected edges of G. A directed edge from u to v is denoted by uv. A *supertournament* is a directed graph T such that for every pair of vertices u, v at least one (and possibly both) edges uv and vu are edges of T. Thus, every tournament is a supertournament, but not vice versa.

Graph Notation. In a directed graph D, the set of *out-neighbors* of a vertex v is defined as $N^+(v) := \{u | vu \in E(D)\}$. Similarly, the set of *in-neighbors* of a vertex v is defined as $N^-(v) := \{u | uv \in E(D)\}$. A *triangle* in a directed graph is a directed cycle of length 3. Note that in this paper, whenever the term triangle is used it refers to a directed triangle. A *topological sort* of a directed graph D is a permutation $\pi : V(D) \to [n]$ of the vertices of the graph such that for all edges $uv \in E(D)$, $\pi(u) < \pi(v)$. Such a permutation exists for a directed graph if and only if the directed graph is acyclic. For an acyclic tournament, the topological sort is unique.

For a graph or multigraph G and vertex v, $G - v$ denotes the graph obtained from G by deleting v and all edges incident to v. For a vertex set S, $G - S$ denotes the graph obtained from G by deleting all vertices in S and all edges incident to them.

For any set of edges C (directed or undirected) and set of vertices X, the set $V_X(C)$ represents the subset of vertices of X which are incident on an edge in C. For a vertex $v \in V(G)$, the set $N_C(v)$ represents the set of vertices $w \in V(G)$ such that there is an undirected edge $wv \in C$.

Fixed Parameter Tractability. A *parameterized problem* Π is a subset of $\Sigma^* \times \mathbb{N}$. A parameterized problem Π is said to be *fixed parameter tractable*(FPT) if there exists an algorithm that takes as input an instance (I, k) and decides whether $(I, k) \in \Pi$ in time $f(k) \cdot n^c$, where n is the length of the string I, $f(k)$ is a computable function depending only on k and c is a constant independent of n and k.

A *kernel* for a parameterized problem Π is an algorithm that given an instance (T, k) runs in time polynomial in $|T|$, and outputs an instance (T', k') such that $|T'|, k' \leq g(k)$ for a computable function g and $(T, k) \in \Pi$ if and only if $(T', k') \in \Pi$. For a comprehensive introduction to FPT algorithms and kernels, we refer to the book by Cygan et al. [5].

Preliminary Results. If a tournament is acyclic then it does not contain any triangles. It is a well-known and basic fact that the converse is also true, see e.g. [8].

▶ **Lemma 1** ([8]). *A tournament is acyclic if and only if it contains no triangles.*

Lemma 1 immediately gives rise to a folklore greedy 3-approximation algorithm for TFVS: as long as T contains a triangle, delete all the vertices in this triangle.

▶ **Lemma 2** (folklore). *There is a polynomial time algorithm that given as input a tournament T and integer k, either correctly concludes that T has no feedback vertex set of size at most k or outputs a feedback vertex set of size at most $3k$.*

In fact, TFVS has a polynomial time factor 2.5-approximation, due to Cai et al. [3]. However, the simpler algorithm from Lemma 2 is already suitable to our needs.

The preliminary phase of our algorithm for TFVS is the kernel of Dom et al. [8]. We will need some additional properties of this kernel that we state here. Essentially, Lemma 3 allows us to focus on the case when the number of vertices in the input tournament is $O(k^3)$.

▶ **Lemma 3** ([8]). *There is a polynomial time algorithm that given as input a tournament T and integer k, runs in polynomial time and outputs a tournament T' and integer k' such that $|V(T')| \leq |V(T)|$, $|V(T')| = O(k^3)$, $k' \leq k$, and T' has a feedback vertex set of size at most k' if and only if T has a feedback vertex set of size at most k.*

3 Finding an Undeletable, Evenly Spread Out Set

Consider a tournament T that has a feedback vertex set H of size at most k. Then $T - H$ is acyclic. Consider now the topological order of $T - H$. Let M be the set of vertices of $T - H$ whose position in the topological order is congruent to 0 mod $\log^2 k$. We have now found a set disjoint from H such that, in the topological order of $T - H$ the distance between two consecutive vertices of M is $O(\log^2 k)$. We shall see later in the article that having such a set at our disposal is very useful for finding the optimum feedback vertex set H. Of course there is a catch; we defined M using the solution H, but we want to use M to find the solution H. In the rest of this section we show how to find a set M with the above properties without knowing the optimum feedback vertex set H in advance. We begin with a few definitions.

▶ **Definition 4.** Let D be a directed graph. For any pair of vertices $u, v \in V(D)$ the set $\mathit{between}(D, u, v)$ is defined as $N^+(u) \cap N^-(v) \setminus \{u, v\}$.

Observe that for an acyclic tournament T, $\mathit{between}(T, u, v)$ is exactly the set of vertices coming after u and before v in the unique topological ordering of T.

▶ **Definition 5.** Let D be a directed graph and $S \subseteq V(D)$. Two vertices $u, v \in S$ are called *S-consecutive* if $uv \in E(D)$ and $\mathit{between}(D, u, v) \cap S = \emptyset$.

In an acyclic tournament T and vertex set S, two vertices u and v in S are S-consecutive if no other vertex of S appears between u and v in the topological ordering.

▶ **Definition 6.** Let D be a directed graph and $S \subseteq V(D)$. We define the set of S-blocks in D. Each pair of *S-consecutive* vertices u and v defines the S-block $\mathit{between}(D, u, v)$. Further, each vertex $u \in S$ with no in-neighbors in S defines an S-block $N^-(u)$. Each vertex $u \in D$ with no out-neighbors in S defines the S-block $N^+(u)$. The *size* of an S-block is its cardinality.

In an acyclic tournament T the S-blocks form a partition of $V - S$, where two vertices are in the same block if and only if no vertex of S appears between them in the topological order of T.

For example, consider an acyclic tournament $T = u_0u_1...u_{11}$ where vertices are topologically sorted. Let $S = \{u_i \mid i \mod 4 = 1\}$. $\mathit{between}(T, u_1, u_9) = \{u_2, u_3, u_4, u_5, u_6, u_7, u_8\}$. u_5 and u_9 are S-consecutive and $\{u_6, u_7, u_8\}$ is an S-block. The set of all S-blocks in T is $\{\{u_0\}, \{u_2, u_3, u_4\}, \{u_6, u_7, u_8\}, \{u_{10}, u_{11}\}\}$.

▶ **Lemma 7.** *There exists an algorithm that given a tournament T with $|V(T)| = O(k^3)$ where k is an integer, outputs a family of sets $\mathcal{M}, |\mathcal{M}| = 2^{O(\frac{k}{\log k})}$ in $2^{O(\frac{k}{\log k})}$ time, such that for every feedback vertex set H of T of size at most k, $\exists M \in \mathcal{M}$, such that :*
1. *$M \cap H = \emptyset$,*
2. *the size of any M-block in $T - H$ is at most $2\log^2 k$.*

Furthermore, $\mathcal{M}$ can be enumerated in polynomial space.

Proof. Let X be a feedback vertex set of size at most $3k$ obtained using Lemma 2. Let $Y := V(T) \setminus X$ and $v_0 v_1 ... v_{|Y|-1}$ be the topological sort of $T[Y]$ such that the edges in $T[Y]$ are directed from left to right. Color Y using $\lfloor \log^2 k \rfloor$ colors such that for each $c' \in [0, ..., |Y| - 1]$, $v_{c'}$ gets color c' mod $\lfloor \log^2 k \rfloor$. For each $c \in [0, ..., \lfloor \log^2 k \rfloor - 1]$, let Y_c be the set of vertices in Y which get color c. Each $M \in \mathcal{M}$ is specified by a 4-tuple $\langle c, \hat{H}, \hat{R}, \hat{X} \rangle$ where

- c is a color in the above coloring of Y,
- $\hat{H} \subseteq Y_c$ such that $|\hat{H}| \leq \frac{k}{\log^2 k}$,
- $\hat{R} \subseteq Y \setminus Y_c$, $|\hat{R}| \leq |\hat{H}|$, and
- $\hat{X} \subseteq X$ such that $|\hat{X}| \leq \frac{3k}{\log^2 k}$.

For each 4-tuple $\langle c, \hat{H}, \hat{R}, \hat{X} \rangle$, let $M := (Y_c \setminus \hat{H}) \bigcup \hat{R} \bigcup \hat{X}$. Hence, $|\mathcal{M}|$ is upper bounded by the maximum number of such 4-tuples.

$$\begin{aligned} |\mathcal{M}| &\leq 2^{\log(\log^2 k)} \times O(k^3)^{\frac{2k}{\log^2 k}} \times (3k)^{\frac{3k}{\log^2 k}} \\ &= 2^{O(\frac{k}{\log k})} \end{aligned}$$

Clearly, all such 4-tuples can be enumerated in polynomial space thereby providing an enumeration of $\mathcal{M}$.

We prove the correctness of the above algorithm by showing that for every feedback vertex set H of T of size at most k, $\mathcal{M}$ contains a set M which satisfies the properties listed in the statement of the lemma. Let H be an arbitrary feedback vertex set of T of size at most k.

For each $j \in [0, ..., \lfloor \log^2 k \rfloor - 1]$, let $H_j := Y_j \cap H$. By averaging, there is a color c such that $0 < |H_c| \leq \frac{k}{\log^2 k}$. For this color c, let $\hat{H} := H_c$. Consider a set $\hat{R}$ obtained as follows: for every vertex $v \in H_c$, pick the first vertex *after* v (if there is any) in $Y \setminus (Y_c \cup H)$ in the topological ordering of $T[Y]$. Note that $T[X \setminus H]$ is acyclic. Color $X \setminus H$ using $\lfloor \log^2 k \rfloor$ colors as was done for Y. Let $\hat{X}$ be the set of all vertices colored 0 in this coloring. The size of any $\hat{X}$-block in $T[X \setminus H]$ is $\log^2 k$. Clearly, $|\hat{X}| \leq \frac{3k}{\log^2 k}$.

The 4-tuple $\langle c, \hat{H}, \hat{R}, \hat{X} \rangle$ described above satisfies all the properties listed in the construction of $\mathcal{M}$. Let $M := (Y_c \setminus H_c) \bigcup \hat{R} \bigcup \hat{X}$. Clearly, $M \cap H = \emptyset$ and $M \in \mathcal{M}$. Since the size of any $[(Y_c \setminus H_c) \bigcup \hat{R}]$-*block* in Y is at most $\log^2 k$, the size of any M-block in $T - H$ is at most $2\log^2 k$. ◀

Lemma 7 gets us quite close to our goal. Indeed, for any feedback vertex set H of size at most k we will find a set M such that the M-blocks in $T - H$ are small. However, the M-blocks of T do not have to be small, because they could contain many vertices from H. The next lemma deals with this problem.

▶ **Definition 8.** Let D be a directed graph. A vertex $v \in V(D)$ is *consistent* with a set $M \subseteq V(D)$ if there are no cycles in $D[M \cup v]$ containing v.

Define a function $\mathcal{I}$ that given a directed graph D and a set $M \subseteq V(D)$ outputs a set of vertices *inconsistent* with M. Define another function $\mathcal{L}$ that given a directed graph D, a set $M \subseteq V(D)$ and an integer k outputs a set of vertices which is the union of all M-blocks of size at least $2\log^4 k$ in $D - \mathcal{I}(D, M)$.

▶ **Lemma 9.** *There exists an algorithm that given a tournament T on $O(k^3)$ vertices where k is an integer outputs a family of set pairs $\mathcal{X} = \{(M_1, P_1), (M_2, P_2), ..., (M_l, P_l)\}$,$|\mathcal{X}| = 2^{O(\frac{k}{\log k})}$ in $2^{O(\frac{k}{\log k})}$ time such that for every feedback vertex set H of size at most k, there exists $(M, P) \in \mathcal{X}$ such that*

1. *$M \cap H = \emptyset$,*
2. *$P \subseteq H$,*
3. *every vertex of $V(T) \setminus P$ is consistent with M, and*
4. *the size of every M-block in $T - P$ is at most $2\log^4 k$.*

Furthermore, $\mathcal{X}$ can be enumerated in polynomial space.

Proof. Use the algorithm of Lemma 7 to compute $\mathcal{M}$. For each $M \in \mathcal{M}$ compute the sets $\mathcal{I}(T, M)$ and $\mathcal{L}(T, M, k)$. For each $B \subseteq \mathcal{L}(T, M, k)$ such that $|B| \leq \frac{2k}{\log^2 k}$ output a pair of sets $(M, P) = (M, \mathcal{I}(T, M) \cup \mathcal{L}(T, M, k) \setminus B)$. The set $\mathcal{X}$ is the collection of all such pair of sets.

We prove that the algorithm satisfies the stated properties. Consider a feedback vertex set H of size at most k. By Lemma 7 there exists $M \in \mathcal{M}$ such that $M \cap H = \emptyset$. Let $C = \mathcal{I}(T, M)$ be the set of vertices that are not consistent with M. These vertices must belong to H. Since for every vertex $v \in T - C$, $T[M \cup v]$ is an acyclic tournament, v can be placed uniquely in the topological ordering of $T[M]$. Hence, for each $v \in T - C$, there is a unique M-block containing it. Since the size of any M-block in $T - H$ is at most $2\log^2 k$, the size of each M-block in $T - C$ will be at most $k + 2\log^2 k$.

An M-block is called *large* if its size is at least $2\log^4 k$. From each *large* M-block at least $2\log^4 k - 2\log^2 k$ vertices belong to H. Hence, in total at most $\frac{k}{2\log^4 k - 2\log^2 k} \times 2\log^2 k \leq \frac{2k}{\log^2 k}$ vertices from the union of *large* M-blocks do not belong to H. Since the algorithm loops over all choices of subsets $B \subseteq \mathcal{L}(T, M, k)$, $|B| \leq \frac{2k}{\log^2 k}$, $\mathcal{X}$ contains a pair (M, P) satisfying the properties listed in the lemma.

Moreover, $|\mathcal{X}|$ is bounded by the product of $|\mathcal{M}|$ and the number of subsets B. Now $|\mathcal{L}(T, M, k)| \leq |V(T)|$ which implies the number of subsets B is at most $(k^3)^{\frac{2k}{\log^2 k}} = 2^{3\log k \times \frac{2k}{\log^2 k}} = 2^{O(\frac{k}{\log k})}$. Hence, $|\mathcal{X}| \leq 2^{O(\frac{k}{\log k})} \times 2^{O(\frac{k}{\log k})} = 2^{O(\frac{k}{\log k})}$ ◀

Observe that the algorithm of Lemma 9 does not store the family $\mathcal{X}$, but enumerates all the pairs $(M, P) \in \mathcal{X}$. Our algorithm for TFVS will go through all pairs in $(M, P) \in \mathcal{X}$ and for each such pair (M, P) search for a feedback vertex set H of size at most k such that (M, P) satisfy the conclusion of Lemma 9 for H. In the next section we shall see that the extra restrictions imposed on H by M and P make it easier to find H.

4 Faster Algorithm for Tournament Feedback Vertex Set

In this section we consider the following problem. We are given as input a tournament T and an integer k, and a pair (M, P) of vertex sets in T. The objective is to find a feedback vertex set H of T of size at most k, such that (M, P) satisfy the conclusion of Lemma 9.

The pair (M, P) naturally leads to a partition of the vertices of $T - (P \cup M)$ into *local subtournaments* corresponding to the induced graphs on the M-blocks in $T - P$. At this point the triangles in $T - P$ can be classified into two types: those that are entirely within a subtournament and those whose vertices are *shared* between more than one subtournament. The goal of our algorithm is to eliminate all the shared triangles. When there are no such triangles left, we can solve the problem independently on each of the subtournaments. Since the subtournaments are small, even brute force search is fast enough.

To formalize our approach it is convenient to define an intermediate problem, and interpret the search for a feedback vertex set H such that (M, P) satisfies the conclusion of Lemma 9 as an instance of this intermediate problem. Let d and t be two positive integers. Consider a class of mixed multigraphs $\mathcal{G}(d,t)$ in which each member is a mixed multigraph $\mathcal{T}$ with the vertex set $V(\mathcal{T})$ partitioned into vertex sets $V_1, V_2, ..., V_t$ such that for each $i \in [t]$, $|V_i| \leq d$ and $T_i := \mathcal{T}[V_i]$ is a *supertournament* and the undirected edge set is $\mathcal{E}(\mathcal{T}) \subseteq \bigcup_{i<j} V_i \times V_j$.

d-Feedback Vertex Cover (d-FVC)
Input: A mixed multigraph $\mathcal{T} \in \mathcal{G}(d,t)$, positive integer k.
Parameter: k
Task: determine whether there exists a set $S \subseteq V(\mathcal{T})$ such that $\|S\| \leq k$ and $\mathcal{T} - S$ contains no undirected edges and is acyclic.

Now we show how TFVS reduces to solving d-feedback vertex cover problem.

▶ **Lemma 10.** *There exists a polynomial time algorithm that given a TFVS instance (T,k) and a subset $M \subseteq V(T)$ outputs a d-FVC instance $(\mathcal{T},k)$ such that T has a feedback vertex set S disjoint from M and $|S| \leq k$ if and only if $(\mathcal{T},k)$ is a* YES*-instance of d-FVC where $d = 2\log^4 k$.*

Proof. We describe an algorithm that reduces T to $\mathcal{T}$ on the same set of vertices as in $T - M$. If $T[M]$ is not acyclic, then output a trivial NO-instance. Otherwise, let $\mathcal{B} := \{B_1, B_2, ..., B_t\}$ be the set of M-blocks in T such that the elements in $\mathcal{B}$ are indexed according to the topological order of $T[M]$. We assume that the topological order of $T[M]$ is such that the edges in $T[M]$ are directed from left to right. Let $V(\mathcal{T}) := V(T) \setminus M$. The directed edge set $E(\mathcal{T})$ is $E(T) \setminus \{e | \forall i,j \in [t], i \neq j \text{ and } e \in B_j \times B_i\}$. The undirected edge set in $\mathcal{T}$ is $\mathcal{E}(\mathcal{T}) := \{\text{undirected}(e) | i,j \in [t], i < j \text{ and } e \in B_j \times B_i\}$ where undirected(e) is an undirected edge between the endpoints of e.

Now we argue about the correctness. Since $\mathcal{T}$ is essentially a subgraph of $T - M$ with some additional undirected edges, we use the same symbol to refer to vertex or directed edge sets in both the instances. Suppose S is a feedback vertex cover of $\mathcal{T}$. Clearly S is disjoint from M. We claim that S is a feedback vertex set of T. The triangles in $T - M$ are of two types: ones whose endpoints lie entirely in B_i for some i and others whose endpoints are shared among multiple M-blocks. Clearly, S hits all the triangles within each subtournament $T[B_i]$ in $\mathcal{T}$. Hence, all that remains to show is that S is also a hitting set for all triangles between different subtournaments $T[B_i]$. For the sake of contradiction suppose that there is a triangle $uvwu$ in $T - M - S$ such that not all of u, v and w belong to the same subtournament of $\mathcal{T}$. Then at least one edge ab in this triangle is such that $a \in B_i, b \in B_j$ and $i > j$. But by the construction of $\mathcal{E}(\mathcal{T})$ there is an undirected edge between a and b implying that at least one of a or b belongs to S, a contradiction.

In the other direction, suppose S is a feedback vertex set of T disjoint from M. Clearly, S hits all the triangles within each subtournament $T[B_i]$ in $\mathcal{T}$. Hence, all that remains to show is that S is a hitting set for $\mathcal{E}(\mathcal{T})$. Suppose not. Then there is an undirected edge $e = uv \in \mathcal{E}$ which is not hit by S. Consider the directed edge in T corresponding to e. Without loss of generality, we can assume that $u \in B_i$ and $v \in B_j$ for some $i, j \in [t]$ such that the directed edge is from u to v and $i > j$. Now in T, there is a vertex $w \in M$ which lies *after* all elements of B_j and *before* all vertices of B_i and forms a triangle $vwuv$. Since, $w \notin S$, either $u \in S$ or $v \in S$, a contradiction. ◀

In light of Lemma 10 we need an efficient algorithm for d-FVC. Next we will give an efficient algorithm for d-FVC and show how it can be used to obtain our claimed algorithm

for TFVS. Our algorithm for d-FVC is based on branching on vertices that appear in at least two edges of $\mathcal{E}(\mathcal{T})$. The case when there are no such vertices has to be handled separately, the algorithm for this case is deferred to Section 6. For now, we simply state the existence of the algorithm for this case, and complete the argument using this algorithm as a black box.

▶ **Lemma 11.** *There exists an algorithm running in* $1.5874^s \cdot 2^{O(d^2+d\log s)} \cdot n^{O(1)}$ *time which finds an optimal feedback vertex cover in a mixed multigraph* $\mathcal{T} \in \mathcal{G}(d,t)$ *in which the undirected edge set* $\mathcal{E}(\mathcal{T})$ *is disjoint and* $|\mathcal{E}(\mathcal{T})| = s$.

The proof of Lemma 11 can be found in Section 6. Armed with Lemma 11 we can give a simple and efficient algorithm for d-FVC. The algorithm is based on branching. In the course of the branching we will sometimes conclude (or guess) that a vertex v is *not* put into the solution S. The operation described below encapsulates the effects of making a vertex undeletable.

In a mixed multigraph D, for any vertex v, D/v is a mixed multigraph obtained by adding a directed edge uw in $D - v$ for every $u \in N^-(v)$ and $w \in N^+(v)$. The next lemma shows that looking for a solution disjoint from v amounts to putting all the undirected neighbors $N_{\mathcal{E}}(v)$ of v into the solution, and finding the optimum solution of $(\mathcal{T} - N_{\mathcal{E}}(v))/v$.

▶ **Lemma 12.** *Let* $(\mathcal{T}, k)$ *be a* d-FVC *instance. If for any vertex* $v \in V(\mathcal{T})$ *it holds that* $N_{\mathcal{E}}(v) = \emptyset$*, then* $(\mathcal{T}, k)$ *has a solution of size at most* k *not containing* v *if and only if* $(\mathcal{T}/v, k)$ *is a* YES*-instance.*

Proof. Let S be a feedback vertex cover of $\mathcal{T}$ of size at most k not containing v. We show that S is a feedback vertex cover of $\mathcal{T}/v$. Clearly, S hits every undirected edge in $\mathcal{T}/v$. For the sake of contradiction suppose there is a cycle of length at most 3 in $\mathcal{T}/v$ not hit by S. If this cycle is in $\mathcal{T} - v$, then it is hit by S. Hence, the triangle must contain an edge not in $\mathcal{T} - v$. Note that $\mathcal{T}/v$ is obtained by adding a directed edge yx for every triangle $xyvx$ in $\mathcal{T} - v$ thereby creating a 2-cycle between x and y. Since $v \notin S$, either $x \in S$ or $y \in S$, a contradiction. For the same reason there are no cycles of length 2 in $\mathcal{T}/v - S$.

Now suppose S is a feedback vertex cover of $\mathcal{T}/v$. Since every cycle in $\mathcal{T} - v$ is a cycle in $\mathcal{T}/v$ which are hit by S, we need to consider cycles in $\mathcal{T}$ containing v. But, for every such cycle $xyvx$ we have a cycle xyx of length 2 in $\mathcal{T}/v$ which is hit by S, we have that $\mathcal{T} - S$ is acyclic. ◀

▶ **Lemma 13.** *There exists an algorithm for* d-FVC *running in* $1.466^n \cdot 2^{O(d^2+d\log n)}$ *time and in* $1.6181^k \cdot 2^{O(d^2+d\log k)} \cdot n^{O(1)}$ *time.*

Proof. We describe a recursive algorithm which searches for a potential solution S of size at most k by branching. For any vertex v, let $N_{\mathcal{E}}(v)$ denote the set of vertices w such that $vw \in \mathcal{E}$. Let $s = |\mathcal{E}|$. As long as there is a vertex v such that $|N_{\mathcal{E}}(v)| \geq 2$ and $k > 0$, the algorithm branches by considering both the possibilities: either $v \in S$ or $v \notin S$. In the branch in which v is picked, n and k are decreased by 1 each and v is removed from the graph. In the other branch, $N_{\mathcal{E}}(v)$ is added to S, and k is decreased by $|N_{\mathcal{E}}(v)|$. At the same time, $N_{\mathcal{E}}(v)$ is removed from the graph. Since $N_{\mathcal{E}}(v) = \emptyset$, by Lemma 12 $(\mathcal{T}, k)$ is reduced to $(\mathcal{T}/v, k)$. Thus the number of vertices is decreased by $|N_{\mathcal{E}}[v]|$. The algorithm stops branching further in a branch in which either $k < 0$ or $k > 0$ and for every vertex v, $|N_{\mathcal{E}}(v)| \leq 1$. In the case that $k < 0$, the algorithm terminates the branch and moves on to other branches. In the other case, if $|\mathcal{E}(T)| > k$, the algorithm terminates that branch, otherwise the algorithm of Lemma 11 is applied. If the size of the optimal solution of Lemma 11 is at most k, then the algorithm outputs YES and terminates, otherwise the algorithm moves to another branch. If the algorithm fails to find any solution of size at most k in every branch, it outputs NO.

Now we do the runtime analysis of the algorithm. At each internal node of the recursion tree, the algorithm spends polynomial time. At a leaf node, either the algorithm terminates or makes a call to the algorithm of Lemma 11 with parameter $s = |\mathcal{E}(\mathcal{T})|$ which is at most k. So, we need to bound the number of times Lemma 11 is called with parameter s for each value of s in $[k]$. Note that for any s, $k \geq s$ with which a call to the algorithm of Lemma 11 is made. Therefore, for each value of $s \in [k]$, the number of calls to the algorithm of Lemma 11 is bounded by the number of nodes in the recursion tree with $k = s$. The recurrence relation for bounding the number of leaves in the recursion tree of the algorithm is given by:

$$f_s(k) \leq f_s(k-1) + f_s(k-2)$$

which solves to $f_s(k) \leq 1.618^{k-s}$ as $f_s(k) \leq 1$ for $k = s$. Hence, the runtime of the algorithm is upper bounded by $\sum_{s=1}^{k} 1.618^{k-s} \times 1.5874^s \cdot 2^{O(d^2 + d\log s)} \cdot n^{O(1)} \leq 1.6181^k \cdot 2^{O(d^2+d\log k)} \cdot n^{O(1)}$.

We can do a similar analysis to bound the runtime in terms of n. Note that in direct correspondence with the fact that whenever k decreases by 1, n decreases by 1 and whenever k decreases by $x \geq 2$, n decreases by $x+1$, we get the following recurrence relation:

$$f_s(n) \leq f_s(n-1) + f_s(n-3)$$

implying $f_s(n) \leq 1.466^{n-s}$ as $f_s(k) \leq 1$ for $n = s$. If s is the size of of the graph, then the largest value of $|\mathcal{E}|$ with which a call to the algorithm of Lemma 11 is made, is at most $\frac{s}{2}$. Hence, the runtime of the algorithm is upper bounded by $\sum_{s=1}^{n} 1.466^{n-s} \times 1.5874^{\frac{s}{2}} \cdot 2^{O(d^2+d\log n)} \cdot n^{O(1)} \leq 1.466^n \cdot 2^{O(d^2+d\log n)} \cdot n^{O(1)}$. ◀

Having shown an efficient algorithm for d-FVC, we are now in position to prove our main theorem.

▶ **Theorem 14.** *There exists an algorithm for TFVS running in $O(1.466^n)$ time and in $O(1.6181^k + n^{O(1)})$ time.*

Proof. The algorithm begins by running the kernelization algorithm of Lemma 3 for the given TFVS instance. In the remainder we assume that $n = O(k^3)$. Next the algorithm proceeds to apply Lemma 9 to create a family of set pairs $\mathcal{X}$. For each set pair $(M, P) \in \mathcal{X}$ it determines whether there is a feedback vertex set H of size at most k such that $H \cap M = \emptyset$ and $P \subseteq H$ as follows:

First it runs the algorithm of Lemma 10 with input $(T - P, k - |P|)$ and M to reduce the problem to an equivalent d-FVC instance which is then passed to the algorithm of Lemma 13 as input. The algorithm outputs YES and terminates if the output of the algorithm of Lemma 13 is YES. If no solution of size at most k is obtained for any set pair in $\mathcal{X}$, the algorithm outputs NO and terminates.

The correctness of the algorithm follows from Lemma 9 and Lemma 10. The running time of the algorithm is upper bounded by $|\mathcal{X}|$ times (the runtime of the algorithm of Lemma 13). Since $|\mathcal{X}| = 2^{o(k)}$ and the bulk of the algorithm is run on a tournament with at most $O(k^3)$ vertices the total time used by the algorithm is upper bounded by $O(1.466^n)$ and $O(1.6181^k + n^{O(1)})$. ◀

We have now proved our main result, assuming the correctness of Lemma 11. The remainder of the paper is devoted to proving Lemma 11. The engine of the algorithm of Lemma 11 is a new graph partitioning theorem. The next section contains the statement and proof of this theorem, while Section 6 wraps up the proof of Lemma 11, thereby completing the proof of Theorem 14.

5 Balanced Edge Partition Theorem

Given an undirected graph $G, |E(G)| = m$, if each vertex in $V(G)$ is colored *red* or *blue* uniformly at random, then in expectation there will be $\frac{m}{4}$ *red* edges and $\frac{m}{4}$ *blue* edges, where a red edge is an edge whose both endpoints are red and a blue edge is an edge whose both endpoints are blue. Using Chebyshev inequality it can be shown that, with high probability, the number of red or blue edges will be within $O(\sqrt{md})$ of $\frac{m}{4}$ where d is the maximum degree of a vertex in the graph. A proof of this fact is skipped in favor of a local search algorithm which runs in polynomial time and provides a coloring with smaller deviation from the expected value than random coloring.

▶ **Theorem 15.** *Given an undirected multigraph without self-loops and isolated vertices G of maximum degree at most d and $|E(G)| = m$, there exists a partition (A, B) of $V(G)$ such that*

- $\frac{m}{4} - \frac{d}{2} \leq |E(G[A])| \leq \frac{m}{4} + \frac{d}{2}$,
- $\frac{m}{4} - \frac{d}{2} \leq |E(G[B])| \leq \frac{m}{4} + \frac{d}{2}$, *and*
- $\frac{m}{2} - d \leq |E(G[A, B])| \leq \frac{m}{2} + d$

where $E(G[A, B])$ is the set of edges with one endpoint in A and other in B. Furthermore, there is a polynomial time algorithm to obtain this partition.

Proof. The following local search algorithm is used to obtain the desired partition:

At each step, the algorithm maintains a partition (A, B) of $V(G)$. As long as there exists a vertex $v \in A$ (or $v \in B$) such that moving it to other part decreases the measure $\mu = ||E(G[A])| - \frac{m}{4}| + ||E(G[B])| - \frac{m}{4}|$, the algorithm changes the partition to $(A \setminus v, B \cup v)$ (or $(A \cup v, B \setminus v)$). The algorithm terminates if no vertex can be moved. Since $\mu \leq m$ and in each step, it decreases by at least one, above algorithm terminates in polynomial time.

Correctness: Let $m_A := |E(G[A])|, m_B := |E(G[B])|$, and $m_C := |E(G[A, B])|$. Let $x := m_A - \frac{m}{4}$ and $y := m_B - \frac{m}{4}$ when the algorithm terminates. Then, $\mu = |x| + |y|$. For any vertex v, let a_v denote the number of edges incident on v whose other endpoints are in A and b_v denote the number of edges incident on v whose other endpoints are in B. Clearly, for every vertex v, $a_v + b_v \leq d$. Suppose that a vertex $v \in A$ is moved to B. The new partition is $(A', B') = (A \setminus v, B \cup v)$. Then, $m_{A'} = \frac{m}{4} + x - a_v, m_{B'} = \frac{m}{4} + y + b_v$ and the measure at this partition is $\mu' = |x - a_v| + |y + b_v|$. Define $\delta_A^v := \mu' - \mu = |x - a_v| - |x| + |y + b_v| - |y|$. Similarly, if a vertex $v \in B$ moves to A creating new partition $(A', B') = (A \cup v, B \setminus v)$, we can define $\delta_B^v := \mu' - \mu = |x + a_v| - |x| + |y - b_v| - |y|$. Note that since the algorithm has terminated, for any vertex $v \in V(G)$, $\delta_A^v \geq 0$ and $\delta_B^v \geq 0$. Then, the claim of the theorem is that $|x| \leq \frac{d}{2}$ and $|y| \leq \frac{d}{2}$. For the sake of contradiction assume the following possible values of x and y:

$\boldsymbol{x > \frac{d}{2}, y > \frac{d}{2}}$: Consider moving a vertex $v \in A$ to B. Then, $\delta_A^v = |x - a_v| - |x| + b_v$. Suppose that $x < a_v, \delta_A^v = a_v - x - x + b_v = a_v + b_v - 2x$. But, for every vertex $v \in V$, $a_v + b_v \leq d$ which implies $\delta_A^v < 0$, a contradiction.
Hence, $\forall v \in A$, $x \geq a_v, \delta_A^v = x - a_v - x + b_v = b_v - a_v$. If $v \in A$ is such that $a_v > b_v$, then $\delta_A^v < 0$, a contradiction. Hence, $\forall v \in A, a_v \leq b_v$. Then, $\sum_{v \in A} a_v \leq \sum_{v \in A} b_v \implies 2m_A \leq m_C \implies m_C \geq \frac{m}{2} + 2x > \frac{m}{2} + d$ which contradicts $m = m_A + m_B + m_C$.

$\boldsymbol{x < -\frac{d}{2}, y < -\frac{d}{2}}$: Consider moving a vertex $v \in A$ to B. Then, $\delta_A^v = |y + b_v| - |y| + a_v$. Suppose that $|y| < b_v, \delta_A^v = b_v - |y| - |y| + a_v = a_v + b_v - 2|y|$. But, for all vertices v, $a_v + b_v \leq d$ which implies that $a_v - 2|y| + b_v < 0$, i.e. $\delta_A^v < 0$, a contradiction.
Hence, for every vertex $v \in A$, we have that $|y| \geq b_v$ and therefore, $\delta_A^v = |y| - b_v - |y| + a_v = a_v - b_v$. If $v \in A$ is such that $a_v < b_v$, then $\delta_A^v < 0$, a contradiction. Hence, for all

vertices $v \in A, a_v \geq b_v$. This implies that $\sum_{v \in A} a_v \geq \sum_{v \in A} b_v \implies 2m_A \geq m_C \implies m_C \leq \frac{m}{2} + 2x < \frac{m}{2} - d$ which is a contradiction.

$x > \frac{d}{2}, y < -\frac{d}{2}$: Consider moving a vertex $v \in A$ to B. Then, $\delta_A^v := \mu' - \mu = |x - a_v| - |x| + |y + b_v| - |y| < 0$ as $|x - a_v| - |x| \leq 0$ and $|y + b_v| - |y| \leq 0$ and at least one of the inequalities is strict, hence a contradiction.

$y > \frac{d}{2}, x < -\frac{d}{2}$: Similar to the previous case.

$x > \frac{d}{2}, |y| \leq \frac{d}{2}$: Consider moving a vertex $v \in A$ to B. Suppose that $x < a_v$, then $\delta_A^v = a_v - 2x + |y + b_v| - |y| \leq a_v - 2x + b_v < 0$, a contradiction.
Hence, for every vertex $v \in A$, $x \geq a_v$, then $\delta_A^v = |y+b_v|-|y|-a_v \leq b_v - a_v$. If $v \in A$ is such that $a_v > b_v$, then $\delta_A^v < 0$, a contradiction. Hence, for every vertex $v \in A$, we have that $a_v \leq b_v$. This implies that $\sum_{v \in A} a_v \leq \sum_{v \in A} b_v \implies 2m_A \leq m_C \implies m_C \geq \frac{m}{2} + 2x > \frac{m}{2} + d$ which contradicts $m = m_A + m_B + m_C$.

$y > \frac{d}{2}, |x| \leq \frac{d}{2}$: Similar to the previous case.

$x < -\frac{d}{2}, |y| \leq \frac{d}{2}$: Consider moving a vertex $v \in B$ to A. If $|x| \geq a_v$, then $\delta_B^v = a_v - 2|x| + |y - b_v| - |y| \leq a_v - 2|x| + b_v < 0$, a contradiction. So, for every vertex $v \in B$, $|x| < a_v$ and $0 \leq \delta_B^v = |x| - a_v - |x| + |y - b_v| - |y| \leq -a_v + b_v$. Hence, for each vertex $v \in B$, $a_v \leq b_v$. This implies $\sum_{v \in B} a_v \leq \sum_{v \in B} b_v \implies m_C \leq 2m_B \implies m_C \leq \frac{m}{2} + 2y < \frac{m}{2}$ which contradicts $m = m_A + m_B + m_C$.

$y < -\frac{d}{2}, |x| \leq \frac{d}{2}$: Similar to the previous case.

Hence, $|x| \leq \frac{d}{2}$ and $|y| \leq \frac{d}{2}$, and thus $\frac{s}{2} - d \leq m_C \leq \frac{s}{2} + d$. This concludes the proof. ◀

6 d-Feedback Vertex Cover with Undirected Degree at Most One

Now that we are equipped with Theorem 15, we are almost ready to prove Lemma 11. First we show a lemma that encapsulates the use of Theorem 15 inside the algorithm of Lemma 11.

▶ **Lemma 16.** *There exists a polynomial time algorithm that given a mixed multigraph $\mathcal{T} \in \mathcal{G}(d, t)$ with disjoint undirected edge set $\mathcal{E}(\mathcal{T})$ outputs a partition (X, Y) of $V(\mathcal{T})$ such that there are no directed edge with one endpoint in X and the other in Y and*

- $||\mathcal{E}(X) \cap \mathcal{E}| - \frac{s}{4}| \leq \frac{d}{2}$,
- $||\mathcal{E}(Y) \cap \mathcal{E}| - \frac{s}{4}| \leq \frac{d}{2}$ *and*
- $||\mathcal{E}(X, Y) \cap \mathcal{E}| - \frac{s}{2}| \leq d$

where $s = |\mathcal{E}(\mathcal{T})|$ and $\mathcal{E}(X)$ is the set of undirected edges in $\mathcal{T}[X]$, $\mathcal{E}(Y)$ is the set of undirected edges in $\mathcal{T}[Y]$ and $\mathcal{E}(X, Y)$ is the set of undirected edges with one endpoint in X and other in Y.

Proof. Construct an undirected, multigraph Z such that $V(Z) = \{z_i | i \in [t]\}$ and $E(Z) = \{z_i z_j | uv \in \mathcal{E}, u \in V_i, v \in V_j\}$. Run the algorithm of Theorem 15 to get the partition (A, B) of $V(Z)$. Output $X := \bigcup_{i, z_i \in A} V_i$ and $Y := \bigcup_{i, z_i \in B} V_i$. Since $\mathcal{E}$ is disjoint and for each $i \in [t]$, $|V_i| \leq d$, the maximum degree of a vertex in Z is at most d. Hence, the correctness of the algorithm and the size bound in the lemma follows from Theorem 15. ◀

We are now ready to prove Lemma 11. For convenience we re-state it here.

▶ **Lemma** (Lemma 11). *There exists an algorithm running in $1.5874^s \cdot 2^{O(d^2 + d \log s)} \cdot n^{O(1)}$ time which finds an optimal feedback vertex cover in a mixed multigraph $\mathcal{T} \in \mathcal{G}(d, t)$ in which the undirected edge set $\mathcal{E}(\mathcal{T})$ is disjoint and $|\mathcal{E}(\mathcal{T})| = s$.*

Proof. The algorithm maintains a set S which is initialized to the empty set $\emptyset$. If the underlying undirected graph of $\mathcal{T}$ is disconnected, then the algorithm solves each connected component independently and outputs S as the union of sets returned for each component. If $s \leq d$, then S is an optimal solution set obtained by a brute force search in the instance. If $s > d$, the algorithm obtains a partition (X, Y) of $V(\mathcal{T})$ by running the algorithm of Lemma 16. Then, it loops over all subsets $C \subseteq \mathcal{E}(X, Y)$, calling itself recursively on $\mathcal{T}\big[V(\mathcal{T}) \setminus (V_X(C) \cup V_Y(\mathcal{E}(X, Y) \setminus C))\big]$ and computes $S_C := V_X(C) \cup V_Y(\mathcal{E}(X, Y) \setminus C) \cup S'$ where S' is the set returned at the recursive call. Finally, the algorithm outputs the smallest set S_C over all choices of $C \subseteq \mathcal{E}(X, Y)$.

Now to argue about the correctness of the algorithm, we use induction on $|\mathcal{E}(\mathcal{T})|$. In the base case $|\mathcal{E}(\mathcal{T})| \leq d$, S is an optimal feedback vertex cover. As the induction hypothesis, suppose that the algorithm outputs an optimal solution for $d < |\mathcal{E}(\mathcal{T})| < s$. Consider $|\mathcal{E}(\mathcal{T})| = s$. Note that for any $C \subseteq \mathcal{E}(X, Y)$, S_C is a d-feedback vertex cover as $V_X(C) \cup V_Y(\mathcal{E}(X, Y) \setminus C)$ is a hitting set for $\mathcal{E}(X, Y)$ and by the induction hypothesis, S' is an optimal solution for $\mathcal{T}[V(\mathcal{T}) \setminus (V_X(C) \cup V_Y(\mathcal{E}(X, Y) \setminus C))]$. At the same time, for any $C \subseteq \mathcal{E}(X, Y)$, $|V_X(C) \cup V_Y(\mathcal{E}(X, Y) \setminus C)| = |\mathcal{E}(X, Y)|$ which is the size of the smallest hitting set for $\mathcal{E}(X, Y)$. Let S_o be an optimal solution and $C_o := \mathcal{E}(S_o \cap X, Y \setminus S_o)$. Then, we claim that $|S_{C_o}| = |S_o|$. Clearly, $|S_{C_o}| \geq |S_o|$. Now, $S_o \setminus V_X(C_o) \cup V_Y(\mathcal{E}(X, Y) \setminus C_o)$ is a d-feedback vertex cover for $\mathcal{T}[V(\mathcal{T}) \setminus (V_X(C_o) \cup V_Y(\mathcal{E}(X, Y) \setminus C_o))]$. Therefore, $|S'| \leq |S_o \setminus (V_X(C_o) \cup V_Y(\mathcal{E}(X, Y) \setminus C_o)| = |S_o| - |\mathcal{E}(X, Y)| \implies |S_{C_o}| \leq |S_o|$, thus proving the claim.

Now we proceed to the runtime analysis of the algorithm. Let $h(s, d)$ be the maximum number of leaves in the recursion tree of the algorithm when run on an input with parameters s and d. Since in each recursive call, s decreases by at least 1, the depth of the recursion tree is at most s. In each internal node of the recursion tree, the algorithm spends polynomial time in size of the input and in each leaf, it spends at most $2^{O(d^2)}$ time as the total number of vertices in each connected component of $\mathcal{T}$ is $O(d^2)$. Thus, the runtime of the algorithm on any input with parameters s and d is upper bounded by $h(s, d) \times 2^{O(d^2)} \times n^{O(1)}$. To upper bound $h(s, d)$, first note that $h(a, d) + h(b, d) \leq h(a + b, d)$ because $h(a, d)$ and $h(b, d)$ represent the number of leaves of two independent subtrees. Now for each $C \subseteq \mathcal{E}(X, Y)$, in $\mathcal{T}[V(\mathcal{T}) \setminus (V_X(C) \cup V_Y(\mathcal{E}(X, Y) \setminus C))]$, the undirected edge set $\mathcal{E}(X, Y) = \emptyset$. Hence, the algorithm effectively solves $\mathcal{T}[V(\mathcal{T}) \setminus (V_X(C)]$ and $\mathcal{T}[V(\mathcal{T}) \setminus V_Y(\mathcal{E}(X, Y) \setminus C)]$ independently where by Lemma 16, the number of undirected edges is at most $\frac{s}{4} + \frac{d}{2}$ for each instance. Again by Lemma 16, $|\mathcal{E}(X, Y)| \leq \frac{s}{2} + d$. Hence, the number of choices for $C \subseteq \mathcal{E}(X, Y)$ is at most $2^{\frac{s}{2}+d}$. As we have seen for each C, the algorithm calls itself twice on graphs with the undirected edge set size at most $\frac{s}{4} + \frac{d}{2}$. So in total, the algorithm makes $2^{\frac{s}{2}+d+1}$ recursive calls with *parameter* $\frac{s}{4} + \frac{d}{2}$. Thus $h(s, d)$ is upper bounded by the recurrence relation $h(s, d) \leq 2^{1+\frac{s}{2}+d} h(\frac{s}{4} + \frac{d}{2}, d)$ which solves to $h(s, d) = 1.5874^s \cdot 2^{O(d \log s)}$. Hence, the runtime of the algorithm is bounded by $1.5874^s \cdot 2^{O(d \log s)} \times 2^{O(d^2)} \times n^{O(1)} = 1.5874^s \cdot 2^{O(d^2 + d \log s)} \cdot n^{O(1)}$. ◀

The proof of Lemma 11 completes the proof of our main result, an algorithm for TFVS with running time upper bounded by $O(1.466^n)$ and by $O(1.6181^k + n^{O(1)})$.

Acknowledgements. 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. 306992 and the Beating Hardness by Preprocessing grant funded by the Bergen Research Foundation.

References

1 Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discrete Math.*, 12(3):289–297, 1999.

2 B. Bollobás and A. D. Scott. Judicious partitions of bounded-degree graphs. *J. Graph Theory*, 46(2):131–143, June 2004. doi:10.1002/jgt.v46:2.

3 Mao-cheng Cai, Xiaotie Deng, and Wenan Zang. An approximation algorithm for feedback vertex sets in tournaments. *SIAM J. Comput.*, 30(6):1993–2007, 2000.

4 Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5), 2008.

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

6 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 *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159, 2011.

7 Michael Dom, Jiong Guo, Falk Hüffner, Rolf Niedermeier, and Anke Truß. Fixed-parameter tractability results for feedback set problems in tournaments. In *Algorithms and Complexity, 6th Italian Conference, CIAC 2006, Rome, Italy, May 29-31, 2006, Proceedings*, pages 320–331, 2006.

8 Michael Dom, Jiong Guo, Falk Hüffner, Rolf Niedermeier, and Anke Truß. Fixed-parameter tractability results for feedback set problems in tournaments. *J. Discrete Algorithms*, 8(1):76–86, 2010.

9 P Erdős and L Pósa. On independent circuits contained in a graph. *Canad. J. Math*, 17:347–352, 1965.

10 Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

11 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010.

12 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences. W. H. Freeman and Co., 1979.

13 Serge Gaspers and Matthias Mnich. Feedback vertex sets in tournaments. *Journal of Graph Theory*, 72(1):72–89, 2013.

14 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Inf. Process. Lett.*, 114(10):556–560, 2014.

15 Venkatesh Raman and Saket Saurabh. Parameterized algorithms for feedback set problems and their duals in tournaments. *Theor. Comput. Sci.*, 351(3):446–458, 2006.

16 Igor Razgon. Computing minimum directed feedback vertex set in $o(1.9977^n)$. In *Theoretical Computer Science, 10th Italian Conference, ICTCS 2007, Rome, Italy, October 3-5, 2007, Proceedings*, pages 70–81, 2007.

17 Bruce Reed, Neil Robertson, Paul Seymour, and Robin Thomas. Packing directed circuits. *Combinatorica*, 16(4):535–554, 1996.

18 Mingyu Xiao and Hiroshi Nagamochi. An improved exact algorithm for undirected feedback vertex set. *J. Comb. Optim.*, 30(2):214–241, 2015.

Knapsack in Graph Groups, HNN-Extensions and Amalgamated Products

Markus Lohrey[1] **and Georg Zetzsche**[*2]

1 **Universität Siegen, Germany**
lohrey@eti.uni-siegen.de
2 **LSV, CNRS & ENS Cachan, Université Paris-Saclay, France**
zetzsche@lsv.fr

Abstract

It is shown that the knapsack problem, which was introduced by Myasnikov et al. for arbitrary finitely generated groups, can be solved in NP for graph groups. This result even holds if the group elements are represented in a compressed form by SLPs, which generalizes the classical NP-completeness result of the integer knapsack problem. We also prove general transfer results: NP-membership of the knapsack problem is passed on to finite extensions, HNN-extensions over finite associated subgroups, and amalgamated products with finite identified subgroups.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Graph groups, HNN-extensions, amalgamated products, knapsack

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.50

1 Introduction

In their paper [36], Myasnikov, Nikolaev, and Ushakov started the investigation of classical discrete optimization problems, which are classically formulated over the integers, for arbitrary in general non-commutative groups. Among other problems, they introduced for a finitely generated group G the *knapsack problem* and the *subset sum problem*. The input for the knapsack problem is a sequence of group elements $g_1, \ldots, g_k, g \in G$ (specified by finite words over the generators of G) and it is asked whether there exists a solution $(x_1, \ldots, x_k) \in \mathbb{N}^k$ of the equation $g_1^{x_1} \cdots g_k^{x_k} = g$. For the subset sum problem one restricts the solution to $\{0,1\}^k$. For the particular case $G = \mathbb{Z}$ (where the additive notation $x_1 \cdot g_1 + \cdots + x_k \cdot g_k = g$ is usually preferred) these problems are NP-complete if the numbers $g_1, \ldots, g_k, g$ are encoded in binary representation. For subset sum, this is a classical result from Karp's seminal paper [24] on NP-completeness. Knapsack for integers is usually formulated in a more general form in the literature; NP-completeness of the above form (for binary encoded integers) was shown in [17], where the problem was called MULTISUBSET SUM.[1] Interestingly, if we consider subset sum for the group $G = \mathbb{Z}$, but encode the input numbers $g_1, \ldots, g_k, g$ in unary notation, then the problem is in DLOGTIME-uniform TC^0 (a small subclass of polynomial time and even of logarithmic space that captures the complexity of multiplication of binary encoded numbers) [14], and the same holds for knapsack, since the instance $x_1 \cdot g_1 + \cdots + x_k \cdot g_k = g$ has a

* This author is supported by a fellowship within the Postdoc-Program of the German Academic Exchange Service (DAAD).

[1] Note that if we ask for a solution $(x_1, \ldots, x_k)$ in $\mathbb{Z}^k$, then knapsack can be solved in polynomial time (even for binary encoded integers) by checking whether $\gcd(g_1, \ldots, g_k)$ divides g.

© Markus Lohrey and Georg Zetzsche;
licensed under Creative Commons License CC-BY

solution if and only if it has a solution with $x_i \leq k \cdot (\max\{g_1, \ldots, g_k, g\})^3$ [37]. This allows to reduce unary knapsack to unary subset sum. See [21] for related results.

In [36] the authors encode elements of the finitely generated group G by words over the group generators and their inverses. For $G = \mathbb{Z}$ this representation corresponds to the unary encoding of integers. Among others, the following results were shown in [36]:

- Subset sum and knapsack can be solved in polynomial time for every hyperbolic group.
- Subset sum for a virtually nilpotent group (a finite extension of a nilpotent group) can be solved in polynomial time.
- For the following groups, subset sum is NP-complete (whereas the word problem can be solved in polynomial time): free metabelian non-abelian groups of finite rank, the wreath product $\mathbb{Z} \wr \mathbb{Z}$, Thompson's group F, and the Baumslag-Solitar group $\mathrm{BS}(1, 2)$.

Further results on knapsack and subset sum have been recently obtained in [27]:

- For a virtually nilpotent group, subset sum belongs to NL (nondeterministic logspace).
- There is a nilpotent group of class 2 (in fact, a direct product of sufficiently many copies of the discrete Heisenberg group $H_3(\mathbb{Z})$), for which knapsack is undecidable.
- The knapsack problem for the discrete Heisenberg group $H_3(\mathbb{Z})$ is decidable. In particular, together with the previous point it follows that decidability of knapsack is not preserved under direct products.
- There is a polycyclic group with an NP-complete subset sum problem.
- The knapsack problem is decidable for every co-context-free group.

The focus of this paper will be on the knapsack problem. We will prove that this problem can be solved in NP for every *graph group*. Graph groups are also known as right-angled Artin groups or free partially commutative groups. A graph group is specified by a finite simple graph. The vertices are the generators of the group, and two generators a and b are allowed to commute if and only if a and b are adjacent. Graph groups somehow interpolate between free groups and free abelian groups and can be seen as a group counterpart of trace monoids (free partially commutative monoids), which have been used for the specification of concurrent behavior. In combinatorial group theory, graph groups are currently an active area of research, mainly because of their rich subgroup structure (see e.g. [5, 8, 16]).

To prove that knapsack belongs to NP for a graph group, we proceed in two steps: We first show that if an instance $g_1^{x_1} \cdots g_k^{x_k} = g$ has a solution in a graph group, then it has a solution where every x_i is bounded exponentially in the input length (the total length of all words representing the group elements $g_1, \ldots, g_k, g$). We then guess the binary encodings of numbers $n_1, \ldots, n_k$ that are bounded by the exponential bound from the previous point and verify in polynomial time the identity $g_1^{n_1} \cdots g_k^{n_k} = g$. The latter problem is an instance of the so-called *compressed word problem* for a graph group. This is the classical word problem, where the input group element is given succinctly by a so-called *straight-line program* (SLP), which is a context-free grammar that produces a single word (here, a word over the group generators and their inverses). An SLP with n productions in Chomsky normal form can produce a string of length 2^n. Nevertheless, the compressed word problem for a fixed graph group can be solved in polynomial time (see [29] for details).

In fact, our proof yields a stronger result: First, it yields an NP procedure for solving knapsack-like equations $h_0 g_1^{x_1} h_1 \cdots h_{k-1} g_k^{x_k} h_k = 1$ where some of the variables $x_1, \ldots, x_k$ are allowed to be identical. We call such an equation an *exponent equation*. Hence, we prove that solvability of exponent equations over a graph group belongs to NP.

Second, we show that the latter result even holds when the group elements $g_1, \ldots, g_k$ and $h_0, \ldots, h_k$ are given succinctly by SLPs; we speak of *solvability of compressed exponent equations*. This is interesting since the SLP-encoding of group elements corresponds in the

case $G = \mathbb{Z}$ to the binary encoding of integers. Hence, membership in NP for solvability of compressed exponent equations over a graph group generalizes the classical NP-membership for knapsack (over $\mathbb{Z}$) to a much wider class of groups.

Furthermore, we extend the class of groups for which solvability of knapsack (resp. compressed exponent equations) is in NP by proving general transfer results. Our first transfer result states that if H is a finite extension of G and solvability of compressed exponent equations (or knapsack) is in NP for G, then the same holds for H. This provides such algorithms for the abundant class of *virtually special groups*. These are finite extensions of subgroups of graph groups. Virtually special groups recently played a major role in a spectacular breakthrough in three-dimensional topology, namely the solution of the virtual Haken conjecture [1]. In the course of this development it turned out that the class of virtually special groups is very rich: It contains Coxeter groups [18], one-relator groups with torsion [41], fully residually free groups [41], and fundamental groups of hyperbolic 3-manifolds [1].

We also prove transfer results for HNN-extensions and amalgamated products with finite associated (resp. identified) subgroups in the case of the knapsack problem. These two constructions are of fundamental importance in combinatorial group theory [34]. Examples include Stallings' decomposition of groups with infinitely many ends [38] or the construction of virtually free groups [9]. Moreover, these constructions are known to preserve a wide range of important structural and algorithmic properties [2, 6, 19, 22, 23, 25, 26, 30, 31, 35].

A side product of our proof is that the set of all solutions $(x_1, \ldots, x_k) \in \mathbb{N}^k$ of an exponent equation $g_1^{x_1} \cdots g_k^{x_k} = g$ over a graph group is semilinear, and a semilinear representation can be produced effectively. This seems to be true for many groups, e.g., for all co-context-free groups [27]. On the other hand, for the discrete Heisenberg group $H_3(\mathbb{Z})$ solvability of exponent equations is decidable, but the set of all solutions of an exponent equation is not semilinear; it is defined by a single quadratic Diophantine equation [27].

Finally, we complement our upper bounds with a new lower bound: Knapsack and subset sum are both NP-complete for a direct product of two free groups of rank two ($F_2 \times F_2$). This group is the graph group corresponding to a cycle of length four. NP-hardness already holds for the case that the input group elements are specified by words over the generators (for SLP-compressed words, NP-hardness already holds for $\mathbb{Z}$) and the exponent variables are allowed to take values in $\mathbb{Z}$ (instead of $\mathbb{N}$). NP-completeness of subset sum for $F_2 \times F_2$ solves an open problem from [15].

A full version of this work can be found in the arXiv [33].

Related work. The knapsack problem is a special case of the more general *rational subset membership problem*. A rational subset of a finitely generated monoid M is the homomorphic image in M of a regular language over the generators of M. In the rational subset membership problem for M the input consists of a rational subset $L \subseteq M$ (specified by a finite automaton) and an element $m \in M$ and it is asked whether $m \in L$. It was shown in [32] that the rational subset membership problem for a graph group G is decidable if and only if the corresponding graph has (i) no induced cycle on four nodes (C4) and (ii) no induced path on four nodes (P4). For the decidable cases, the precise complexity is open.

Knapsack for G can be also viewed as the question, whether a word equation $z_1 z_2 \cdots z_n = 1$, where $z_1, \ldots, z_n$ are variables, together with constraints of the form $\{g^n \mid n \geq 0\}$ for the variables has a solution in G. Such a solution is a mapping $\varphi\colon \{z_1, \ldots, z_n\} \to G$ such that $\varphi(z_1 z_2 \cdots z_n)$ evaluates to 1 in G and all constraints are satisfied. For another class of constraints (so-called normalized rational constraints, which do not cover constraints of

the form $\{g^n \mid n \geq 0\}$), solvability of general word equations was shown to be decidable (PSPACE-complete) for graph groups by Diekert and Muscholl [13]. This result was extended in [12] to a transfer theorem for graph products. A graph product is specified by a finite simple graph where every node is labeled with a group. The associated group is obtained from the free product of all vertex groups by allowing elements from adjacent groups to commute. Note that decidability of knapsack is not preserved under graph products: It is not even preserved under direct products (see the above mentioned results from [27]).

2 Words and Straight-Line Programs

For a word w we denote with $\mathrm{alph}(w)$ the set of symbols occurring in w. The length of the word w is $|w|$. A *straight-line program*, briefly *SLP*, is basically a context-free grammar that produces exactly one string. To ensure this, the grammar has to be acyclic and deterministic (every variable has a unique production where it occurs on the left-hand side). Formally, an SLP is a tuple $\mathcal{G} = (V, \Sigma, \mathrm{rhs}, S)$, where V is a finite set of *variables* (or *nonterminals*), Σ is the *terminal alphabet*, $S \in V$ is the *start variable*, and rhs maps every variable to a *right-hand side* $\mathrm{rhs}(A) \in (V \cup \Sigma)^*$. We require that there is a linear order $<$ on V such that $B < A$ whenever $B \in N \cap \mathrm{alph}(\mathrm{rhs}(A))$. Every variable $A \in V$ derives to a unique string $\mathrm{val}_{\mathcal{G}}(A)$ by iteratively replacing variables by the corresponding right-hand sides, starting with A. Finally, the string *derived by* $\mathcal{G}$ is $\mathrm{val}(\mathcal{G}) = \mathrm{val}_{\mathcal{G}}(S)$.

Let $\mathcal{G} = (V, \Sigma, \mathrm{rhs}, S)$ be an SLP. The *size* of $\mathcal{G}$ is $|\mathcal{G}| = \sum_{A \in V} |\mathrm{rhs}(A)|$, i.e., the total length of all right-hand sides. A simple induction shows that for every SLP $\mathcal{G}$ of size m one has $|\mathrm{val}(\mathcal{G})| \leq \mathcal{O}(3^{m/3}) \subseteq 2^{\mathcal{O}(m)}$ [7, proof of Lemma 1]. On the other hand, it is straightforward to define an SLP $\mathcal{H}$ of size $2n$ such that $|\mathrm{val}(\mathcal{H})| \geq 2^n$. This justifies to see an SLP $\mathcal{G}$ as a compressed representation of the string $\mathrm{val}(\mathcal{G})$, and exponential compression rates can be achieved in this way. More details on SLPs can be found in [29].

3 Knapsack and Exponent Equations

We assume that the reader has some basic knowledge concerning (finitely generated) groups (see e.g. [34] for further details). Let G be a finitely generated group, and let A be a finite generating set for G. Then, elements of G can be represented by finite words over the alphabet $A^{\pm 1} = A \cup A^{-1}$. An *exponent equation* over G is an equation of the form

$$v_0 u_1^{x_1} v_1 u_2^{x_2} v_2 \cdots u_n^{x_n} v_n = 1$$

where $u_1, u_2, \ldots, u_n, v_0, v_1, \ldots, v_n \in G$ are group elements that are given by finite words over the alphabet $A^{\pm 1}$ and $x_1, x_2, \ldots, x_n$ are not necessarily distinct variables. Such an exponent equation is *solvable* if there exists a mapping $\sigma\colon \{x_1, \ldots, x_n\} \to \mathbb{N}$ such that $v_0 u_1^{\sigma(x_1)} v_1 u_1^{\sigma(x_2)} v_2 \cdots u_n^{\sigma(x_n)} v_n = 1$ in the group G. *Solvability of exponent equations over* G is the following computational problem:

Input: An exponent equation E over G (where elements of G are specified by words over the group generators and their inverses).
Question: Is E solvable?

The *knapsack problem* for the group G is the restriction of solvability of exponent equations over G to exponent equations of the form $u_1^{x_1} u_2^{x_2} \cdots u_n^{x_n} u^{-1} = 1$ or, equivalently, $u_1^{x_1} u_2^{x_2} \cdots u_n^{x_n} = u$ where the exponent variables $x_1, \ldots, x_n$ have to be pairwise different.

We will also study a compressed version of exponent equations over G, where elements of G are given by SLPs over $A^{\pm 1}$. A *compressed exponent equation* is an exponent equation

$v_0 u_1^{x_1} v_1 u_2^{x_2} v_2 \cdots u_n^{x_n} v_n = 1$ where the group elements $u_1, u_2, \ldots, u_n, v_0, v_1, \ldots, v_n \in G$ are given by SLPs over the terminal alphabet $A^{\pm 1}$. The sum of the sizes of these SLPs is the size of the compressed exponent equation. Let us define *solvability of compressed exponent equations over* G as the following computational problem:

Input: A compressed exponent equation E over G.
Question: Is E solvable?

The *compressed knapsack problem* for G is defined analogously. Note that with this terminology, the classical knapsack problem for binary encoded integers is the compressed knapsack problem for the group $\mathbb{Z}$. The binary encoding of an integer can be easily transformed into an SLP over the alphabet $\{a, a^{-1}\}$ (where a is a generator of $\mathbb{Z}$) and vice versa. Here, the number of bits in the binary encoding and the size of the SLP are linearly related.

It is a simple observation that the decidability and complexity of solvability of (compressed) exponent equations over G as well as the (compressed) knapsack problem for G does not depend on the chosen finite generating set for the group G. Therefore, we do not have to mention the generating set explicitly in these problems.

▶ Remark 1. Since we are dealing with a group, one might also allow solution mappings $\sigma\colon \{x_1, \ldots, x_n\} \to \mathbb{Z}$ to the integers. But this variant of solvability of (compressed) exponent equations (knapsack, respectively) can be reduced to the above version, where σ maps to $\mathbb{N}$, by simply replacing a power $u_i^{x_i}$ by $u_i^{x_i}(u_i^{-1})^{y_i}$, where y_i is a fresh variable.

The goal of this paper is to prove the decidability of solvability of exponent equations for so-called graph groups. We actually prove that solvability of compressed exponent equations for a graph group belongs to NP. Graph groups will be introduced in the next section.

4 Traces and Graph Groups

Let (A, I) be a finite simple graph. In other words, the edge relation $I \subseteq A \times A$ is irreflexive and symmetric. It is also called the *independence relation*, and (A, I) is called an *independence alphabet*. We consider the monoid $\mathbb{M}(A, I) = A^*/\equiv_I$, where $\equiv_I$ is the smallest congruence relation on the free monoid A^* that contains all pairs (ab, ba) with $a, b \in A$ and $(a, b) \in I$. This monoid is called a *trace monoid* or *partially commutative free monoid*; it is cancellative, i.e., $xy = xz$ or $yx = zx$ implies $y = z$. Elements of $\mathbb{M}(A, I)$ are called *Mazurkiewicz traces* or simply *traces*. The trace represented by the word u is denoted by $[u]_I$, or simply u if no confusion can arise. For a language $L \subseteq A^*$ we denote with $[L]_I = \{u \in A^* \mid \exists v \in L : u \equiv_I v\}$ its *partially commutative closure*. The length of the trace $[u]_I$ is $|[u]_I| = |u|$ and its alphabet is $\mathrm{alph}([u]_I) = \mathrm{alph}(u)$. It is easy to see that these definitions do not depend on the concrete word that represents the trace $[u]_I$. For subsets $B, C \subseteq A$ we write BIC for $B \times C \subseteq I$. If $B = \{a\}$ we simply write aIC. For traces s, t we write sIt for $\mathrm{alph}(s) I \mathrm{alph}(t)$. The empty trace $[\varepsilon]_I$ is the identity element of the monoid $\mathbb{M}(A, I)$ and is denoted by 1. A trace t is *connected* if we cannot factorize t as $t = uv$ with $u \neq 1 \neq v$ and uIv. For a trace $t \in \mathbb{M}(A, I)$ let $\rho(t)$ be the number of prefixes of t. We will use the following statement from [4].

▶ **Lemma 2.** *Let $t \in \mathbb{M}(A, I)$ be a trace of length n. Then $\rho(t) \in \mathcal{O}(n^\alpha)$, where α is the size of a largest clique of the complementary graph $(A, I)^c = (A, (A \times A) \setminus I)$.*

We define the group $\mathbb{G}(A, I) = \langle A \mid ab = ba \ ((a, b) \in I)\rangle$. Such a group is called a *graph group*, or *right-angled Artin group*, or *free partially commutative group*. Here, we use the term graph group. We represent elements of $\mathbb{G}(A, I)$ by traces over an extended independence alphabet. For this, let $A^{-1} = \{a^{-1} \mid a \in A\}$ be a disjoint copy of the alphabet A, and let

$A^{\pm 1} = A \cup A^{-1}$. We define $(a^{-1})^{-1} = a$ and for a word $w = a_1 a_2 \cdots a_n$ with $a_i \in A^{\pm 1}$ we define $w^{-1} = a_n^{-1} \cdots a_2^{-1} a_1^{-1}$. This defines an involution (without fixed points) on $(A^{\pm 1})^*$. We extend the independence relation I to $A^{\pm 1}$ by $(a^x, b^y) \in I$ for all $(a, b) \in I$ and $x, y \in \{-1, 1\}$. For a trace $t = [u]_I$ ($u \in (A^{\pm 1})^*$) we can then define $t^{-1} = [u^{-1}]_I$. This is well-defined, since $u \equiv_I v$ implies $u^{-1} \equiv_I v^{-1}$. There is a canonical surjective morphism $h \colon \mathbb{M}(A^{\pm 1}, I) \to \mathbb{G}(A, I)$ that maps every symbol $a \in A^{\pm 1}$ to the corresponding group element. Of course, h is not injective, but we can easily define a subset $\mathrm{IRR}(A^{\pm 1}, I) \subseteq \mathbb{M}(A^{\pm 1}, I)$ of *irreducible traces* such that h restricted to $\mathrm{IRR}(A^{\pm 1}, I)$ is bijective. The set $\mathrm{IRR}(A^{\pm 1}, I)$ consists of all traces $t \in \mathbb{M}(A^{\pm 1}, I)$ such that t does not contain a factor $[aa^{-1}]_I$ with $a \in A^{\pm 1}$, i.e., there do not exist $u, v \in \mathbb{M}(A^{\pm 1}, I)$ and $a \in A^{\pm 1}$ such that in $\mathbb{M}(A^{\pm 1}, I)$ we have a factorization $t = u[aa^{-1}]_I v$. For every trace t there exists a corresponding *irreducible normal form* that is obtained by removing from t factors $[aa^{-1}]_I$ with $a \in A^{\pm 1}$ as long as possible. It can be shown that this reduction process is terminating (which is trivial since it reduces the length) and confluent (in [28] a more general confluence lemma for graph products of monoids is shown). Hence, the irreducible normal form of t does not depend on the concrete order of reduction steps. For a group element $g \in \mathbb{G}(A, I)$ we denote with $|g|$ the length of the unique trace $t \in \mathrm{IRR}(A^{\pm 1}, I)$ such that $h(t) = g$.

5 Three Auxiliary Results

Based on Levi's lemma for traces (see e.g. [10, p. 74]) one can show the following factorization result for powers of a connected trace.

▶ **Lemma 3.** *Let $u \in \mathbb{M}(A, I) \setminus \{1\}$ be a connected trace and $m \in \mathbb{N}$, $m \geq 2$. Then, for all $x \in \mathbb{N}$ and traces $y_1, \ldots, y_m$ we have: $u^x = y_1 y_2 \cdots y_m$ if and only if there exist traces $p_{i,j}$ $(1 \leq j < i \leq m)$, s_i $(1 \leq i \leq m)$ and $x_i, c_j \in \mathbb{N}$ $(1 \leq i \leq m,\ 1 \leq j \leq m-1)$ such that:*

- $y_i = (\prod_{j=1}^{i-1} p_{i,j}) u^{x_i} s_i$ *for all* $1 \leq i \leq m$,
- $p_{i,j} I p_{k,l}$ *if* $j < l < k < i$ *and* $p_{i,j} I (u^{x_k} s_k)$ *if* $j < k < i$
- $s_m = 1$ *and for all* $1 \leq j < m$, $s_j \prod_{i=j+1}^{m} p_{i,j} = u^{c_j}$
- $c_j \leq |A|$ *for all* $1 \leq j \leq m-1$,
- $x = \sum_{i=1}^{m} x_i + \sum_{i=1}^{m-1} c_i$.

▶ Remark 4. In Section 6 we will apply Lemma 3 to replace an equation $u^x = y_1 y_2 \cdots y_m$ (where $x, y_1, \ldots, y_m$ are variables and u is a concrete connected trace) by an equivalent disjunction. Note that the length of all factors $p_{i,j}$ and s_i above is bounded by $|A| \cdot |u|$. Hence, one can guess these traces as well as the numbers $c_j \leq |A|$ (the guess results in a disjunction). We can also guess which of the numbers x_i are zero and which are greater than zero. After these guesses we can verify the independences $p_{i,j} I p_{k,l}$ $(j < l < k < i)$ and $p_{i,j} I (u^{x_k} s_k)$ $(j < k < i)$, and the identities $s_m = 1$, $s_j \prod_{i=j+1}^{m} p_{i,j} = u^{c_j}$ $(1 \leq j < m)$. If one of them does not hold, the specific guess does not contribute to the disjunction. In this way, we can replace the equation $u^x = y_1 y_2 \cdots y_m$ by a disjunction of formulas of the form

$$\exists x_i > 0\ (i \in K) : x = \sum_{i \in K}^{m} x_i + c \wedge \bigwedge_{i \in K} y_i = p_i u^{x_i} s_i \wedge \bigwedge_{i \in [1,m] \setminus K} y_i = p_i s_i \,,$$

where $K \subseteq [1, m]$, $c \leq |A| \cdot (m-1)$ and the p_i, s_i are concrete traces of length at most $|A| \cdot (m-1) \cdot |u|$. The number of disjuncts in the disjunction is not important for our purpose.

The second auxiliary result that we need is (recall that $\rho(t)$ is the number of prefixes of the trace t):

▶ **Lemma 5.** *Let $p, q, u, v, s, t \in \mathbb{M}(A, I)$ such that $u \neq 1$ and $v \neq 1$ are connected. Furthermore, let $m = \max\{\rho(p), \rho(q), \rho(s), \rho(t)\}$ and $n = \max\{\rho(u), \rho(v)\}$. Then the set $L(p, u, s, q, v, t) := \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid pu^x s = qv^y t\}$ is semilinear and is a union of $\mathcal{O}(m^8 \cdot n^{4|A|})$ many linear sets of the form $\{(a + bz, c + dz) \mid z \in \mathbb{N}\}$ with $a, b, c, d \in \mathcal{O}(m^8 \cdot n^{4|A|})$.*

The proof of Lemma 5 applies the theory of recognizable trace languages. We construct an automaton for the language $L = [pu^x s]_I \cap [qv^y t]_I$ with at most $4m^4 \cdot n^{2 \cdot |A|}$ states. Then, we analyze the set of all lengths of words from L using results on unary finite automata [39].

Finally, we need a bound on the norm of a smallest vector in a certain kind of semilinear sets. We easily obtain this bound from a result by zur Gathen and Sieveking [40].

▶ **Lemma 6.** *Let $A \in \mathbb{Z}^{n \times m}$, $\overline{a} \in \mathbb{Z}^n$, $C \in \mathbb{N}^{k \times m}$, $\overline{c} \in \mathbb{N}^k$. Let β be an upper bound for the absolute value of all entries in A, $\overline{a}$, C, $\overline{c}$. The set $L = \{C\overline{z} + \overline{c} \mid \overline{z} \in \mathbb{N}^m, A\overline{z} = \overline{a}\} \subseteq \mathbb{N}^k$ is semilinear. Moreover, if $L \neq \emptyset$ then L contains a vector with all entries bounded by $\beta + n! \cdot m \cdot (m + 1) \cdot \beta^{n+1}$.*

6 Exponent Equations in Graph Groups

In this section, we prove the following two statements, where G is a fixed graph group:

- The set of solutions of an exponent equation over G is (effectively) semilinear.
- Solvability of compressed exponent equations over G belongs to NP.

In the next section, we will extend these results to the larger class of virtually special groups. We start with some definitions. As usual, we fix an independence alphabet (A, I). In the following we will consider reduction rules on sequences of traces. For better readability we separate the consecutive traces in such a sequence by commas. Let $u_1, u_2, \ldots, u_n \in \mathrm{IRR}(A^{\pm 1}, I)$ be irreducible traces. The sequence $u_1, u_2, \ldots, u_n$ is *I-freely reducible* if the sequence $u_1, u_2, \ldots, u_n$ can be reduced to the empty sequence ε by the following rules:

- $u_i, u_j \to u_j, u_i$ if $u_i I u_j$,
- $u_i, u_j \to \varepsilon$ if $u_i = u_j^{-1}$ in $\mathbb{G}(A, I)$,
- $u_i \to \varepsilon$ if $u_i = \varepsilon$ (this rule deletes the empty trace ε from a sequence of traces).

A concrete sequence of these rewrite steps leading to the empty sequence is a *reduction* of the sequence $u_1, u_2, \ldots, u_n$. Such a reduction can be seen as a witness for the fact that $u_1 u_2 \cdots u_n = 1$ in $\mathbb{G}(A, I)$. On the other hand, $u_1 u_2 \cdots u_n = 1$ does not necessarily imply that $u_1, u_2, \ldots, u_n$ has a reduction. For instance, the sequence a^{-1}, ab, b^{-1} has no reduction. But we can show that every sequence which multiplies to 1 in G can be refined (by factorizing the elements of the sequence) such that the resulting refined sequence has a reduction. For getting an NP-algorithm, it is important to bound the length of the refined sequence exponentially in the length of the initial sequence.

▶ **Lemma 7.** *Let $n \geq 2$ and $u_1, u_2, \ldots, u_n \in \mathrm{IRR}(A^{\pm 1}, I)$. If $u_1 u_2 \cdots u_n = 1$ in $\mathbb{G}(A, I)$, then there exist factorizations $u_i = u_{i,1} \cdots u_{i,k_i}$ such that the sequence*

$$u_{1,1}, \ldots, u_{1,k_1},\ u_{2,1}, \ldots, u_{2,k_2},\ \ldots,\ u_{n,1}, \ldots, u_{n,k_n}$$

is I-freely reducible. Moreover, $\sum_{i=1}^{n} k_i \leq 2^n - 2$.

We now come to the main technical result of this paper:

▶ **Theorem 8.** *Let $u_1, u_2, \ldots, u_n \in \mathbb{G}(A, I) \setminus \{1\}$, $v_0, v_1, \ldots, v_n \in \mathbb{G}(A, I)$ and let $x_1, \ldots, x_n$ be variables (we may have $x_i = x_j$ for $i \neq j$) ranging over $\mathbb{N}$. Then, the set of solutions of the exponent equation*

$$v_0 u_1^{x_1} v_1 u_2^{x_2} v_2 \cdots u_n^{x_n} v_n = 1, \tag{1}$$

is semilinear. Furthermore, if there exists a solution, then there is a solution such that $x_i \in \mathcal{O}((\alpha n)! \cdot 2^{2\alpha^2 n(n+3)} \cdot \mu^{8\alpha(n+1)} \cdot \nu^{8\alpha|A|(n+1)})$*, where*

- $\alpha \leq |A|$ *is the size of a largest clique of the complementary graph* $(A, I)^c = (A, (A \times A) \setminus I)$*,*
- $\lambda = \max\{|u_1|, |u_2|, \ldots, |u_n|, |v_0|, |v_1|, \ldots, |v_n|\}$*,*
- $\mu \in \mathcal{O}(|A|^\alpha \cdot 2^{2\alpha^2 n} \cdot \lambda^\alpha)$*, and*
- $\nu \in \mathcal{O}(\lambda^\alpha)$*.*

Proof. Let us choose irreducible traces for $u_1, u_2, \ldots, u_n, v_0, v_1, \ldots, v_n$; we denote these traces with the same letters as the group elements. A trace u is called *cyclically reduced* if there do not exist $a \in A^{\pm 1}$ and v such that $u = ava^{-1}$. For every trace u there exist unique traces p, w such that $u = pwp^{-1}$ and w is cyclically reduced (since the reduction relation $a^{-1}xa \to x$ is terminating and confluent [11, Lemma 16]). These traces p and w can be computed in polynomial time. Note that for a cyclically reduced irreducible trace w, every power w^n is irreducible. Let $u_i = p_i w_i p_i^{-1}$ with w_i cyclically reduced. By replacing every $u_i^{x_i}$ by $p_i w_i^{x_i} p_i^{-1}$, we can assume that all u_i are cyclically reduced and irreducible. In case one of the traces u_i is not connected, we can write u_i as $u_i = u_{i,1}u_{i,2}$ with $u_{i,1} I u_{i,2}$ and $u_{i,1} \neq 1 \neq u_{i,2}$. Thus, we can replace the power $u_i^{x_i}$ by $u_{i,1}^{x_i} u_{i,2}^{x_i}$. Note that $u_{i,1}$ and $u_{i,2}$ are still irreducible and cyclically reduced. By doing this, the number n from the theorem multiplies by at most α (which is the maximal number of pairwise independent letters). In order to keep the notation simple we still use the letter n for the number of u_i, but at the end of the proof we have to multiply n by α in the derived bound. Hence, for the further proof we can assume that all u_i are connected, irreducible and cyclically reduced. Let λ be the maximal length of one of the traces $u_1, u_2, \ldots, u_n, v_0, v_1, \ldots, v_n$, which does not increase by the above preprocessing.

We now apply Lemma 7 to the equation (1), where every $u_i^{x_i}$ is viewed as a single factor. Note that by our preprocessing, all factors $u_1^{x_1}, u_2^{x_2}, \ldots, u_n^{x_n}, v_0, \ldots, v_n$ are irreducible (for all choices of the x_i). By taking the disjunction over (i) all possible factorizations of the $2n+1$ factors $u_1^{x_1}, u_2^{x_2}, \ldots, u_n^{x_n}, v_0, \ldots, v_n$ into totally at most $2^{2n+1} - 2$ factors and (ii) all possible reductions of the resulting refined factorization of $v_0 u_1^{x_1} v_1 u_2^{x_2} v_2 \cdots u_n^{x_n} v_n$, it follows that (1) is equivalent to a disjunction of statements of the following form: There exist traces $y_{i,1}, \ldots, y_{i,k_i}$ $(1 \leq i \leq n)$ and $z_{i,1}, \ldots, z_{i,l_i}$ $(0 \leq i \leq n)$ such that

(a) $u_i^{x_i} = y_{i,1} \cdots y_{i,k_i}$ $(1 \leq i \leq n)$
(b) $v_i = z_{i,1} \cdots z_{i,l_i}$ $(0 \leq i \leq n)$
(c) $y_{i,j} I y_{k,l}$ for all $(i,j,k,l) \in J_1$
(d) $y_{i,j} I z_{k,l}$ for all $(i,j,k,l) \in J_2$
(e) $z_{i,j} I z_{k,l}$ for all $(i,j,k,l) \in J_3$
(f) $y_{i,j} = y_{k,l}^{-1}$ for all $(i,j,k,l) \in M_1$
(g) $y_{i,j} = z_{k,l}^{-1}$ for all $(i,j,k,l) \in M_2$
(h) $z_{i,j} = z_{k,l}^{-1}$ for all $(i,j,k,l) \in M_3$

Here, the numbers k_i and l_i sum up to at most $2^{2n+1} - 2$ (hence, some k_i can be exponentially large, whereas l_i can be bound by the length of v_i, which is at most λ). The tuple sets J_1, J_2, J_3 collect all independences between the factors $y_{i,j}$, $z_{k,l}$ that are necessary to carry out the chosen reduction of the refined left-hand side in (1). Similarly, the tuple sets M_1, M_2, M_3 tell us which of the factors $y_{i,j}$, $z_{k,l}$ cancels against which of the factors $y_{i,j}$, $z_{k,l}$ in our chosen reduction of the refined left-hand side in (1). Note that every factor $y_{i,j}$ (resp., $z_{k,l}$) appears in exactly one of the identities (f), (g), (h) (since in the reduction every factor cancels against another unique factor).

Next, we simplify our statements. Since the v_i are concrete traces (of length at most λ), we can take a disjunction over all possible factorizations $v_i = v_{i,1} \cdots v_{i,l_i}$ $(1 \leq i \leq n+1)$. This allows to replace every variable $z_{i,j}$ by a concrete trace $v_{i,j}$. Statements of the form $v_{i,j} I v_{k,l}$ and $v_{i,j} = v_{k,l}^{-1}$ can, of course, be eliminated. Moreover, if there is an identity $y_{i,j} = v_{k,l}^{-1}$ then we can replace the variable $y_{i,j}$ by the concrete trace $v_{k,l}^{-1}$ (of length at most λ).

In the next step, we replace statements of the form $u_i^{x_i} = y_{i,1} \cdots y_{i,k_i}$ $(1 \leq i \leq n)$. Note that some of the variables $y_{i,j}$ might have been replaced by concrete traces of length at most λ. We apply to each of these equations Lemma 3, or better Remark 4. This allows us to replace every equation $u_i^{x_i} = y_{i,1} \cdots y_{i,k_i}$ $(1 \leq i \leq n)$ by a disjunction of statements of the following form: There exist numbers $x_{i,j} > 0$ $(1 \leq i \leq n,\ j \in K_i)$ such that

- $x_i = c_i + \sum_{j \in K_i} x_{i,j}$ for all $1 \leq i \leq n$,
- $y_{i,j} = p_{i,j} u_i^{x_{i,j}} s_{i,j}$ for all $1 \leq i \leq n$, $j \in K_i$,
- $y_{i,j} = p_{i,j} s_{i,j}$ for all $1 \leq i \leq n$, $j \in [1, k_i] \setminus K_i$.

Here, $K_i \subseteq [1, k_i]$, the c_i are concrete numbers with $c_i \leq |A| \cdot (k_i - 1)$, and the $p_{i,j}, s_{i,j}$ are concrete traces of length at most $|A| \cdot (k_i - 1) \cdot |u_i| \leq |A| \cdot (2^{2n+1} - 3) \cdot \lambda$. Hence, the lengths of these traces can be exponential in n.

Note that since $x_i > 0$, we know the alphabet of $y_{i,j} = p_{i,j} u_i^{x_{i,j}} s_{i,j}$ (resp., $y_{i,j} = p_{i,j} s_{i,j}$). This allows us to replace all independences of the form $y_{i,j} I y_{k,l}$ for $(i,j,k,l) \in J_1$ (see (c)) and $y_{i,j} I z_{k,l}$ for $(i,j,k,l) \in J_2$ (see (d)) by concrete truth values. Note that all variables $z_{k,l}$ have already been replaced by concrete traces. If $y_{i,j}$ was already replaced by a concrete trace, then we can determine from an equation $y_{i,j} = p_{i,j} u_i^{x_{i,j}} s_{i,j}$ the exponent $x_{i,j}$. Since $y_{i,j}$ was replaced by a trace of length at most λ (a small number), we get $x_{i,j} \leq \lambda$, and we can replace $x_{i,j}$ in $x_i = \sum_{j \in K_i} x_{i,j} + c_i$ by a concrete number of size at most λ. Finally, if $y_{i,j}$ was replaced by a concrete trace, and we have an equation of the form $y_{i,j} = p_{i,j} s_{i,j}$, then the resulting identity is either true or false and can be eliminated.

After this step, we obtain a disjunction of statements of the following form: There exist numbers $x_{i,j} > 0$ $(1 \leq i \leq n,\ j \in K_i')$ such that

(a') $x_i = c_i + \sum_{j \in K_i'} x_{i,j}$ for all $1 \leq i \leq n$, and

(b') $p_{i,j} u_i^{x_{i,j}} s_{i,j} = s_{k,l}^{-1} (u_k^{-1})^{x_{k,l}} p_{k,l}^{-1}$ for all $(i,j,k,l) \in M$.

Here, $K_i' \subseteq K_i$ is a set of size at most $k_i \leq 2^{2n+1} - 2$, $c_i \leq |A| \cdot (k_i - 1) + \lambda \cdot k_i < (|A| + \lambda) \cdot (2^{2n+1} - 2)$, and the $p_{i,j}, s_{i,j}$ are concrete traces of length at most $|A| \cdot (2^{2n+1} - 3) \cdot \lambda$. The set M specifies a matching in the sense that for every exponent $x_{a,b}$ $(1 \leq a \leq n,\ b \in K_i')$ there is a unique $(i,j,k,l) \in M$ such that $(i,j) = (a,b)$ or $(k,l) = (a,b)$.

We now apply Lemma 5 to the identities $p_{i,j} u_i^{x_{i,j}} s_{i,j} = s_{k,l}^{-1} (u_k^{-1})^{x_{k,l}} p_{k,l}^{-1}$. Each such identity can be replaced by a disjunction of constraints

$$(x_{i,j}, x_{k,l}) \in \{(a_{i,j,k,l} + b_{i,j,k,l} \cdot z_{i,j,k,l}, c_{i,j,k,l} + d_{i,j,k,l} \cdot z_{i,j,k,l}) \mid z_{i,j,k,l} \in \mathbb{N}\}.$$

For the numbers $a_{i,j,k,l}, b_{i,j,k,l}, c_{i,j,k,l}, d_{i,j,k,l}$ we obtain the bound

$$a_{i,j,k,l}, b_{i,j,k,l}, c_{i,j,k,l}, d_{i,j,k,l} \in \mathcal{O}(\mu^8 \cdot \nu^{8|A|})$$

(the alphabet of the traces is $A^{\pm 1}$ which has size $2|A|$, therefore, we have to multiply in Lemma 5 $|A|$ by 2), where, by Lemma 2,

$$\mu = \max\{\rho(p_{i,j}), \rho(p_{k,l}), \rho(s_{i,j}), \rho(s_{k,l})\} \in \mathcal{O}(|A|^\alpha \cdot 2^{2\alpha n} \cdot \lambda^\alpha) \text{ and} \tag{2}$$

$$\nu = \max\{\rho(u_i), \rho(u_k)\} \in \mathcal{O}(\lambda^\alpha). \tag{3}$$

Note that $\rho(t) = \rho(t^{-1})$ for every trace t. The above condition (a') for x_i can be written as

$$x_i = c_i + \sum_{(i,j,k,l) \in M} (a_{i,j,k,l} + b_{i,j,k,l} \cdot z_{i,j,k,l}) + \sum_{(k,l,i,j) \in M} (c_{k,l,i,j} + d_{k,l,i,j} \cdot z_{k,l,i,j}).$$

Note that the two sums in this equation contain in total $|K_i'| \leq 2^{2n+1}$ many summands (since for every $j \in K_i'$ there is a unique pair (k,l) with $(i,j,k,l) \in M$ or $(k,l,i,j) \in M$).

Hence, after a renaming of symbols, the initial equation (1) becomes equivalent to a finite disjunction of statements of the form: There exist $z_1, \dots, z_m \in \mathbb{N}$ (these z_i are the above $z_{i,j,k,l}$ and $m = \max_i |K_i'|$) such that

$$x_i = a_i + \sum_{j=1}^{m} a_{i,j} z_j \text{ for all } 1 \leq i \leq n. \tag{4}$$

Moreover, we have the following size bounds:

- $m = \max_i |K_i'| \leq 2^{2n+1}$,
- $a_i \in \mathcal{O}(c_i + |K_i'| \cdot \mu^8 \cdot \nu^{8|A|}) \subseteq \mathcal{O}(2^{2n}(|A| + \lambda + \mu^8 \cdot \nu^{8|A|})) \subseteq \mathcal{O}(2^{2n} \cdot \mu^8 \cdot \nu^{8|A|})$
- $a_{i,j} \in \mathcal{O}(\mu^8 \cdot \nu^{8|A|})$

Recall that some of the variables x_i can be identical. W.l.o.g. assume that $x_1, \dots, x_k$ are pairwise different and for all $k+1 \leq i \leq n$, $x_i = x_{f(i)}$, where $f : [k+1, n] \to [1, k]$. Then, the system of equations (4) is equivalent to

$$x_i = a_i + \sum_{j=1}^{m} a_{i,j} z_j \ (1 \leq i \leq k) \quad \text{and} \quad a_i - a_{f(i)} = \sum_{j=1}^{m} (a_{f(i),j} - a_{i,j}) z_j \ (k+1 \leq i \leq n).$$

The set of all $(x_1, \dots, x_k) \in \mathbb{N}^k$ for which there exist $z_1, \dots, z_m \in \mathbb{N}$ satisfying these equalities is semilinear by Lemma 6, and if it is non-empty then it contains $(x_1, \dots, x_k) \in \mathbb{N}^k$ such that $x_i \in \mathcal{O}(n! \cdot m^2 \cdot 2^{2n(n+1)} \cdot \mu^{8(n+1)} \cdot \nu^{8|A|(n+1)}) \subseteq \mathcal{O}(n! \cdot 2^{2n(n+3)} \cdot \mu^{8(n+1)} \cdot \nu^{8|A|(n+1)})$. Recall that in this bound we have to replace n by $\alpha \cdot n$ due to the initial preprocessing. This proves the theorem. ◀

▶ **Theorem 9.** *Let (A, I) be a fixed independence alphabet. Solvability of compressed exponent equations over the graph group $\mathbb{G}(A, I)$ is in NP.*

Proof. Consider a compressed exponent equation $E = (v_0 u_1^{x_1} v_1 u_2^{x_2} v_2 \cdots u_n^{x_n} v_n = 1)$, where $u_i = \mathrm{val}(\mathcal{G}_i)$ and $v_i = \mathrm{val}(\mathcal{H}_i)$ for SLPs $\mathcal{G}_1, \dots, \mathcal{G}_n, \mathcal{H}_0, \dots, \mathcal{H}_n$, which form the input. Let $m = \max\{|\mathcal{G}_1|, \dots, |\mathcal{G}_n|, |\mathcal{H}_0|, \dots, |\mathcal{H}_n|\}$, $\lambda = \max\{|u_1|, |u_2|, \dots, |u_n|, |v_0|, |v_1|, \dots, |v_n|\} \in 2^{\mathcal{O}(m)}$. By Thm. 8 we know that if there exists a solution for E then there exists a solution σ with $\sigma(x_i) \in \mathcal{O}((\alpha n)! \cdot 2^{2\alpha^2 n(n+3)} \cdot \mu^{8\alpha(n+1)} \cdot \nu^{8\alpha|A|(n+1)})$, where $\mu \in \mathcal{O}(|A|^\alpha \cdot 2^{2\alpha^2 n} \cdot \lambda^\alpha)$, $\nu \in \mathcal{O}(\lambda^\alpha)$, and $\alpha \leq |A|$. Note that the bound on the $\sigma(x_i)$ is exponential in the input length (the sum of the sizes of all $\mathcal{G}_i$ and $\mathcal{H}_i$). Hence, we can guess in polynomial time the binary encodings of numbers $k_i \in \mathcal{O}((\alpha n)! \cdot 2^{2\alpha^2 n(n+3)} \cdot \mu^{8\alpha(n+1)} \cdot \nu^{8\alpha|A|(n+1)})$ (where $k_i = k_j$ if $x_i = x_j$). It remains to verify the identity $v_0 u_1^{k_1} v_1 u_2^{k_2} v_2 \cdots u_n^{k_n} v_n = 1$ in $\mathbb{G}(A, I)$, where all u_i and v_i are given by SLPs. This is an instance of the so-called *compressed word problem* for $\mathbb{G}(A, I)$, where the input consists of an SLP $\mathcal{G}$ over the alphabet $A^{\pm 1}$ and it is asked whether $\mathrm{val}(\mathcal{G}) = 1$ in $\mathbb{G}(A, I)$. Note that the big powers $\mathrm{val}(\mathcal{G}_i)^{k_i}$ can be produced with the productions of $\mathcal{G}_i$ and additional $\lceil \log k_i \rceil$ many productions (using iterated squaring). Since the compressed word problem for a graph group can be solved in polynomial time [29] (NP would suffice), the theorem follows. For the last step, it is important that (A, I) is fixed. ◀

▶ Remark 10. Note that the bound on the exponents $\sigma(x_i)$ in the previous proof is still exponential in the input length if the independence alphabet (A, I) is part of the input as well. The problem is that we do not know whether the *uniform compressed word problem* for graph groups (where the input is an independence alphabet (A, I) together with an SLP over the terminal alphabet $A^{\pm 1}$) can be solved in polynomial time or at least in NP. The latter would suffice to get an NP-algorithm for solvability of compressed exponent equations over a graph group that is part of the input.

7 Transfer Results

Here, we show that the property of having an NP-algorithm for the knapsack problem (or compressed exponent equations) is preserved by certain group constructions.

Finite extensions and virtually special groups. Our first transfer result concerns finite extensions. Together with our result on graph groups, this will provide an abundant class of groups with an NP-algorithm for compressed exponent equations. A group G is called *virtually special* if it is a finite extension of a subgroup of a graph group. Recently, this class of groups turned out to be very rich. It includes all Coxeter groups [18], one-relator groups with torsion [41], fully residually free groups [41], and fundamental groups of hyperbolic 3-manifolds [1].

The following is our transfer theorem for finite extensions. The idea is to guess the cosets that occur on the left-hand side of an exponent equation. Given a collection of cosets, one can then reduce to exponent equations over G.

▶ **Theorem 11.** *Let G and H be finitely generated groups such that H is a finite extension of G. If knapsack (resp. solvability of compressed exponent equations) belongs to NP for G, then the same holds for H.*

From Theorem 9 it follows that solvability of compressed exponent equations belongs to NP for every subgroup of a graph group. Therefore, our transfer theorem implies:

▶ **Theorem 12.** *Solvability of compressed exponent equations belongs to NP for every virtually special group. In particular, solvability of compressed exponent equations belongs to NP for Coxeter groups, one-relator groups with torsion, fully residually free groups, and fundamental groups of hyperbolic 3-manifolds.*

HNN-extensions and amalgamated products. The remaining transfer results concern two constructions that are of fundamental importance in combinatorial group theory [34], namely HNN-extensions with finite associated subgroups and amalgamated products with finite identified subgroups. These constructions are known to preserve a variety of important structural and algorithmic properties [2, 6, 19, 22, 23, 25, 26, 30, 31, 35].

Suppose G is a finitely generated group that has two isomorphic subgroups A and B with an isomorphism $\varphi\colon A \to B$. Then the corresponding *HNN-extension* is the group $H = \langle G, t \mid t^{-1}at = \varphi(a)\ (a \in A)\rangle$, where t is a new letter not contained in G. Intuitively, H is obtained from G by adding a new element t such that conjugating elements of A with t applies the isomorphism φ. Here, t is called the *stable letter* and the groups A and B are the *associated subgroups*. A basic fact about HNN-extensions is that the group G embeds naturally into H [20].

Our algorithm for knapsack in HNN-extensions is an adaptation of the saturation algorithm of Benois [3] for the membership problem for rational subsets of free groups. Here, for each path spelling aa^{-1}, one adds a parallel edge labeled with the empty word. Since knapsack is a special case of this problem, we need to choose a suitable subclass of automata that is preserved by our saturation and corresponds to the knapsack problem. This subclass is the class of *knapsack automata*, where each strongly connected component is a singleton or an induced (directed) cycle.

With respect to NP-membership, one can show that the knapsack problem is equivalent to the membership problem for knapsack automata (see [33]). Therefore, it suffices to show that NP-membership of the latter problem is preserved by HNN-extensions, which we prove

using a saturation procedure. Here, the basic idea is to successively guess states p and q and check (using the algorithm for G) whether there is a path from p to q reading a word $u = t^{-1}wt$ (resp. $u = twt^{-1}$) such that w represents an element $a \in A$ (resp. $b \in B$). If this is the case, u represents $\varphi(a)$ (resp. $\varphi^{-1}(b)$) and we add an edge $p \xrightarrow{\varphi(a)} q$ (resp. $p \xrightarrow{\varphi^{-1}(b)} q$), which is termed *shortcut edge*.

This, however, it is not possible if p and q lie on a cycle, as that would create connected components that are not cycles. In this case, we replace the cycle segment between p and q with the shortcut edge and glue in new paths to make up for lost edges that entered or left the cycle between p and q. In the end, we show that every element accepted by the automaton has an accepting run that avoids factors $t^{-1}wt$ and twt^{-1} as above. This allows us then to apply the algorithm for G to decide the membership problem.

▶ **Theorem 13.** *Let H be an HNN-extension of the finitely generated group G with finite associated subgroups. If knapsack for G belongs to NP, then the same holds for H.*

In our last transfer theorem, we consider amalgamated free products. For each $i \in \{0, 1\}$, let $G_i = \langle \Sigma_i \mid R_i \rangle$ be a finitely generated group and let F be a finite group that is embedded in each G_i via an injective morphism $\varphi_i \colon F \to G_i$. Then, the *free product with amalgamation with identified subgroup* F is defined as $G_0 *_F G_1 = \langle G_0 * G_1 \mid \varphi_0(f) = \varphi_1(f) \ (f \in F) \rangle$. Here, $G_0 * G_1$ denotes the free product $G_0 * G_1 = \langle \Sigma_0 \uplus \Sigma_1 \mid R_0 \uplus R_1 \rangle$. Intuitively, $G_0 *_F G_1$ consists of alternating sequences of elements of G_0 and G_1 where the elements of $\varphi_0(F)$ and $\varphi_1(F)$ are identified.

The transfer theorem states that taking amalgamated products with finite identified subgroups preserves NP-membership of knapsack. Since $G_0 *_F G_1$ can be embedded into the HNN-extension $\langle G_0 * G_1, t \mid t^{-1}\varphi_0(f)t = \varphi_1(f) \ (f \in F) \rangle$, it suffices to prove the theorem for a free product $G_0 * G_1$. As above, we work with the membership problem for knapsack automata and use a saturation procedure. Note that $G_0 * G_1$ is generated by $\Sigma_0 \cup \Sigma_1$. We guess states p and q and $i \in \{0, 1\}$ and then check, using the NP algorithm for G_i, whether there is a path from p to q that reads a representative of $1 \in G_i$ in $(\Sigma_i^{\pm 1})^*$. If so, we add a shortcut edge (p, ε, q). Again, the case that p and q lie on a cycle is somewhat more involved.

▶ **Theorem 14.** *Let G_0 and G_1 be finitely generated groups with a common finite subgroup F. If knapsack for G_0 and for G_1 belongs to NP, then the same holds for $G_0 *_F G_1$.*

8 Hardness Results

Since knapsack for binary encoded integers is NP-complete, it follows that the compressed knapsack problem is NP-hard for every group that contains an element of infinite order. Our final result states that (uncompressed) knapsack and subset sum are NP-complete for a direct product of two free groups of rank two. Since this group is a graph group, we obtain a matching lower bound for Thm. 9. Moreover, we solve an open problem from [15].

▶ **Theorem 15.** *The subset sum problem and the knapsack problem are NP-complete for $F_2 \times F_2$, where F_2 is the free group of rank two. For knapsack, NP-hardness already holds for the variant where the exponent variables are allowed to take values from $\mathbb{Z}$ (see Remark 1).*

References

1 I. Agol. The virtual Haken conjecture. Technical report, arXiv.org, 2012. URL: `http://arxiv.org/abs/1204.2810`.

2 R. B. J. T. Allenby and R. J. Gregorac. On locally extended residually finite groups. In *Conference on Group Theory (Univ. Wisconsin-Parkside, Kenosha, Wis., 1972)*, number 319 in Lecture Notes in Mathematics, pages 9–17. Springer, 1973.

3 M. Benois. Parties rationnelles du groupe libre. *C. R. Acad. Sci. Paris*, Sér. A, 269:1188–1190, 1969.

4 A. Bertoni, G. Mauri, and N. Sabadini. Membership problems for regular and context free trace languages. *Information and Computation*, 82:135–150, 1989.

5 M. Bestvina and N. Brady. Morse theory and finiteness properties of groups. *Inventiones Mathematicae*, 129(3):445–470, 1997.

6 V. N. Bezverkhniĭ. On the intersection of subgroups in HNN-groups. *Fundamentalnaya i Prikladnaya Matematika*, 4(1):199–222, 1998.

7 M. Charikar, E. Lehman, A. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.

8 J. Crisp and B. Wiest. Embeddings of graph braid and surface groups in right-angled Artin groups and braid groups. *Algebraic & Geometric Topology*, 4:439–472, 2004.

9 W. Dicks and M. J. Dunwoody. *Groups Acting on Graphs*. Cambridge University Press, 1989.

10 V. Diekert and G.Rozenberg, editors. *The Book of Traces*. World Scientific, 1995.

11 V. Diekert and J. Kausch. Logspace computations in graph products. *Journal of Symbolic Computation*, 2015. to appear. `doi:10.1016/j.jsc.2015.11.009`.

12 V. Diekert and M. Lohrey. Word equations over graph products. *International Journal of Algebra and Computation*, 18(3):493–533, 2008.

13 V. Diekert and A. Muscholl. Solvability of equations in graph groups is decidable. *International Journal of Algebra and Computation*, 16(6):1047–1069, 2006.

14 M. Elberfeld, A. Jakoby, and T. Tantau. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:128, 2011.

15 E. Frenkel, A. Nikolaev, and A. Ushakov. Knapsack problems in products of groups. *Journal of Symbolic Computation*, 74:96–108, 2016.

16 R. Ghrist and V. Peterson. The geometry and topology of reconfiguration. *Advances in Applied Mathematics*, 38(3):302–323, 2007.

17 C. Haase. *On the complexity of model checking counter automata*. PhD thesis, University of Oxford, St Catherine's College, 2011.

18 F. Haglund and D. T. Wise. Coxeter groups are virtually special. *Advances in Mathematics*, 224(5):1890–1903, 2010.

19 N. Haubold and M. Lohrey. Compressed word problems in HNN-extensions and amalgamated products. *Theory of Computing Systems*, 49(2):283–305, 2011.

20 G. Higman, B. H. Neumann, and H. Neumann. Embedding theorems for groups. *Journal of the London Mathematical Society. Second Series*, 24:247–254, 1949.

21 B. Jenner. Knapsack problems for NL. *Information Processing Letters*, 54(3):169–174, 1995.

22 M. Kambites, P. V. Silva, and B. Steinberg. On the rational subset problem for groups. *Journal of Algebra*, 309(2):622–639, 2007.

23 I. Kapovich, R. Weidmann, and A. Myasnikov. Foldings, graphs of groups and the membership problem. *International Journal of Algebra and Computation*, 15(1):95–128, 2005.

24 R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

25 A. Karrass and D. Solitar. The subgroups of a free product of two groups with an amalgamated subgroup. *Transactions of the American Mathematical Society*, 150:227–255, 1970.

26 A. Karrass and D. Solitar. Subgroups of HNN groups and groups with one defining relation. *Canadian Journal of Mathematics*, 23:627–643, 1971.

27 D. König, M. Lohrey, and G. Zetzsche. Knapsack and subset sum problems in nilpotent, polycyclic, and co-context-free groups. Technical report, arXiv.org, 2015. `http://arxiv.org/abs/1507.05145`.

28 D. Kuske and M. Lohrey. Logical aspects of Cayley-graphs: the monoid case. *International Journal of Algebra and Computation*, 16(2):307–340, 2006.

29 M. Lohrey. *The Compressed Word Problem for Groups*. SpringerBriefs in Mathematics. Springer, 2014.

30 M. Lohrey and G. Sénizergues. Theories of HNN-extensions and amalgamated products. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP 2006)*, number 4052 in Lecture Notes in Computer Science, pages 504–515. Springer, 2006.

31 M. Lohrey and G. Sénizergues. Rational subsets in HNN-extensions and amalgamated products. *International Journal of Algebra and Computation*, 18(1):111–163, 2008.

32 M. Lohrey and B. Steinberg. The submonoid and rational subset membership problems for graph groups. *Journal of Algebra*, 320(2):728–755, 2008.

33 M. Lohrey and G. Zetzsche. Knapsack in graph groups, HNN-extensions and amalgamated products. Technical report, arXiv.org, 2015. `http://arxiv.org/abs/1509.05957`.

34 R. C. Lyndon and P. E. Schupp. *Combinatorial Group Theory*. Springer, 1977.

35 V. Metaftsis and E. Raptis. Subgroup separability of graphs of abelian groups. *Proceedings of the American Mathematical Society*, 132:1873–1884, 2004.

36 A. Myasnikov, A. Nikolaev, and A. Ushakov. Knapsack problems in groups. *Mathematics of Computation*, 84:987–1016, 2015.

37 C. H. Papadimitriou. On the complexity of integer programming. *Journal of the Association for Computing Machinery*, 28(4):765–768, 1981.

38 J. R. Stallings. *Group Theory and Three-Dimensional Manifolds*. Number 4 in Yale Mathematical Monographs. Yale University Press, 1971.

39 A. W. To. Unary finite automata vs. arithmetic progressions. *Information Processing Letters*, 109(17):1010–1014, 2009.

40 J. von zur Gathen and M. Sieveking. A bound on solutions of linear integer equalities and inequalities. *Proceedings of the American Mathematical Society*, 72(1):155–158, 1978.

41 D. T. Wise. Research announcement: the structure of groups with a quasiconvex hierarchy. *Electronic Research Announcements in Mathematical Sciences*, 16:44–55, 2009.

FPTAS for Hardcore and Ising Models on Hypergraphs*

Pinyan Lu[1], Kuan Yang[2], and Chihao Zhang[3]

1 **School of Information Management and Engineering, Shanghai University of Finance and Economics, No. 100 Wudong Road, Yangpu District, Shanghai, China**
lu.pinyan@mail.shufe.edu.cn

2 **Zhiyuan College, Shanghai Jiao Tong University, No. 800 Dongchuan Road, Minhang District, Shanghai, China**
kuan.yang.6@gmail.com

3 **Department of Computer Science and Engineering, Shanghai Jiao Tong University, No. 800 Dongchuan Road, Minhang District, Shanghai, China**
chihao.zhang@gmail.com

Abstract

Hardcore and Ising models are two most important families of two state spin systems in statistic physics. Partition function of spin systems is the center concept in statistic physics which connects microscopic particles and their interactions with their macroscopic and statistical properties of materials such as energy, entropy, ferromagnetism, etc. If each local interaction of the system involves only two particles, the system can be described by a graph. In this case, fully polynomial-time approximation scheme (FPTAS) for computing the partition function of both hardcore and anti-ferromagnetic Ising model was designed up to the uniqueness condition of the system. These result are the best possible since approximately computing the partition function beyond this threshold is NP-hard. In this paper, we generalize these results to general physics systems, where each local interaction may involves multiple particles. Such systems are described by hypergraphs. For hardcore model, we also provide FPTAS up to the uniqueness condition, and for anti-ferromagnetic Ising model, we obtain FPTAS under a slightly stronger condition.

1998 ACM Subject Classification F.2.2 Computations on Discrete Structures

Keywords and phrases hard-core model, ising model, hypergraph, spatial mixing, correlation decay

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.51

1 Introduction

In recent couple of years, there are remarkable progress on designing approximate counting algorithms based on correlation decay approach [26, 1, 7, 22, 13, 23, 14, 20, 15, 19, 18, 17, 16]. Unlike the previous major approximate counting approach that based on random sampling such as Markov Chain Monte Carlo (MCMC) (see for examples [11, 10, 12, 8, 3, 9, 25, 4, 5, 21]), correlation decay based approach provides deterministic fully polynomial-time approximation scheme (FPTAS). New FPTASes were designed for a number of interesting combinatorial counting problems and computing partition functions for statistic physics systems, where partition function is a weighted counting function from the computational point of view. One

* The full version of the paper can be found at http://arxiv.org/abs/1509.05494.


© Pinyan Lu, Kuan Yang, and Chihao Zhang;
licensed under Creative Commons License CC-BY
33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 51; pp. 51:1–51:14

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


most successful example is the algorithm for anti-ferromagnetic two-spin systems [13, 23, 14], including counting independent sets [26]. The correlation decay based FPTAS is beyond the best known MCMC based FPRAS and achieves the boundary of approximability [24, 6].

In this paper, we generalize these results of anti-ferromagnetic two-spin systems to hypergraphs. For physics point of view, this corresponds to spin systems with higher order interactions, where each local interaction involves more than two particles. There are two main ingredients for the original algorithms and analysis on normal graphs (we will use the term normal graph for a graph to emphasize that it is not hypergraphs): (1) the construction of the self-avoiding walk tree by Weitz [26], which transform a general graph to a tree; (2) correlation decay proof for the tree, which enables one to truncate the tree to get a good approximation in polynomial time. However, the construction of the self-avoiding walk tree cannot be extended to hypergraphs, which is the main obstacle for the generalization.

The most related previous work is counting independent sets for hypergraphs by Liu and Lu [17]. They established a computation tree with a two-layers recursive function instead of the self-avoiding walk tree and provided a FPTAS to count the number of independent sets for hypergraphs with maximum degree of 5, extending the algorithm for normal graph with the same degree bound. Their proof was significantly more complicated than the previous one due to the complication of the two-layers recursive function. In particular, the "right" degree bound for the problem is a real number between 5 and 6 if one allow fraction degree in some sense. This integer gap provides some room of flexibility and enables them to do some case-by-case numerical argument to complete the proof. However, the parameters for the anti-ferromagnetic two-spin systems on hypergraphs are real numbers. To get a sharp threshold, we do not have any room for numerical approximation.

1.1 Our Results

We study two most important anti-ferromagnetic two-spin systems on hypergraphs: the hardcore model and the anti-ferromagnetic Ising model. The formal definitions of these two models can be found in Section 2.

Our first result is an FPTAS to compute the partition function of hypergraph hardcore model.

▶ **Theorem 1.** *For hardcore model with a constant activity parameter of λ, there is an FPTAS to compute the partition function for hypergraphs with maximum degree $\Delta \geq 2$ if $\lambda < \frac{(\Delta-1)^{\Delta-1}}{(\Delta-2)^{\Delta}}$.*

This bound is exactly the uniqueness threshold for the hardcore on normal graphs. Thus, it is tight since normal graphs are special cases of hypergraphs. To approximately compute the partition function beyond this threshold is NP-hard. In particular, The FPTAS in [17] for counting the number of independent sets for hypergraphs with maximum degree of 5 can be viewed as a special case of our result with parameters $\Delta = 5, \lambda = 1$, which satisfies the above uniqueness condition. Another interesting special case is when $\Delta = 2$. This is not an interesting case for normal graphs since a normal graph with maximum degree of 2 is simply a disjoint union of paths and cycles, whose partition function can be computed exactly. However, the problem becomes more complicated on hypergraphs: it can be interpreted as counting weighted edge covers on normal graphs by viewing vertices of degree two as edges and hyperedges as vertices. The exact counting of this problem is known to be #P-complete and an FPTAS was found recently [18]. In our model, the uniqueness bound $\frac{(\Delta-1)^{\Delta-1}}{(\Delta-2)^{\Delta}}$ is infinite for $\Delta = 2$ and as a result we give an FPTAS for counting weighted edge covers for any constant edge weight λ. This gives an alternative proof for the main result in [18].

Our second result is on computing the partition function of anti-ferromagnetic Ising model.

▶ **Theorem 2.** *For Ising model with interaction parameter $0 < \beta < 1$ and external field λ, there is an FPTAS to compute the partition function for hypergraphs with maximum degree Δ if $\beta \geq 1 - \frac{2}{2e^{-1/2}\Delta+3}$.*

The tight uniqueness bound for anti-ferromagnetic Ising model on normal graphs is $\beta \geq 1 - \frac{2}{\Delta}$. So, our bound is in the same asymptotic order but a bit worse in the constant coefficient as $2e^{-1/2} \approx 1.213 > 1$. Moreover, our result can apply beyond Ising model to a larger family of anti-ferromagnetic two-spin systems on hypergraphs.

1.2 Our Techniques

We also use the correlation decay approach. Although the framework of this method is standard, in many work along this line of research, new tools and techniques are developed to make this relatively new approach more powerful and widely applicable. This is indeed the case for the current paper as well. We summarize the new techniques we introduced here.

For hardcore model, we replace the numerical case-by-case analysis by a monotone argument with respect to the edge size of the hypergraph which shows that the normal graphs with edge size of 2 is indeed the worst cases. This gives a tight bound for hardcore model.

To handle hypergraph with unbounded edge sizes, we need to prove that the decay rate is much smaller for edges of larger size. Such effect is called computationally efficient correlation decay, which has been used in many previous works to obtain FPTASes for systems with unbounded degrees or edge sizes. In all those works, one sets a threshold for the parameter and proves different types of bounds for large and small ones separately. Such artificial separation gets a discontinuous bound which adds some complications in the proof and usually ends with a case-by-case discussion. In particular, this separation is not compatible with the above monotone argument. To overcome this, we propose a new uniform and smooth treatment for this by modifying the decay rate by a polynomial function of the edge size. After this modification, we only need to prove one single bound which automatically provides computationally efficient correlation. We believe that this idea is important and may find applications in other related problems.

For the Ising model, the main difficulty is to get a computation tree as a replacement of the self-avoiding walk tree. We proposed one, which also works for general anti-ferromagnetic two-spin systems on hypergraphs. However, unlike the case of the hardcore model, the computation tree is not of perfect efficiency and this is the main reason that the bound we achieve in Theorem 2 is not tight. To get the computationally efficient correlation decay, we also use the above mentioned uniform and smooth treatment. We also extend our result beyond Ising to a family of anti-ferromagnetic two-spin systems on hypergraphs.

1.3 Discussion and Open Problems

One obvious open question is to close the gap for Ising model, or more generally extend our work to anti-ferromagnetic two-spin systems on hypergraphs with better parameters. However, it seems that it is impossible to obtain a tight result in these models using the computation tree proposed in this paper, due to its imperfectness. How to overcome this is an important open question.

Even for the hardcore model, our result is tight only for the family of all hypergraphs since the normal graphs are special cases. From both physics and combinatorics point of view, it would be very interesting to study the family of w-uniform hypergraphs where each hyperedge is of the same size w. By our monotone argument, it is plausible to conjecture that one can get better bound for larger w. In particular, MCMC based approach does show that larger edge size helps: for hypergraph independent set with maximum degree of Δ and minimum edge size w, an FPRAS for $w \geq 2\Delta + 1$ was shown in [2]. However, their result is not tight. Can we get a tight bounds in terms of Δ and w by correlation decay approach? The high level idea sounds promising, but there is an obstacle to prove such result by our computation tree. To construct the computation tree, we need to construct modified instances. In these modified instances, the size of a hyperedge may decrease to as low as 2. Therefore, even if we start with w-uniform hypergraphs or hypergraphs with minimum edge size of w, we may need to handle the worst case of normal graphs during the analysis. How to avoid this effect is a major open question whose solution may have applications in many other problems.

The fact that larger hyperedge size only makes the problem easier is not universally true for approximation counting. One interesting example is counting hypergraph matchings. FPTAS for counting 3D matchings of hypergraphs with maximum degree 4 is given in [17], and extension to weighted setting are studied in [27]. In particular, a uniqueness condition in this setting is defined in [27], and it is a very interesting open question whether this uniqueness condition is also the transition boundary for approximability.

2 Preliminaries

A hypergraph $G(V, \mathcal{E})$ consists of a vertex set V and a set of hyperedges $\mathcal{E} \subseteq 2^V$. For every hyperedge $e \in \mathcal{E}$ and vertex $v \in V$, we use $e - v$ to denote $e \setminus \{v\}$ and use $e + v$ to denote $e \cup \{v\}$.

2.1 Hypergraph Hardcore Model

The hardcore model is parameterized by the activity parameter $\lambda > 0$. Let $G(V, \mathcal{E})$ be a hypergraph. An independent set of G is a vertex set $I \subseteq V$ such that $e \nsubseteq I$ for every hyperedge $e \in \mathcal{E}$. We use $\mathcal{I}(G)$ to denote the set of independent sets of G. The weight of an independent set I is defined as $w(I) \triangleq \lambda^{|I|}$. We let $Z(G)$ denote the partition function of $G(V, \mathcal{E})$ in the hardcore model, which is defined as

$$Z(G) \triangleq \sum_{I \in \mathcal{I}(G)} w(I).$$

The weight of independent sets induces a Gibbs measure on G. For every $I \in \mathcal{I}$, we use

$$\mathbf{Pr}_G[I] \triangleq \frac{w(I)}{Z(G)}$$

to denote the probability of obtaining I if we sample according to the Gibbs measure. For every $v \in V$, we use

$$\mathbf{Pr}_G[v \in I] \triangleq \sum_{\substack{I \in \mathcal{I}(G) \\ v \in I}} \mathbf{Pr}_G[I]$$

to denote the *marginal probability* of v.

2.2 Hypergraph Two State Spin Model

Now we give a formal definition to hypergraph two state spin systems. This model is parameterized by the external field $\lambda > 0$. An instance of the model is a labeled hypergraph $G(V, \mathcal{E}, (\boldsymbol{\beta}, \boldsymbol{\gamma}))$ where $\boldsymbol{\beta}, \boldsymbol{\gamma} : \mathcal{E} \to \mathbb{R}$ are two labeling functions that assign each edge $e \in \mathcal{E}$ two reals $\boldsymbol{\beta}(e), \boldsymbol{\gamma}(e)$. A configuration on G is an assignment $\sigma : V \to \{0, 1\}$ whose weight $w(\sigma)$ is defined as

$$w(\sigma) \triangleq \prod_{e \in \mathcal{E}} w(e, \sigma) \prod_{v \in V} w(v, \sigma)$$

where for a hyperedge $e = \{v_1, \ldots, v_w\}$

$$w(e, \sigma) \triangleq \begin{cases} \boldsymbol{\beta}(e) & \text{if } \sigma(v_1) = \sigma(v_2) = \cdots = \sigma(v_w) = 0 \\ \boldsymbol{\gamma}(e) & \text{if } \sigma(v_1) = \sigma(v_2) = \cdots = \sigma(v_w) = 1 \\ 1 & \text{otherwise} \end{cases}$$

and for a vertex v,

$$w(v, \sigma) \triangleq \begin{cases} \lambda & \text{if } \sigma(v) = 1 \\ 1 & \text{otherwise.} \end{cases}$$

The partition function of the instance is given by

$$Z(G) = \sum_{\sigma \in \{0,1\}^V} w(\sigma).$$

Similarly, the weight of configurations induces a Gibbs measure on G. For every $\sigma \in \{0, 1\}^V$, we use

$$\mathbf{Pr}_G[\sigma] \triangleq \frac{w(I)}{Z(G)}$$

to denote the probability of σ in the measure. For every $v \in V$, we use

$$\mathbf{Pr}_G[\sigma(v) = 1] \triangleq \sum_{\substack{\sigma \in \{0,1\}^V \\ \sigma(v)=1}} \mathbf{Pr}_G[\sigma]$$

to denote the *marginal probability* of v.

The *anti-ferromagnetic* Ising model is the special case that $\beta \triangleq \boldsymbol{\beta}(e) = \boldsymbol{\gamma}(e) \leq 1$ for all $e \in \mathcal{E}$. In this model, we call β the *interaction parameter* of the model. The hardcore model introduced in previous section is the special case that $\boldsymbol{\beta}(e) = 1$ and $\boldsymbol{\gamma}(e) = 0$ for all $e \in \mathcal{E}$.

The whole proof of Theorem 2 can be found in the full version of the paper. More precisely, we design an FPTAS for the more general two state spin system and establish the following theorem:

▶ **Theorem 3.** *Consider a class of two state spin system with external field λ such that each instance $G(V, \mathcal{E}, (\boldsymbol{\beta}, \boldsymbol{\gamma}))$ in the class satisfies $1 - \frac{2}{2e^{-1/2}\Delta+3} \leq \boldsymbol{\beta}(e), \boldsymbol{\gamma}(e) \leq 1$ where Δ is the maximum degree of G. There exists an FPTAS to compute the partition function for every instance in the class.*

Theorem 2 then follows since it is a special case of Theorem 3.

Actually, the main idea of FPTAS design and proof for this model is similar to the idea we use to solve hypergraph hardcore model. However, the details of recursion function design and techniques for proof of correlation decay property are pretty different from that in hypergraph hardcore model, see the full version of the paper for details.

3 Hypergraph Hardcore Model

3.1 Recursion for Computing Marginal Probability

We first fix some notations on graph modification specific to hypergraph independent set. Let $G(V,\mathcal{E})$ be a hypergraph.

- For every $v \in V$, we denote $G - v \triangleq (V \setminus \{v\}, \mathcal{E}')$ where $\mathcal{E}' \triangleq \{e \setminus \{v\} \mid e \in \mathcal{E}\}$.
- For every $e \in \mathcal{E}$, we denote $G - e \triangleq (V, \mathcal{E} \setminus \{e\})$.
- Let x be a vertex or an edge and y be a vertex or an edge, we denote $G - x - y \triangleq (G-x)-y$.
- Let $S = \{v_1, \ldots, v_k\} \subseteq V$, we denote $G - S \triangleq G - v_1 - v_2 \cdots - v_k$.
- Let $\mathcal{F} = \{e_1, \ldots, e_k\} \subseteq \mathcal{E}$, we denote $G - \mathcal{F} \triangleq G - e_1 - e_2 \cdots - e_k$.

Let $G(V,\mathcal{E})$ be a hypergraph and $v \in V$ be an arbitrary vertex with degree d. Let $\{e_1, \ldots, e_d\}$ be the set of hyperedges incident to v and for every $i \in [d]$, $e_i = \{v\} \cup \{v_{ij} \mid j \in [w_i]\}$ consists of $w_i + 1$ vertices.

We first define a hypergraph $G'(V', \mathcal{E}')$, which is the graph obtained from G by replacing v by d copies of itself and each e_i contains a distinct copy. Formally, $V' \triangleq (V \setminus \{v\}) \cup \{v_1, \ldots, v_d\}$, $\mathcal{E}' \triangleq \{e \in \mathcal{E} \mid v \notin e\} \cup \{e_i - v + v_i \mid i \in [d]\}$.

For every $i \in [d]$ and $j \in [w_i]$, we define a hypergraph $G_{ij}(V_{ij}, \mathcal{E}_{ij})$:

$$G_{ij} \triangleq G' - \{v_k \mid i \le k \le d\} - \{e_k \mid 1 \le k \le i\} - \{v_{ik} \mid 1 \le k < j\}.$$

Let $R_v = \frac{\mathbf{Pr}_G[v \in I]}{\mathbf{Pr}_G[v \notin I]}$ and $R_{ij} = \frac{\mathbf{Pr}_{G_{ij}}[v_{ij} \in I]}{\mathbf{Pr}_{G_{ij}}[v_{ij} \notin I]}$. We can compute R_v by following recursion:

▶ **Lemma 4.**

$$R_v = \lambda \prod_{i=1}^{d} \left(1 - \prod_{j=1}^{w_i} \frac{R_{ij}}{1 + R_{ij}}\right). \tag{1}$$

Proof. By the definition of R_v, we have

$$R_v = \frac{\mathbf{Pr}_G[v \in I]}{\mathbf{Pr}_G[v \notin I]} = \lambda \cdot \frac{\mathbf{Pr}_{G'}\left[\bigwedge_{i=1}^{d} v_i \in I\right]}{\mathbf{Pr}_{G'}\left[\bigwedge_{i=1}^{d} v_i \notin I\right]}$$
$$= \lambda \cdot \prod_{i=1}^{d} \frac{\mathbf{Pr}_{G'}\left[v_i \in I \wedge \bigwedge_{j=1}^{i-1} v_j \notin I \wedge \bigwedge_{j=i+1}^{d} v_j \in I\right]}{\mathbf{Pr}_{G'}\left[v_i \notin I \wedge \bigwedge_{j=1}^{i-1} v_j \notin I \wedge \bigwedge_{j=i+1}^{d} v_j \in I\right]}$$

For every $i \in [d]$, define $G_i \triangleq G' - \{v_k \mid i < k \le d\} - \{e_k \mid 1 \le k < i\}$, we have

$$\frac{\mathbf{Pr}_{G'}\left[v_i \in I \wedge \bigwedge_{j=1}^{i-1} v_i \notin I \wedge \bigwedge_{j=i+1}^{d} v_j \in I\right]}{\mathbf{Pr}_{G'}\left[v_i \notin I \wedge \bigwedge_{j=1}^{i-1} v_j \notin I \wedge \bigwedge_{j=i+1}^{d} v_j \in I\right]} = \frac{\mathbf{Pr}_{G'}\left[v_i \in I \;\middle|\; \bigwedge_{j=1}^{i-1} v_j \notin I \wedge \bigwedge_{j=i+1}^{d} v_j \in I\right]}{\mathbf{Pr}_{G'}\left[v_i \notin I \;\middle|\; \bigwedge_{j=1}^{i-1} v_j \notin I \wedge \bigwedge_{j=i+1}^{d} v_j \in I\right]}$$
$$= \frac{\mathbf{Pr}_{G_i}[v_i \in I]}{\mathbf{Pr}_{G_i}[v_i \notin I]}.$$

This is because fixing $v_j \in I$ is equivalent to removing v_j from the graph and fixing $v_j \notin I$ is equivalent to removing all edges incident to v_j from the graph.

Since e_i is the unique hyperedge in G_i that contains v_i, we have

$$\frac{\mathbf{Pr}_{G_i}[v_i \in I]}{\mathbf{Pr}_{G_i}[v_i \notin I]} = 1 - \mathbf{Pr}_{G_i - v_i - e_i}\left[\bigwedge_{j=1}^{w_i} v_{ij} \in I\right] = 1 - \prod_{j=1}^{w_i} \mathbf{Pr}_{G_{ij}}[v_{ij} \in I] = 1 - \prod_{j=1}^{w_i} \frac{R_{ij}}{1 + R_{ij}}.$$

◀

The Uniqueness Condition

Let the underlying graph be an infinite d-ary tree, then the recursion (1) becomes

$$f_{\lambda,d}(x) = \lambda\left(\frac{1}{1+x}\right)^d.$$

Let $\hat{x}$ be the positive fixed-point of $f_{\lambda,d}(x)$, i.e., $\hat{x} > 0$ and $f_{\lambda,d}(\hat{x}) = \hat{x}$. The condition on λ for the uniqueness of the Gibbs measure is that $\left|f'_{\lambda,d}(\hat{x})\right| < 1$. The following proposition is well-known.

▶ **Proposition 5.** *Let* $\lambda_c = \frac{d^d}{(d-1)^{d+1}}$, *then* $\left|f'_{\lambda_c,d}(\hat{x})\right| = 1$ *and for every* $0 < \lambda < \lambda_c$, *it holds that* $\left|f'_{\lambda,d}(\hat{x})\right| < 1$.

3.2 The Algorithm to Compute Marginal Probability

Let $G(V,\mathcal{E})$ be a hypergraph with maximum degree Δ and $v \in V$ be an arbitrary vertex with degree d. Define G_{ij}, R_v, R_{ij} as in Section 3.1. Then the recursion (1) gives a way to compute the marginal probability $\mathbf{Pr}_G[v \in I]$ *exactly*. However, an exact evaluation of the recursion requires a computation tree with exponential size. Thus we introduce the following truncated version of the recursion, with respect to constants $c > 0$ and $0 < \alpha < 1$.

$$R(G,v,L) = \begin{cases} \lambda\prod_{i=1}^{d}\left(1-\prod_{j=1}^{w_i}\frac{R(G_{ij},v_{ij},L)}{1+R(G_{ij},v_{ij},L)}\right) & \text{if } d=\Delta \\ \lambda\prod_{i=1}^{d}\left(1-\prod_{j=1}^{w_i}\frac{R(G_{ij},v_{ij},L-\lfloor 1+c\log_{1/\alpha} w_i\rfloor)}{1+R(G_{ij},v_{ij},L-\lfloor 1+c\log_{1/\alpha} w_i\rfloor)}\right) & \text{if } d<\Delta \text{ and } L>0 \\ \lambda & \text{otherwise.} \end{cases}$$

The recursion can be directly used to compute $R(G,v,L)$ for any given L and it induces a truncated computation tree (with height L in some special metric). It is worth noting that, the case that $d = \Delta$ can only happen at the root of the computation tree, since in each smaller instance, the degree of v_{ij} is decreased by at least one.

We claim that $R(G,v,L)$ is a good estimate of R_v with a suitable choice of c and α, for those (λ,Δ) in the uniqueness region.

▶ **Lemma 6.** *Let* $G(V,\mathcal{E})$ *be a hypergraph with maximum degree* $\Delta \geq 2$. *Let* $v \in V$ *be a vertex with degree* d *and let* $\lambda < \lambda_c = \frac{(\Delta-1)^{\Delta-1}}{(\Delta-2)^{\Delta}}$ *be the activity parameter. There exist constants* $C > 0$ *(more precisely,* $C = 6\lambda\sqrt{1+\lambda}$*) and* $\alpha \in (0,1)$ *such that*

$$|R(G,v,L) - R_v| \leq C\cdot\alpha^{\max\{0,L\}}$$

for every L.

The whole proof of this lemma is postponed to the next section. Now we can prove Theorem 1 via using this lemma.

Proof of Theorem 1. The input of the FPTAS is an instance $G(V,\mathcal{E})$ and an accuracy parameter $0 < \varepsilon < 1/2$. Assume $V = \{v_1,\ldots,v_n\}$. Note that $I = \varnothing$ is an independent set of G with $w(I) = 1$. Therefore

$$Z(G) = 1/\mathbf{Pr}_G[I] = \left(\mathbf{Pr}_G\left[\bigwedge_{i=1}^{n} v_i \notin I\right]\right)^{-1} = \left(\prod_{i=1}^{n}\mathbf{Pr}_G\left[v_i \notin I \,\middle|\, \bigwedge_{j=1}^{i-1} v_j \notin I\right]\right)^{-1}.$$

For every $1 \leq i \leq n$, we define a graph $G_i(V_i,E_i)$:

- $G_1 \triangleq G$;
- For every $i \geq 2$, $G_i \triangleq G_{i-1} - v_{i-1} - \mathcal{E}'$ where $\mathcal{E}' \triangleq \{e \in \mathcal{E}_{i-1} \mid v_{i-1} \in e\}$ consists of edges in G_{i-1} incident to v_i.

It is straightforward to verify that $\mathbf{Pr}_G\left[v_i \notin I \,\middle|\, \bigwedge_{j=1}^{i-1} v_j \notin I\right] = \mathbf{Pr}_{G_i}\left[v_i \notin I\right]$ for every $1 \leq i \leq n$. Thus,

$$Z(G) = \prod_{i=1}^{n} \left(\mathbf{Pr}_{G_i}\left[v_i \notin I\right]\right)^{-1} = \prod_{i=1}^{n} \left(1 + R_i\right),$$

where $R_i \triangleq \frac{\mathbf{Pr}_{G_i}[v_i \in I]}{\mathbf{Pr}_{G_i}[v_i \notin I]}$. Let C and α be constants in Lemma 6. We compute $R(G_i, v_i, L)$ with $L = \frac{\log\left(2Cn\varepsilon^{-1}\right)}{\log \alpha^{-1}}$ for every $1 \leq i \leq n$, then $|R_i - R(G_i, v_i, L)| \leq \frac{\varepsilon}{2n}$. This implies

$$1 - \frac{\varepsilon}{2n} \leq \frac{1 + R_i}{1 + R(G, v_i, L)} \leq 1 + \frac{\varepsilon}{2n}.$$

Let $\hat{Z} = \prod_{i=1}^{n} \left(1 + R(G_i, v_i, L)\right)^{-1}$ be our estimate of the partition function, then it holds that

$$e^{-\varepsilon} \leq \frac{Z(G)}{\hat{Z}} \leq e^{\varepsilon}.$$

It remains to bound the running time of our algorithm. Let $T(L)$ denote the maximum running time of computing $R(G, v, L)$ (over all choices of $d \leq \Delta$ and arbitrary w_i). Then by the definition of $R(G, v, L)$, for every $L > 0$,

$$T(L) \leq \sum_{i=1}^{d} \sum_{j=1}^{w_i} T(L - \lfloor 1 + c \log_{1/\alpha} w_i \rfloor) + O(n).$$

It is easy to verify that $T(L) = n\Delta^{O(L)} = \left(\frac{n}{\varepsilon}\right)^{O(\log \Delta)}$ for our choice of L. Thus our algorithm is an FPTAS for computing $Z(G)$. ◀

3.3 Correlation Decay

In this section, we establish Lemma 6. We first prove some technical lemmas.

Suppose $f : D^d \to \mathbb{R}$ is a d-ary function where $D \subseteq \mathbb{R}$ is a convex set, let $\phi : \mathbb{R} \to \mathbb{R}$ be an increasing differentiable function and $\Phi(x) \triangleq \phi'(x)$. The following proposition is a consequence of the mean value theorem:

▶ **Proposition 7.** *For every $\mathbf{x} = (x_1, \ldots, x_d)$, $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_d) \in D^d$, it holds that*

1. $|f(\mathbf{x}) - f(\hat{\mathbf{x}})| = \frac{1}{\Phi(\tilde{x})} |\phi(f(\mathbf{x})) - \phi(f(\hat{\mathbf{x}}))|$ *for some $\tilde{x} \in D$;*
2. $|\phi(f(\mathbf{x})) - \phi(f(\hat{\mathbf{x}}))| \leq \sum_{i=1}^{d} \frac{\Phi(f)}{\Phi(\tilde{x}_i)} \left|\frac{\partial f(\tilde{\mathbf{x}})}{\partial x_i}\right| \cdot |\phi(x_i) - \phi(\hat{x}_i)|$ *for some $\tilde{\mathbf{x}} = (\tilde{x}_1, \ldots, \tilde{x}_d) \in D^d$.*

▶ **Lemma 8.** *Let $\Delta \geq 2$ be a constant integer and $\lambda < \lambda_c = \frac{(\Delta-1)^{\Delta-1}}{(\Delta-2)^{\Delta}}$ be a constant real. Let $d < \Delta$ and $w_1, \ldots, w_d > 0$ be integers and $f = \lambda \prod_{i=1}^{d} \left(1 - \prod_{j=1}^{w_i} \frac{x_{ij}}{1+x_{ij}}\right)$ be a $\left(\sum_{i=1}^{d} w_i\right)$-ary function. Let $\Phi(x) = \frac{1}{\sqrt{x(1+x)}}$. Let $c < \min\left\{\frac{\log(1+\lambda) - \log \lambda}{2+4\lambda}, \frac{2\lambda+1}{2} \log\left(\frac{1+\lambda}{\lambda}\right) - 1\right\}$ be a positive number. There exists a constant $\alpha < 1$ depending on λ and d (but not depending on w_i for all $i \in [d]$) such that*

$$\sum_{a=1}^{d} w_a^c \sum_{b=1}^{w_a} \frac{\Phi(f)}{\Phi(x_{ab})} \left|\frac{\partial f(\mathbf{x})}{\partial x_{ab}}\right| \leq \alpha < 1$$

for every $\mathbf{x} = (x_{ij})_{i \in [d], j \in [w_i]}$ where each $x_{ij} \in [0, \lambda]$

The lemma bounds the amortized decay rate, which is the key to the proof of correlation decay. In previous works, the amortized decay rate is defined as

$$\sum_{a=1}^{d}\sum_{b=1}^{w_a}\frac{\Phi(f)}{\Phi(x_{ab})}\left|\frac{\partial f(\mathbf{x})}{\partial x_{ab}}\right|,$$

without the w_a^c factor. Then one need to give a constant $\alpha<1$ bound for small w_a and a sub constant bound for large w_a. With this modification, we only need to prove a single bound as above.

Notice that we require c to be a positive constant, so it is necessary to verify that $\frac{2\lambda+1}{2}\log\left(\frac{1+\lambda}{\lambda}\right)-1>0$ for every $\lambda>0$. To see this, let $h(\lambda)\triangleq\frac{2\lambda+1}{2}\log\left(\frac{1+\lambda}{\lambda}\right)-1$, then we can compute that

$$h'(\lambda)=\log\left(\frac{1+\lambda}{\lambda}\right)-\frac{1+2\lambda}{2\lambda+2\lambda^2},$$
$$h''(\lambda)=\frac{1}{2\lambda^2(1+\lambda)^2}.$$

Since $h''(\lambda)>0$ for every λ, $h'(\lambda)$ is increasing. Along with the fact that $\lim_{\lambda\to\infty}h'(\lambda)=0$, we have $h'(\lambda)<0$ for every $\lambda>0$. This implies that $h(\lambda)$ is decreasing. Also note that

$$\lim_{\lambda\to\infty}h(\lambda)=\lim_{\lambda\to\infty}\log\left(\left(1+\frac{1}{\lambda}\right)^{\lambda}\left(1+\frac{1}{\lambda}\right)^{1/2}\right)-1=0.$$

It holds that $h(\lambda)>0$ for every $\lambda>0$. Thus a positive c satisfying $c<h(\lambda)$ exists for every $\lambda>0$.

Proof of Lemma 8. To simplify the notation, we first let $t_{ij}=\frac{x_{ij}}{1+x_{ij}}$, then for every $i\in[d]$ and $j\in[w_i]$, it holds that $t_{ij}\in\left[0,\frac{\lambda}{1+\lambda}\right]$ and

$$f=\lambda\prod_{i=1}^{d}\left(1-\prod_{j=1}^{w_i}t_{ij}\right).$$

For every $a\in[d]$ and $b\in[w_i]$, we have

$$\left|\frac{\partial f}{\partial x_{ab}}\right|=\lambda(1-t_{ab})^2\prod_{\substack{j\in[w_a]\\ j\neq b}}t_{aj}\cdot\prod_{\substack{i\in[d]\\ i\neq a}}\left(1-\prod_{j=1}^{w_i}t_{ij}\right)=f\cdot\frac{(1-t_{ab})^2}{t_{ab}}\cdot\frac{\prod_{j=1}^{w_a}t_{aj}}{1-\prod_{j=1}^{w_a}t_{aj}}.$$

Thus

$$\sum_{a=1}^{d}w_a^c\sum_{b=1}^{w_a}\frac{\Phi(f)}{\Phi(x_{ab})}\left|\frac{\partial f}{\partial x_{ab}}\right|=\sqrt{\frac{f}{1+f}}\sum_{a=1}^{d}\frac{w_a^c\prod_{j=1}^{w_a}t_{aj}}{1-\prod_{j=1}^{w_a}t_{aj}}\sum_{b=1}^{w_a}\frac{1-t_{ab}}{\sqrt{t_{ab}}}.$$

Let $\mathbf{t}=(t_{ij})_{i\in[d],j\in[w_i]}$, define

$$\begin{aligned}h(\mathbf{t})&\triangleq\sqrt{\frac{f}{1+f}}\sum_{a=1}^{d}\frac{w_a^c\prod_{j=1}^{w_a}t_{aj}}{1-\prod_{j}^{w_a}t_{aj}}\sum_{b=1}^{w_a}\frac{1-t_{ab}}{\sqrt{t_{ab}}}\\&=\sqrt{\frac{\lambda\prod_{i=1}^{d}\left(1-\prod_{j=1}^{w_i}t_{ij}\right)}{1+\lambda\prod_{i=1}^{d}\left(1-\prod_{j=1}^{w_i}t_{ij}\right)}}\sum_{a=1}^{d}\frac{w_a^c\prod_{j=1}^{w_a}t_{aj}}{1-\prod_{j}^{w_a}t_{aj}}\sum_{b=1}^{w_a}\frac{1-t_{ab}}{\sqrt{t_{ab}}}.\end{aligned}$$

For every $\mathbf{t} = (t_{ij})_{i\in[d],j\in[w_i]}$ where each $t_{ij} \in [0, \frac{\lambda}{1+\lambda}]$, define a tuple $\hat{\mathbf{t}} = (\hat{t}_{ij})_{i\in[d],j\in w_i}$ such that for every $i \in [d]$,

$$\hat{t}_{ij} = \begin{cases} \left(\frac{1+\lambda}{\lambda}\right)^{w_i-1} \prod_{k=1}^{w_i} t_{ik} & \text{if } j = 1 \\ \frac{\lambda}{1+\lambda} & \text{otherwise.} \end{cases}$$

We claim that $h(\mathbf{t}) \le h(\hat{\mathbf{t}})$. To see this, first note that for every $i \in [d]$, $\prod_{j=1}^{w_i} t_{ij} = \prod_{j=1}^{w_i} \hat{t}_{ij}$, it is sufficient to prove that for every $i \in [d]$, $\sum_{j=1}^{w_i} \frac{1-t_{ij}}{\sqrt{t_{ij}}} \le \sum_{j=1}^{w_i} \frac{1-\hat{t}_{ij}}{\sqrt{\hat{t}_{ij}}}$. This is a consequence of the Karamata's inequality by noticing that the function $\frac{1-e^x}{\sqrt{e^x}}$ is convex.

We rename $\hat{t}_{i1}$ to t_i and it is sufficient to upper bound

$$g(\mathbf{t}, \mathbf{w}) \triangleq \sqrt{\frac{\lambda \prod_{i=1}^{d} \left(1 - \left(\frac{\lambda}{1+\lambda}\right)^{w_i-1} t_i\right)}{1 + \lambda \prod_{i=1}^{d} \left(1 - \left(\frac{\lambda}{1+\lambda}\right)^{w_i-1} t_i\right)}} \cdot \sum_{i=1}^{d} \frac{w_i^c \left(\frac{\lambda}{1+\lambda}\right)^{w_i-1} t_i}{1 - \left(\frac{\lambda}{1+\lambda}\right)^{w_i-1} t_i} \cdot \left(\frac{1-t_i}{\sqrt{t_i}} + \frac{(w_i-1)}{\sqrt{\lambda+\lambda^2}}\right) \tag{2}$$

where $t_i \in \left[0, \frac{\lambda}{1+\lambda}\right]$ and $w_i \in \mathbb{Z}^+$ for every $i \in [d]$.

The argument so far is similar to the proof in [17]. In the following, we prove a monotonicity property of each w_i and thus avoid the heavy numerical analysis in [17] and allow us to obtain a tight result.

For every $i \in [d]$, we let $z_i \triangleq 1 - \left(\frac{\lambda}{1+\lambda}\right)^{w_i-1} t_i$ and thus equivalently $t_i = (1-z_i)\left(\frac{1+\lambda}{\lambda}\right)^{w_i-1}$. For every fixed $\mathbf{z} = (z_1, \ldots, z_d)$, we can write (2) as

$$g_{\mathbf{z}}(\mathbf{w}) = \sqrt{\frac{\lambda \prod_{i=1}^{d} z_i}{1 + \lambda \prod_{i=1}^{d} z_i}} \sum_{i=1}^{d} \frac{1-z_i}{z_i} \left(\frac{1-t_i}{\sqrt{t_i}} + \frac{w_i-1}{\sqrt{\lambda+\lambda^2}}\right) w_i^c. \tag{3}$$

We show that $g_{\mathbf{z}}(\mathbf{w})$ is monotonically decreasing with w_i for every $i \in [d]$.

Denote $T_i \triangleq \frac{1-t_i}{\sqrt{t_i}} + \frac{(w_i-1)}{\sqrt{\lambda+\lambda^2}}$, then

$$\frac{\partial g_{\mathbf{z}}(\mathbf{w})}{\partial w_i} = \sqrt{\frac{\lambda \prod_{i=1}^{d} z_i}{1 + \lambda \prod_{i=1}^{d} z_i}} \cdot \frac{1-z_i}{z_i} \left(\frac{\partial T_i}{\partial w_i} w_i^c + c w_i^{c-1} T_i\right). \tag{4}$$

The partial derivative (4) is negative for a suitable choice of c:

$$\begin{aligned}
&\frac{1-z_i}{z_i} \cdot \left(\frac{\partial T_i}{\partial z_i} w_i^c + c w_i^{c-1} T_i\right) \\
&= \frac{1-z_i}{z_i} \cdot \left(\left(-\frac{1}{2} t_i'(t_i^{-1/2} + t_i^{-3/2}) + \frac{1}{\sqrt{\lambda+\lambda^2}}\right) w_i^c + c w_i^{c-1} \left(\frac{1-t_i}{\sqrt{t_i}} + \frac{(w_i-1)}{\sqrt{\lambda+\lambda^2}}\right)\right) \\
&= \frac{1-z_i}{z_i} \cdot w_i^{c-1} \left(\left(-\frac{1}{2} \log\left(\frac{1+\lambda}{\lambda}\right)(t_i^{1/2} + t_i^{-1/2}) + \frac{1}{\sqrt{\lambda+\lambda^2}}\right) w_i \right. \\
&\qquad \left. + c\left(t_i^{-1/2} - t_i^{1/2} + \frac{(w_i-1)}{\sqrt{\lambda+\lambda^2}}\right)\right) \\
&= \frac{1-z_i}{z_i} \cdot w_i^{c-1} \left(\frac{(c+1)w_i - c}{\sqrt{\lambda+\lambda^2}} - \right. \\
&\qquad \left. \left(t_i^{1/2}\left(\frac{1}{2} w_i \log\left(\frac{1+\lambda}{\lambda}\right) + c\right) + t_i^{-1/2}\left(\frac{1}{2} w_i \log\left(\frac{1+\lambda}{\lambda}\right) - c\right)\right)\right)
\end{aligned}$$

Denote

$$p(t,w) \triangleq \frac{(c+1)w-c}{\sqrt{\lambda+\lambda^2}} - \left(t^{1/2}\left(\frac{1}{2}w\log\left(\frac{1+\lambda}{\lambda}\right)+c\right)+t^{-1/2}\left(\frac{1}{2}w\log\left(\frac{1+\lambda}{\lambda}\right)-c\right)\right)$$

Since $c \le \frac{\log(1+\lambda)-\log\lambda}{2+4\lambda}$, the term

$$t^{1/2}\left(\frac{1}{2}w\log\left(\frac{1+\lambda}{\lambda}\right)+c\right)+t^{-1/2}\left(\frac{1}{2}w\log\left(\frac{1+\lambda}{\lambda}\right)-c\right)$$

achieves its minimum at $t=\frac{\lambda}{1+\lambda}$. Thus

$$p(t,w) \le p\left(\frac{\lambda}{1+\lambda},w\right) = \left(\frac{\lambda}{1+\lambda}\right)^{1/2}\left(\frac{c+1}{\lambda}-\frac{2\lambda+1}{2\lambda}\log\left(\frac{1+\lambda}{\lambda}\right)\right)w.$$

Moreover, $c < \frac{2\lambda+1}{2}\log\left(\frac{1+\lambda}{\lambda}\right)-1$ implies that $\frac{c+1}{\lambda} < \frac{2\lambda+1}{2\lambda}\log\left(\frac{1+\lambda}{\lambda}\right)$ holds, which consequently leads to $p\left(\frac{\lambda}{1+\lambda},1\right)<0$.

In all, we choose a positive constant $c < \min\left\{\frac{\log(1+\lambda)-\log\lambda}{2+4\lambda}, \frac{2\lambda+1}{2}\log\left(\frac{1+\lambda}{\lambda}\right)-1\right\}$, and this results in $p\left(\frac{\lambda}{1+\lambda},w\right) \le p\left(\frac{\lambda}{1+\lambda},1\right)<0$.

In light of the monotonicity of w_i's, for every fixed $\mathbf{z}$, $g_{\mathbf{z}}(\mathbf{w})$ achieves its maximum when $\mathbf{w}=\mathbf{1}$. Thus

$$\begin{aligned}\max_{\mathbf{t}\in\left[0,\frac{\lambda}{1+\lambda}\right]^d} g(\mathbf{t},\mathbf{w}) &= \max_{\substack{\mathbf{z}=(z_1,\ldots,z_d)\\ \forall i\in[d], z_i\in\left[1-\left(\frac{\lambda}{1+\lambda}\right)^{w_i-1},1\right]}} g_{\mathbf{z}}(\mathbf{w})\\ &\le \max_{\substack{\mathbf{z}=(z_1,\ldots,z_d)\\ \forall i\in[d], z_i\in\left[1-\left(\frac{\lambda}{1+\lambda}\right)^{w_i-1},1\right]}} g_{\mathbf{z}}(\mathbf{1}) \le \max_{\mathbf{z}\in[0,1]^d} g_{\mathbf{z}}(\mathbf{1}).\end{aligned}$$

Actually, the case that all w_i's are 1 corresponds to counting weighted independent sets on normal graphs and arguments to bound $g_{\mathbf{z}}(\mathbf{1})$ can be found in [14]. For the sake of completeness, we give a proof of $g_{\mathbf{z}}(\mathbf{1}) \le \alpha < 1$ (Lemma 9) below. ◀

▶ **Lemma 9.** *Let $\Delta > 1$, be a constant. Assume $\lambda < \lambda_c = \frac{(\Delta-1)^{\Delta-1}}{(\Delta-2)^{\Delta}}$ be a constant and $d < \Delta$. Then for some constant $\alpha < 1$, $g_{\mathbf{z}}(\mathbf{1}) \le \alpha < 1$ where $\mathbf{z} = (z_1,\ldots,z_d) \in [0,1]^d$.*

Proof. Let $\lambda_c' \triangleq \frac{d^d}{(d-1)^{d+1}}$ be the uniqueness threshold for the d-ary tree. Then $\lambda < \lambda_c \le \lambda_c'$.

Plugging $\mathbf{w}=\mathbf{1}$ into (3), we have

$$g_{\mathbf{z}}(\mathbf{1}) = \sqrt{\frac{\lambda\prod_{i=1}^d z_i}{1+\lambda\prod_{i=1}^d z_i}}\cdot\sum_{i=1}^d\sqrt{1-z_i}.$$

Let $z = \left(\prod_{i=1}^d z_i\right)^{\frac{1}{d}}$, it follows from Jensen's inequality that

$$g(\mathbf{t},\mathbf{1}) \le d\sqrt{\frac{\lambda z^d(1-z)}{1+\lambda z^d}} < d\sqrt{\frac{\lambda_c' z^d(1-z)}{1+\lambda_c' z^d}} \tag{5}$$

Recall that $f_{\lambda,d}(x) = \lambda\left(\frac{1}{1+x}\right)^d$. Let $\hat{x}$ be the positive fixed-point of $f_{\lambda_c',d}(x)$ and $\hat{z} = \frac{1}{1+\hat{x}}$. We show that $d\sqrt{\frac{\lambda_c' z^d(1-z)}{1+\lambda_c' z^d}}$ achieves its maximum when $z=\hat{z}$. The derivative of $\frac{\lambda_c' z^d(1-z)}{1+\lambda_c' z^d}$ with respect to z is

$$\left(\frac{\lambda_c' z^d(1-z)}{1+\lambda_c' z^d}\right)' = -\frac{\lambda_c' z^{d-1}}{\left(1+\lambda_c' z^d\right)^2}\left(z+\lambda_c' z^{d+1}-d(1-z)\right).$$

Since $\lambda'_c = \frac{d^d}{(d-1)^{d+1}}$, the above achieves maximum at $\tilde{z} = \frac{d-1}{d}$. If we let $\tilde{x} = \frac{1-\tilde{z}}{\tilde{z}}$, then it is easy to verify that $f_{\lambda'_c,d}(\tilde{x}) = \tilde{x}$, which implies $\hat{z} = \tilde{z}$ because of the uniqueness of the positive fixed-point.

Therefore, we have for some $\alpha < 1$,

$$g_{\mathbf{z}}(\mathbf{1}) \le \alpha < d\sqrt{\frac{\lambda'_c \hat{z}^d(1-\hat{z})}{1+\lambda'_c \hat{z}^d}} = \left|f'_{\lambda'_c,d}(\hat{x})\right| = 1. \qquad \blacktriangleleft$$

We are now ready to prove the main lemma.

Proof of Lemma 6. Let $\Phi(x) = \frac{1}{\sqrt{x(1+x)}}$ and $\phi(x) = \int \Phi(x)\,\mathrm{d}x = 2\sinh^{-1}(\sqrt{x})$. We first apply induction on $\ell \triangleq \max\{0, L\}$ to show that, if $d < \Delta$, then $|\phi(R_v) - \phi(R(G, v, L))| \le 2\sqrt{\lambda}\alpha^L$ for some constant $\alpha < 1$.

If $\ell = 0$, note that $R_v \in [0, \lambda]$, thus $|\phi(R(G, v, L)) - \phi(R_v)| \le 2\sqrt{\lambda}$. We now assume $L = \ell > 0$ and the lemma holds for smaller ℓ. For every $i \in [d]$ and $j \in [w_i]$, we denote $x_{ij} = R_{ij}$ and $\hat{x}_{ij} = R(G_{ij}, v_{ij}, L - \lfloor 1 + c\log_{1/\alpha} w_i \rfloor)$. Let $\mathbf{x} = (x_{ij})_{i\in[d],j\in[w_i]}$, $\hat{\mathbf{x}} = (\hat{x}_{ij})_{i\in[d],j\in[w_i]}$. Let $f = \lambda\prod_{i=1}^d \left(1 - \prod_{j=1}^{w_i} \frac{x_{ij}}{1-x_{ij}}\right)$, then it follows from Proposition 7 that for some $\tilde{\mathbf{x}} = (\tilde{x}_{ij})_{i\in[d],j\in[w_i]}$ with each $\tilde{x}_{ij} \in [0, \lambda]$

$$\begin{aligned}
|\phi(R_v) - \phi(R(G, v, L))| &\le \sum_{i=1}^{d}\sum_{j=1}^{w_i} \frac{\Phi(f)}{\Phi(\tilde{x}_{ij})}\left|\frac{\partial f(\tilde{\mathbf{x}})}{\partial x_{ij}}\right| \cdot |\phi(x_{ij}) - \phi(\hat{x}_{ij})| \\
&\overset{(\spadesuit)}{\le} 2\sqrt{\lambda}\sum_{i=1}^{d}\sum_{j=1}^{w_i} \frac{\Phi(f)}{\Phi(\tilde{x}_{ij})}\left|\frac{\partial f(\tilde{\mathbf{x}})}{\partial x_{ij}}\right| \alpha^{L - \lfloor 1 + c\log_{1/\alpha} w_i\rfloor} \\
&\overset{(\heartsuit)}{\le} 2\sqrt{\lambda}\alpha^L.
\end{aligned}$$

($\spadesuit$) follows from the induction hypothesis and ($\heartsuit$) is due to Lemma 8.

The case that $d = \Delta$ can only happen at the root of our computational tree. Following the arguments in the proofs of 8, 9 and the bound in (5), it is easy to see that a universal constant upper bound for the error contraction exists, i.e.,

$$\sum_{i=1}^{\Delta}\sum_{j=1}^{w_i} \frac{\Phi(f)}{\Phi(\tilde{x}_{ij})}\left|\frac{\partial f(\tilde{\mathbf{x}})}{\partial x_{ij}}\right| < \max_{z\in[0,1]} \sqrt{\frac{\Delta^2(\Delta-1)^{\Delta-1}z^\Delta(1-z)}{(\Delta-2)^\Delta + (\Delta-1)^{\Delta-1}z^\Delta}} < 3.$$

Thus $|\phi(R_v) - \phi(R(G, v, L))| \le 6\sqrt{\lambda}\alpha^L$ for every v.

Then the lemma follows from Proposition 7, since

$$\begin{aligned}
|R_v - R(G, v, L)| &= \frac{1}{\Phi(\tilde{x})} \cdot |\phi(R_v) - \phi(R(G, v, L))| \qquad \text{for some } \tilde{x} \in [0, \lambda] \\
&\le 6\lambda\sqrt{1+\lambda} \cdot \alpha^L
\end{aligned}$$

$\blacktriangleleft$

Acknowledgement. Chihao Zhang is supported in part by the National Natural Science Foundation of China Grant 61261130589, 61472239.

References

1. Antar Bandyopadhyay and David Gamarnik. Counting without sampling: Asymptotics of the log-partition function for certain statistical physics models. *Random Structures & Algorithms*, 33(4):452–479, 2008.
2. Magnus Bordewich, Martin Dyer, and Marek Karpinski. Path coupling using stopping times and counting independent sets and colorings in hypergraphs. *Random Structures & Algorithms*, 32(3):375–399, 2008. doi:10.1002/rsa.20204.
3. Martin Dyer, Mark Jerrum, and Eric Vigoda. Rapidly mixing Markov chains for dismantleable constraint graphs. In *Randomization and Approximation Techniques in Computer Science*, pages 68–77. Springer, 2002.
4. Martin E. Dyer, Alan M. Frieze, and Mark Jerrum. On counting independent sets in sparse graphs. *SIAM Jounal on Computing*, 31(5):1527–1541, 2002. URL: http://epubs.siam.org/sam-bin/dbq/article/38384.
5. Martin E. Dyer and Catherine S. Greenhill. On Markov chains for independent sets. *Journal of Algorithms*, 35(1):17–49, 2000.
6. A. Galanis, D. Stefankovic, and E. Vigoda. Inapproximability of the partition function for the antiferromagnetic Ising and hard-core models. *Arxiv preprint arXiv:1203.2226*, 2012.
7. David Gamarnik and Dmitriy Katz. Correlation decay and deterministic FPTAS for counting colorings of a graph. *Journal of Discrete Algorithms*, 12:29–47, 2012.
8. Leslie Ann Goldberg and Mark Jerrum. A polynomial-time algorithm for estimating the partition function of the ferromagnetic Ising model on a regular matroid. *SIAM Journal on Computing*, 42(3):1132–1157, 2013.
9. Mark Jerrum. A very simple algorithm for estimating the number of k-colorings of a low-degree graph. *Random Structures & Algorithms*, 7(2):157–166, 1995.
10. Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM Journal on Computing*, 22(5):1087–1116, 1993.
11. Mark Jerrum and Alistair Sinclair. The Markov chain monte carlo method: an approach to approximate counting and integration. *Approximation algorithms for NP-hard problems*, pages 482–520, 1996.
12. Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM*, 51:671–697, July 2004. doi:10.1145/1008731.1008738.
13. Liang Li, Pinyan Lu, and Yitong Yin. Approximate counting via correlation decay in spin systems. In *Proceedings of the 23th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'12)*, pages 922–940. SIAM, 2012.
14. Liang Li, Pinyan Lu, and Yitong Yin. Correlation decay up to uniqueness in spin systems. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'13)*, pages 67–84. SIAM, 2013.
15. Chengyu Lin, Jingcheng Liu, and Pinyan Lu. A simple FPTAS for counting edge covers. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*, pages 341–348, 2014. doi:10.1137/1.9781611973402.25.
16. Jingcheng Liu and Pinyan Lu. FPTAS for #BIS with degree bounds on one side. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC'15)*, 2015.
17. Jingcheng Liu and Pinyan Lu. FPTAS for counting monotone CNF. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*, pages 1531–1548, 2015.
18. Jingcheng Liu, Pinyan Lu, and Chihao Zhang. FPTAS for counting weighted edge covers. In *In Proceedings of the 22nd European Symposium on Algorithms (ESA'14)*, pages 654–665, 2014.

19 Pinyan Lu, Menghui Wang, and Chihao Zhang. FPTAS for weighted Fibonacci gates and its applications. *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP'14)*, pages 787–799, 2014.

20 Pinyan Lu and Yitong Yin. Improved FPTAS for multi-spin systems. In *Proceedings of APPROX-RANDOM*, pages 639–654. Springer, 2013.

21 Michael Luby and Eric Vigoda. Approximately counting up to four. In *Proceedings of the 29th Annual ACM Symposium on Theory of computing (STOC'97)*, pages 682–687. ACM, 1997.

22 Ricardo Restrepo, Jinwoo Shin, Prasad Tetali, Eric Vigoda, and Linji Yang. Improved mixing condition on the grid for counting and sampling independent sets. *Probability Theory and Related Fields*, 156(1-2):75–99, 2013.

23 Alistair Sinclair, Piyush Srivastava, and Marc Thurley. Approximation algorithms for two-state anti-ferromagnetic spin systems on bounded degree graphs. *Journal of Statistical Physics*, 155(4):666–686, 2014.

24 Allan Sly and Nike Sun. The computational hardness of counting in two-spin models on d-regular graphs. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS'12)*, pages 361–369. IEEE, 2012.

25 Eric Vigoda. Improved bounds for sampling colorings. *Journal of Mathematical Physics*, 41(3):1555–1569, 2000.

26 Dror Weitz. Counting independent sets up to the tree threshold. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC'06)*, pages 140–149. ACM, 2006.

27 Yitong Yin and Jinman Zhao. Counting hypergraph matchings up to uniqueness threshold. *arXiv preprint arXiv:1503.05812*, 2015.

Efficient Enumeration of Solutions Produced by Closure Operations

Arnaud Mary[1] **and Yann Strozecki**[2]

1 **Université Lyon 1; CNRS, UMR5558, LBBE / INRIA – ERABLE, France**
2 **Université de Versailles Saint-Quentin-en-Yvelines, DAVID laboratory, France**

Abstract

In this paper we address the problem of generating all elements obtained by the saturation of an initial set by some operations. More precisely, we prove that we can generate the closure by polymorphisms of a boolean relation with a polynomial delay. Therefore we can compute with polynomial delay the closure of a family of sets by any set of "set operations" (e.g. by union, intersection, difference, symmetric difference...). To do so, we prove that for any set of operations $\mathcal{F}$, one can decide in polynomial time whether an element belongs to the closure by $\mathcal{F}$ of a family of sets. When the relation is over a domain larger than two elements, we prove that our generic enumeration method fails, since the associated decision problem is NP-hard.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases enumeration, set saturation, polynomial delay, Post's lattice

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.52

1 Introduction

In enumeration we are interested in listing a set of elements, which can be of exponential cardinality in the size of the input. The complexity of enumeration problems is thus measured in terms of the input size and output size. The enumeration algorithms with a complexity polynomial in both the input and output are called output polynomial or total polynomial time. Another, more precise notion of complexity, is the *delay* which measures the time between the production of two consecutive solutions. We are especially interested in problems solvable with a delay polynomial in the input size, which are considered as the *tractable problems* in enumeration complexity. For instance, the maximal independent sets of a graph can be enumerated with polynomial delay [7].

If we allow the delay to grow during the algorithm, we obtain incremental delay algorithms: the first k solutions can be enumerated in a time polynomial in k and in the size of the input. Many problems which can be solved with an incremental delay have the following form: given a set of elements and a polynomial time function acting on tuples of elements, produce the closure of the set by the function. For instance, the best algorithm to generate all circuits of a matroid is in incremental delay because it uses some closure property of the circuits [8].

In this article, we try to understand when saturation problems which are natural incremental delay problems can be in fact solved by a polynomial delay algorithm. To tackle this question we need to restrict the saturation operation. In this article, an element will be a vector over some finite set and we ask the saturation operation to act *coefficient-wise* and in the same way on each coefficient. We prove that, when the vector is over the boolean domain, every possible saturation can be computed in polynomial delay. To do that we study a decision version of our problem, denoted by $\textsc{Closure}_{\mathcal{F}}$: given a vector v and a set


© Arnaud Mary and Yann Strozecki;
licensed under Creative Commons License CC-BY
33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 52; pp. 52:1–52:13

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


of vectors $\mathcal{S}$ decide whether v belongs to the closure of $\mathcal{S}$ by the operations of $\mathcal{F}$. We prove $\textsc{Closure}_{\mathcal{F}} \in \mathsf{P}$ for all sets of operations $\mathcal{F}$ over the boolean domain.

When the domain is boolean, the problem can be reformulated in term of set systems or hypergraphs. It is equivalent to generating the smallest hypergraph which contains a given hypergraph and which is closed by some operations. We show how to efficiently compute the closure of a hypergraph by any family of set operations (any operation that is the composition of unions, intersections and complementations) on the hyperedges. This extends known methods such as the closure of a hypergraph by union, by union and intersection or the generation of the cycles of a graph by computing the closure of the fundamental cycles by symmetric difference. In general, knowing how to compute a closure may serve as a good tool to design other enumeration algorithms. One only has to express an enumeration problem as the closure of some sufficiently small and easy to compute set of elements and then to apply the algorithms presented in this article.

The closure computation is also related to constraint satisfaction problems (CSP). Indeed, the set of vectors can be seen as a relation R and the problem of generating its closure by some operation f is equivalent to the computation of the smallest relation R' containing R such that f is a polymorphism of R'. There are several works on enumeration in the context of CSP, which deal with enumerating solutions of a CSP in polynomial delay [5, 3]. The simplest such result [5] states that in the boolean case, there is a polynomial delay algorithm if and only if the constraint language is Horn, anti-Horn, bijunctive or affine. Our work is completely unrelated to these results, since we are not interested in the solutions of CSPs but only in generating the closure of relations. However, we use tools from CSPs such as the Post's lattice [10], used by Schaefer in his seminal paper [13], and the Baker-Pixley theorem [2].

The main theorem of this article settles the complexity of a whole family of decision problems and implies, quite surprisingly, that the backtrack search is enough to obtain a polynomial delay algorithm to enumerate the closure of boolean vectors under any fixed set of binary operations. For all these enumeration problems, compared to the naive saturation algorithm, our method has a better time complexity (even from a practical point of view) and a better space complexity (polynomial rather than exponential). Moreover, besides the generic enumeration algorithm, we try to give for each closure rule an algorithm with the best possible complexity. In doing so, we illustrate several classical methods used to enumerate objects such as amortized backtrack search, hill climbing, Gray code ...

1.1 Organization of the Paper

In Sec. 2, we define enumeration complexity, our problem and the backtrack search. In Sec. 3, we use Post's lattice, restricted through suitable reductions between clones, to determine the complexity of $\textsc{Closure}_{\mathcal{F}}$ for all sets of binary operations $\mathcal{F}$. It turns out that there are only a few types of closure operations: the monotone operations (Sec. 3.1), the addition over $\mathbb{F}_2$ (Sec. 3.2), the set of all operations (Sec. 3.3), two infinite hierarchies related to the majority function (Sec. 3.4) and the limit cases of the previous hierarchies (Sec. 3.5). Finally, in Sec. 4, we give polynomial delay algorithms for three classes of closure operation over any finite domain and prove that the method we use in the boolean case fails.

2 Preliminaries

Given $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, ..., n\}$. For a set D and a vector $v \in D^n$, we denote by v_i the i^{th} coordinate of v. Let $i, j \in [n]$, we denote by $v_{i,j}$ the vector (v_i, v_j). More generally,

for a subset $I = \{i_1, ..., i_k\}$ of $[n]$ with $i_1 < ... < i_k$ we denote by v_I the vector $(v_{i_1}, ..., v_{i_k})$. If $\mathcal{S}$ is a set of vectors we denote by $\mathcal{S}_I$ the set $\{v_I \mid v \in \mathcal{S}\}$. The characteristic vector v of a subset E of $[n]$ is the vector in $\{0,1\}^n$ such that $v_i = 1$ if and only if $i \in X$.

2.1 Complexity

In this section, we recall basic definitions about enumeration problems and their complexity, for further details and examples see [14].

Let Σ be some finite alphabet. An *enumeration problem* is a function A from Σ^* to $\mathcal{P}(\Sigma^*)$. That is to each input word, A associates a set of words. An algorithm which solves the enumeration problem A takes any input word w and produces the set $A(w)$ word by word and *without redundancies*. We always require the sets $A(w)$ to be finite. We may also ask $A(w)$ to contain only words of polynomial size in the size of w and that one can test whether an element belongs to $A(w)$ in polynomial time. If those two conditions hold, the problem is in the class EnumP which is the counterpart of NP for enumeration. Because of this relationship to NP, we often call solutions the elements we enumerate.

The *delay* is the time between the productions of two consecutive solutions. It also includes the time to find the first solution and the time to detect that no further solution exists. Usually we want to bound the delay of an algorithm for all pairs of consecutive solutions and for all inputs of the same size. If this delay is polynomial in the size of the input, then we say that the algorithm is in *polynomial delay* and the problem is in the class DelayP. We also require that the time to find the first solution and the time to detect that no further solution exists is polynomial. If the delay is polynomial in the input and the number of already generated solutions, we say that the algorithm is in *incremental delay* and the problem is in the class IncP. By definition we have DelayP $\subset$ IncP. Moreover (DelayP $\cap$ EnumP) $\neq$ (IncP $\cap$ EnumP) modulo the exponential time hypothesis [4]. Note that in an enumeration algorithm we allow a polynomial precomputation step, usually to set up data structures, which is not taken into account in the delay. This is why we can have a delay smaller than the size of the input.

We now explain a very classical and natural enumeration method called the *Backtrack Search* (sometimes also called the *flashlight method*) used in many previous articles [11, 15]. We represent the solutions we want to enumerate as vectors of size n and coefficients in D. In practice solutions are often subsets of $[n]$ which means that $D = \{0,1\}$ and the vector is the characteristic vector of the subset.

The enumeration algorithm is a depth first traversal of a tree whose nodes are partial solutions. The nodes of the tree will be all vectors v of size l, for all $l \leq n$, such that $v = w_{[l]}$ and w is a solution. The children of the node v will be the vectors of size $l+1$, which restricted to $[l]$ are equal to v. The leaves of this tree are the solutions of our problem, therefore a depth first traversal will visit all leaves and yield all solutions. We want an enumeration algorithm with a delay polynomial in n. Since a branch of the tree is of size n, we need to be able to find the children of a node in a time polynomial in n to obtain a polynomial delay. The delay also depends linearly on $|D|$, but in the rest of the paper $|D|$ will be constant. Therefore the problem is reduced to the following *decision* problem: given v of size l is there w a solution such that $v = w_{[l]}$? This problem is called the *extension problem* associated to the enumeration problem.

▶ **Proposition 1.** *Given an enumeration problem A, such that for all w, $A(w)$ can be seen as vectors of size n with coefficients in D, with n and $|D|$ polynomially related to $|w|$. If the extension problem associated to A is in* P*, then A is in* DelayP*.*

2.2 Closure of Families by Set Operations

We fix D a finite domain. Given a t-ary operation f (a function from D^t to D), f can be naturally extended to a t-ary operation over vectors of the same size. For a t-uples of vectors of size n $v^1, \dots v^t$, f will then act coefficient-wise, that is for all $i \leq n$, $f(v^1, \dots, v^t)_i = f(v_i^1, \dots, v_i^t)$.

▶ **Definition 2.** Let $\mathcal{F}$ be a finite set of operations over D. Let $\mathcal{S}$ be a set of vectors of size n over D. Let $\mathcal{F}^i(\mathcal{S}) = \{f(v_1, \dots, v_t) \mid v_1, \dots, v_t \in \mathcal{F}^{i-1}(S) \text{ and } f \in \mathcal{F}\}$ and $\mathcal{F}^0(\mathcal{S}) = \mathcal{S}$. The closure of $\mathcal{S}$ by $\mathcal{F}$ is $Cl_{\mathcal{F}}(\mathcal{S}) = \cup_i \mathcal{F}^i(\mathcal{S})$.

Remark that $Cl_{\mathcal{F}}(\mathcal{S})$ is also the smallest set which contains $\mathcal{S}$ and which is closed by the operations of $\mathcal{F}$. The set $Cl_{\mathcal{F}}(\mathcal{S})$ is invariant under the operations of $\mathcal{F}$: these operations are called *polymorphisms* of the set $Cl_{\mathcal{F}}(\mathcal{S})$, a notion which comes from universal algebra.

As an illustration, assume that $D = \{0, 1\}$ and that $\mathcal{F} = \{\vee\}$. Then the elements of $\mathcal{S}$ can be seen as subsets of $[n]$ (each vector of size n is the characteristic vector of a subset of $[n]$) and $\textsc{Closure}_{\{\vee\}}(\mathcal{S})$ is the closure by union of all sets in S. Let $\mathcal{S} = \{\{1, 2, 4\}, \{2, 3\}, \{1, 3\}\}$ then $Cl_{\{\vee\}}(\mathcal{S}) = \{\{1, 2, 4\}, \{1, 2, 3, 4\}, \{2, 3\}, \{1, 3\}, \{1, 2, 3\}\}$. Remark that $Cl_{\{\vee\}}(\mathcal{S})$ is indeed closed by union, that is $\vee$ is a polymorphism of $Cl_{\{\vee\}}(\mathcal{S})$.

The problem we try to solve in this article, for all sets of operations $\mathcal{F}$ over D, is $\textsc{EnumClosure}_{\mathcal{F}}$: given a set of vectors $\mathcal{S}$ compute $Cl_{\mathcal{F}}(S)$. We will always denote the size of the vectors of $\mathcal{S}$ by n and the cardinality of $\mathcal{S}$ by m. We introduce two related decision problems. First, the extension problem associated to a set of operations $\mathcal{F}$, is the problem $\textsc{ExtClosure}_{\mathcal{F}}$: given $\mathcal{S}$ a set of vectors of size n, and a vector v of size $l \leq n$, is there a vector $v' \in Cl_{\mathcal{F}}(\mathcal{S})$ such that $v = v'_{[l]}$. Second, the closure problem, denoted by $\textsc{Closure}_{\mathcal{F}}$, is a restricted version of the extension problem where v is of size n. Remark that $\textsc{ExtClosure}_{\mathcal{F}}$ can be reduced in linear time to $\textsc{Closure}_{\mathcal{F}}$ by transforming the instance $(\mathcal{S}, v)$ of $\textsc{ExtClosure}_{\mathcal{F}}$ with v of size l into the instance $(\mathcal{S}_{[l]}, v)$ of $\textsc{Closure}_{\mathcal{F}}$. By combining the previous remark and Proposition 1, we have the following proposition.

▶ **Proposition 3.** *If* $\textsc{Closure}_{\mathcal{F}} \in \mathsf{P}$ *then* $\textsc{EnumClosure}_{\mathcal{F}} \in \mathsf{DelayP}$.

We have introduced an infinite family of problems, whose complexity we want to determine. Several families of operations may produce the same closure. To deal with that, we need to introduce the notion of functional clone.

▶ **Definition 4.** Let $\mathcal{F}$ be a finite set of operations over D, the functional clone generated by $\mathcal{F}$, denoted by $< \mathcal{F} >$, is the set of operations obtained by any composition of the operations of $\mathcal{F}$ and of the projections $\pi_k^n : D^n \to D$ defined by $\pi_k^n(x_1, \dots, x_n) = x_k$.

This notion is useful, because two sets of functions which generate the same clone applied to the same set produce the same closure. Therefore to prove our main theorem, we need to consider all clones rather than all sets of functions.

▶ **Lemma 5.** *For all set of operations* $\mathcal{F}$ *and all set of vectors* $\mathcal{S}$, $Cl_{\mathcal{F}}(\mathcal{S}) = Cl_{<\mathcal{F}>}(\mathcal{S})$.

The number of clones over D is infinite even when D is the boolean domain (of size 2). However, in this case the clones form a countable lattice, called Post's lattice [10]. Moreover there is a *finite* number of well described clones plus a few very regular infinite families of clones.

Clone	Base
I_2	$\emptyset$
L_2	$x+y+z$
L_0	$x+y$
E_2	$\wedge$
S_{10}	$x \wedge (y \vee z)$
S_{10}^k	$Th_k^{k+1}, x \wedge (y \vee z)$
S_{12}	$x \wedge (y \to z)$
S_{12}^k	$Th_k^{k+1}, x \wedge (y \to z)$
D_2	maj
D_1	$maj, x+y+z$
M_2	$\vee, \wedge$
R_2	$x\,?\,y : z$
R_0	$\vee, +$

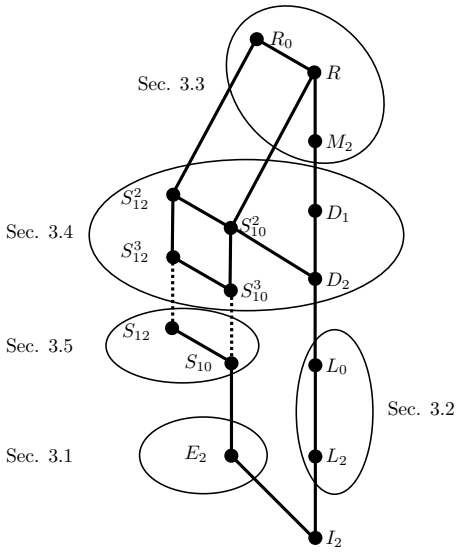


Figure 1 The reduced Post's lattice, the edges represent inclusions of clones.

3 The Boolean Domain

In this part we will prove our main theorem on the complexity of $\textsc{Closure}_{\mathcal{F}}$, when the domain is boolean. An instance of one such problem, denoted by $\mathcal{S}$, will be equivalently seen as a set of vectors of size n or a set of subsets of $[n]$.

▶ **Theorem 6.** *Let $\mathcal{F}$ be any fixed finite set of operations over the boolean domain, then* $\textsc{Closure}_{\mathcal{F}} \in \mathsf{P}$ *and* $\textsc{EnumClosure}_{\mathcal{F}} \in \mathsf{DelayP}$.

To prove our main theorem, we will prove that $\textsc{Closure}_{\mathcal{F}} \in \mathsf{P}$, for each clone $\mathcal{F}$ in Post's lattice. We first show that for certain $\mathcal{F}$ the problem $\textsc{Closure}_{\mathcal{F}}$ can be reduced to $\textsc{Closure}_{\mathcal{G}}$ where $\mathcal{G}$ is a simpler clone obtained from $\mathcal{F}$. This helps to reduce the number of cases we need to consider.

To an operation f we can associate its dual $\overline{f}$ defined by $\overline{f}(s_1, \dots, s_t) = \neg f(\neg s_1, \dots, \neg s_t)$. If $\mathcal{F}$ is a set of operations, $\overline{\mathcal{F}}$ is the set of duals of operation in $\mathcal{F}$. We denote by $\mathbf{0}$ and $\mathbf{1}$ the constant functions which always return 0 and 1. By a slight abuse of notation, we will also denote by $\mathbf{0}$ the all zero vector and by $\mathbf{1}$ the all one vector.

▶ **Proposition 7.** *The following problems can be polynomially reduced to* $\textsc{Closure}_{\mathcal{F}}$*:*

1. $\textsc{Closure}_{\mathcal{F}\cup\{\mathbf{0}\}}$, $\textsc{Closure}_{\mathcal{F}\cup\{\mathbf{1}\}}$, $\textsc{Closure}_{\mathcal{F}\cup\{\mathbf{0},\mathbf{1}\}}$
2. $\textsc{Closure}_{\overline{\mathcal{F}}}$
3. $\textsc{Closure}_{\mathcal{F}\cup\{\neg\}}$ *when* $\mathcal{F} = \overline{\mathcal{F}}$

For a modern presentation of all boolean clones, their bases and the Post's lattice see [12]. In Fig. 1, we represent only the clones such that any clone of Post's lattice can be reduced to one of those using Proposition 7. We have already explained how our main theorem need only to be proved for all clones rather than all sets of functions. By Proposition 7, it is enough to prove it for all clones of Fig. 1, as we now do in the rest of this section.

3.1 Conjunction

We first study one of the simplest clones: $E_2 =< \wedge >$. We give an elementary proof that $\text{CLOSURE}_{E_2} \in \mathsf{P}$, then we explain how to obtain a good delay for ENUMCLOSURE_{E_2}. For a binary vector v, let us denote by $\not\vdash(v)$ (resp. $\not\dashv(v)$) the set of indices i for which $v_i = 0$ (resp. $v_i = 1$).

▶ **Proposition 8.** $\text{CLOSURE}_{E_2} \in \mathsf{P}$.

Proof. Let $\mathcal{S}$ be a set of Boolean vectors. If we apply $\wedge$ to a couple of vectors in $\mathcal{S}$ it produces the intersection of two vectors when seen as sets. Since the intersection operation is associative and commutative, $Cl_{E_2}(\mathcal{S})$ is the set of arbitrary intersections of elements of $\mathcal{S}$. Let v be a vector and let $\mathcal{S}_1$ be the set $\{w \in \mathcal{S} \mid w_{\not\dashv(v)} = \mathbf{1}\}$. Assume now that v can be obtained as an intersection of elements $v_1, \ldots, v_t$, those elements must be in $\mathcal{S}_1$ because of the monotonicity of the intersection for the inclusion. On the other hand, by definition of $\mathcal{S}_1$, v will always be smaller than or equal to $\cap_{w \in \mathcal{S}_1} w$. Therefore, $v \in Cl_{E_2}(\mathcal{S})$ if and only if $v = \cap_{w \in \mathcal{S}_1} w$. This intersection can be computed in time $O(mn)$ which concludes the proof. ◀

By Proposition 1, we can turn the algorithm for CLOSURE_{E_2} into an enumeration algorithm for ENUMCLOSURE_{E_2} with delay $O(mn^2)$. We explain in the next proposition how to reduce this delay to $O(mn)$, which is the best known complexity for this problem.

▶ **Proposition 9.** *There is an algorithm solving* ENUMCLOSURE_{E_2} *with a delay* $O(mn)$.

Proof. We use the backtrack search described in Proposition 1 but we maintain data structures which allow us to decide CLOSURE_{E_2} quickly. Let $\mathcal{S}$ be the input set of m vectors of size n. During the traversal of the tree we update the partial solution p, represented by an array of size n which stores whether $p_i = 1$, $p_i = 0$ or is yet undefined.

A vector v of $\mathcal{S}$ is compatible with the partial solution if $\not\dashv_p \subseteq \not\dashv_v$. We maintain an array $COMP$ indexed by the sets of $\mathcal{S}$, which stores whether each vector of $\mathcal{S}$ is compatible or not with the current partial solution. Finally we update an array $COUNT$, such that $COUNT[i]$ is the number of compatible vectors $v \in \mathcal{S}$ such that $v_i = 0$. Remark that a partial solution p can be extended into a vector of $Cl_{E_2}(\mathcal{S})$ if and only if for all $i \in \not\vdash_p COUNT[i] > 0$, the solution is then the intersection of all compatible vectors.

At each step of the traversal, we select an index i such that p_i is undefined and we set first $p_i = 0$ then $p_i = 1$. When we set $p_i = 0$, there is no change to do in $COUNT$ and $COMP$ and we can check whether this extended partial solution is correct by checking if $COUNT[i] > 0$ in constant time. When we set $p_i = 1$, we need to update $COMP$ by removing from it every vector v such that $v_i = 0$. Each time we remove such a vector v, we decrement $COUNT[j]$ for all j such that $v_j = 0$. If there is a j such that $COUNT[j]$ is decremented to 0 then the extension of p by $p_i = 1$ is not possible.

When we traverse a whole branch of the tree of partial solutions during the backtrack search, we will set $p_i = 1$ for each i at most once and then we need to remove each vector from $COMP$ at most once. Therefore the total number of operations we do to maintain $COMP$ and $COUNT$ is $O(mn)$ and so is the delay. ◀

The problem ENUMCLOSURE_{E_2} is related to several interesting enumeration problems such as listing the solutions of a DNF formula. There is an intriguing open question on its complexity: can we have a delay sublinear in m or only dependent on n, that is a delay polynomial in the size of the solutions? For all other clones, in contrast, we give enumeration algorithms with a delay polynomial in the size of the solutions.

3.2 Algebraic Operations

We first deal with the clone $L_0 =< + >$ where $+$ is the boolean addition. Note that $Cl_{L_0}(\mathcal{S})$ is the vector space generated by the vectors in $\mathcal{S}$. Seen as an operation on sets, $+$ is the symmetric difference of the two sets.

▶ **Proposition 10.** $\textsc{Closure}_{L_0} \in \mathsf{P}$.

Proof. Let $\mathcal{S}$ be the set of input vectors, let v be a vector and let A be the matrix whose rows are the elements of $\mathcal{S}$. The vector v is in $Cl_{L_0}(\mathcal{S})$ if and only if there is a solution over $\mathbb{F}_2$ to $Ax = v$. Solving a linear system over $\mathbb{F}_2$ can be done in polynomial time which proves the proposition. ◀

The previous proposition yields a polynomial delay algorithm by applying Proposition 1. One can get a better delay, by computing in polynomial time a maximal free family M of $\mathcal{S}$, which is a basis of $Cl_{L_0}(\mathcal{S})$. The basis M is a succinct representation of $Cl_{L_0}(\mathcal{S})$. One can generate all elements of $Cl_{L_0}(\mathcal{S})$ by going over all possible subsets of elements of M and summing them. The subsets can be enumerated in constant time by using Gray code enumeration (see [9]). The sum can be done in time n by adding a single vector since two consecutive sets differ by a single element in the Gray code order. Therefore we have, after the polynomial time computation of M, an enumeration in delay $O(n)$.

With some care, we can extend this result to the clone L_2 generated by the sum modulo two of three elements.

▶ **Proposition 11.** $\textsc{Closure}_{L_2} \in \mathsf{P}$.

Proof. First remark that any vector in $Cl_{L_2}(\mathcal{S})$ is the sum of an odd number of vectors in $\mathcal{S}$. In other words $v \in Cl_{L_2}(\mathcal{S})$ if and only if there is a vector x such that $Ax = v$ and that the Hamming weight of x is odd. One can compute a basis B of the vector space of the solutions to the equation $Ax = v$. If all elements of B have Hamming weight even, then their sums also have Hamming weight even. Therefore $v \in Cl_{L_2}(\mathcal{S})$ if and only if there is an element in B with odd Hamming weight, which can be decided in polynomial time. ◀

3.3 Conjunction and Disjunction

In this subsection, we deal with the largest possible clones of our reduced Post lattice: $M_2 =< \wedge, \vee >$, $R_2 =< x\,?\,y : z >$ and $R_0 =< \vee, + >$.

▶ **Proposition 12.** $\textsc{Closure}_{M_2} \in \mathsf{P}$.

Proof. Let $\mathcal{S}$ be a vector set and for all $i \in [n]$, let $X_i := \{v \in \mathcal{S} \mid v_i = 1\}$. We will show that a vector u belongs to $Cl_{M_2}(\mathcal{S})$ if and only if $u = \bigvee_{i \in \not\curlywedge(u)} \bigwedge_{v \in X_i} v$. Clearly, if $u = \bigvee_{i \in \not\curlywedge(u)} \bigwedge_{v \in X_i} v$ then $u \in Cl_{M_2}(\mathcal{S})$.

Assume first that there exists $i \in \not\curlywedge(u)$ such that $X_i = \emptyset$ i.e. for all $v \in \mathcal{S}$, $v_i = 0$. Then clearly, for all $w \in Cl_{M_2}(\mathcal{S})$, $w_i = 0$ and then $u \notin Cl_{M_2}(\mathcal{S})$. Assume now that $X_i \neq \emptyset$ for all $i \in \not\curlywedge(u)$ and assume that $u \neq t := \bigvee_{i \in \not\curlywedge(u)} \bigwedge_{v \in X_i} v$. So there exists $j \in \curlywedge(u)$ such that $t_j = 1$. Thus, there exists $i \in \not\curlywedge(u)$ such that for all $v \in X_i$, $v_j = 1$. We have that for all $v \in \mathcal{S}$, $v_i = 1 \implies v_j = 1$. Let us show that this property is preserved by both operations $\wedge$ and $\vee$ and then that this property holds for all $w \in Cl_{M_2}(\mathcal{S})$. Assume that the property holds for a set $\mathcal{F}$. Let $a, b \in \mathcal{F}$ and let $v := a \wedge b$. If $v_i = 1$, we have $a_i = 1$ and $b_i = 1$ and then $a_j = 1$ and $b_j = 1$. We conclude that $v_j = a_j \wedge b_j = 1$. Assume now that $v = a \vee b$ and

that $v_i = 1$. Then either $a_i = 1$ or $b_i = 1$, say w.l.o.g. that $a_i = 1$. Then $a_j = 1$ and we have $v_j = a_j \vee b_j = 1$. We have shown that the property is preserved by both operations, therefore u cannot belong to $Cl_{M_2}(\mathcal{S})$ since $u_i = 1$ and $u_j = 0$. ◀

We can decide $\textsc{Closure}_{M_2}$ in time $O(mn^2)$ therefore by applying Proposition 1, we get an enumeration algorithm with delay $O(mn^3)$. We can precompute the n vectors $x^i = \bigwedge_{v \in X_i} v$ and generate their unions in delay $O(n^2)$ thanks to Proposition 9. We can do better by using the inclusion structure of the x^i to obtain a $O(n)$ delay.

▶ **Proposition 13.** $\textsc{EnumClosure}_{M_2}$ *can be solved with delay* $O(n)$.

If we consider $\textsc{EnumClosure}_{M_2 \cup \{\neg\}}(\mathcal{S})$, it is very easy to enumerate. Let $X^i = \{v \mid v \in \mathcal{S}, v_i = 1\} \cup \{\neg v \mid v \in \mathcal{S}, v_i = 0\}$ and let $x^i = \bigwedge_{v \in X_i} v$. The set $Cl_{M_2 \cup \{\neg\}}(\mathcal{S})$ is in fact a boolean algebra, whose atoms are the x^i. Indeed, either $x^i_{i,j} = x^j_{i,j}$ and they are equal or $\not\models_{x^i} \cap \not\models_{x^j} = \emptyset$. If $A = \{x^i \mid i \in [n]\}$, two distinct unions of elements in A produce distinct elements. Hence by enumerating all possible subsets of A with a Gray code, we can generate $Cl_{M_2 \cup \{\neg\}}(\mathcal{S})$ with a delay $O(n)$ (even $O(1)$ when always equal coefficients are grouped together).

The closures by the clones R_2 and R_0 are equal to the closure by $M_2 \cup \{\neg\}$ up to some coefficients which are fixed to 0 or 1, thus they are as easy to enumerate.

▶ **Proposition 14.** *The problems* $\textsc{Closure}_{R_2}$, $\textsc{Closure}_{R_0}$ *can be reduced to* $\textsc{Closure}_{M_2}$ *in polynomial time.*

3.4 Majority and Threshold

An operation f is a *near unanimity* of arity k if it satisfies $f(x_1, x_2, \ldots, x_k) = x$ for each k-tuple with at most one element different from x. The *threshold* function of arity k, denoted by Th^k_{k-1}, is defined by $Th^k_{k-1}(x_1, \ldots, x_k) = 1$ if and only if at least $k-1$ of the elements $x_1, \ldots, x_k$ are equal to one. It is the smallest near unanimity operation over the booleans. The threshold function Th^3_2 is the majority operation over three booleans that we denote by maj and the clone it generates is D_2. We first give a characterization of $Cl_{D_2}(\mathcal{S})$ which helps prove that $\textsc{Closure}_{D_2} \in P$. The characterization is a particular case of a universal algebra theorem that we then use to compute the closure by any clone which contains a threshold function.

▶ **Lemma 15.** *Let* $\mathcal{S}$ *be a vector set. A vector* v *belongs to* $Cl_{D_2}(\mathcal{S})$ *if and only if for all* $i, j \in [n]$, $i \neq j$, *there exists* $x \in \mathcal{S}$ *such that* $x_{i,j} = v_{i,j}$.

Proof. ($\Longrightarrow$) Given $a, b \in \{0,1\}$ and $i, j \in [n]$, $i \neq j$, we first show that if for all $v \in \mathcal{S}$, $v_i \neq a$ or $v_j \neq b$ then for all $u \in Cl_{D_2}(\mathcal{S})$, $v_i \neq a$ or $v_j \neq b$. It is sufficient to prove that this property is preserved by applying maj to a vector set i.e. that if $\mathcal{S}$ has this property, then $maj(\mathcal{S})$ has also this property. Let $x, y, z \in \mathcal{S}$, $v := maj(x, y, z)$, and assume for contradiction that $v_{i,j} = (a, b)$. Since $v_i = a$, there are at least two vectors among $\{x, y, z\}$ that are equal to a at index i. Without loss of generality, let x and y be these two vectors. Since for all $u \in \mathcal{S}$, $u_i \neq a$ or $u_j \neq b$, we have $x_j \neq b$ and $y_j \neq b$ and then $v_j \neq b$ which contradicts the assumption. We conclude that if $v \in Cl_{D_2}(\mathcal{S})$, then for all $i, j \in [n]$, there exists $u \in \mathcal{S}$ with $v_{i,j} = u_{i,j}$.
($\Longleftarrow$) Let $k \leq n$ and let $a_1, ..., a_k \in \{0,1\}$. We will show by induction on k, that if for all $i, j \leq k$ there exists $v \in \mathcal{S}$ with $v_i = a_i$ and $v_j = a_j$, then there exists $u \in Cl_{D_2}(\mathcal{S})$ with $u_1 = a_1$, $u_2 = a_2$, ..., $u_k = a_k$. The assertion is true for $k = 2$. Assume it is true for $k-1$,

and let $a_1, ..., a_k \in \{0,1\}$. By induction hypothesis there exists a vector $w \in Cl_{D_2}(\mathcal{S})$ with $w_1 = a_1$, ..., $w_{k-1} = a_{k-1}$. By hypothesis, for all $i \leq k$ there exists $v^i \in \mathcal{S}$ with $v^i_i = a_i$ and $v^i_k = a_k$. We then construct a sequence of vectors $(u^i)_{i \leq k}$ as follow. We let $u^1 = v^1$ and for all $1 < i < k$, $u^i = maj(w, u^{i-1}, v^i)$. We claim that $u := u^{k-1}$ has the property sought i.e. for all $i \leq k$, $u_i = a_i$. First let prove that for all $i < k$ and for all $j \leq i$, $u^i_j = a_j$. It is true for u_1 by definition. Assume now that the property holds for u^{i-1}, $i < k$. Then, by construction, for all $j \leq i-1$, we have $u^i_j = a_j$ since $w_j = a_j$ and $u^{i-1}_j = a_j$. Furthermore, we have $u^i_i = maj(w_i, u^{i-1}_i, v^i_i) = a_i$ since $w_i = a_i$ and $v_i = a_i$. We conclude that for all $i \leq k-1$, $u_i = u^{k-1}_i = a_i$.

We claim now that for all $i < k$, $u^i_k = a_k$. It is true for u^1. Assume it is true for u^{i-1}, $i < k$. Then we have $u^i_k = maj(w_k, u^{i-1}_k, v^i_k)$ which is equal to a_k since $u^{i-1}_k = a_k$ by induction and $v^i_k = a_k$ by definition. We then have $u_i = a_i$ for all $i \leq k$ which concludes the proof. ◀

As an immediate consequence we get the following corollary and proposition.

▶ **Corollary 16.** $\textsc{Closure}_{D_2} \in \mathsf{P}$.

Proof. Using Lemma 15, one decides whether a vector v is in $Cl_{D_2}(S)$, by considering every pair of indices i, j and checking whether there is a vector $w \in S$ such that $v_{i,j} = w_{i,j}$. The complexity is in $O(mn^2)$. ◀

▶ **Proposition 17.** $\textsc{EnumClosure}_{D_2}$ *can be solved in delay* $O(n^2)$.

Proof. We do a backtrack search and we explain how to efficiently decide $\textsc{Closure}_{D_2}$ during the enumeration. We first precompute for each pair (i, j) all values (a, b) such that there exists $v \in \mathcal{S}$, $v_{i,j} = (a, b)$. When we want to decide whether the vector v of size l can be extended into a solution, it is enough that it satisfies the condition of Lemma 15. Moreover, we already know that $v_{[l-1]}$ satisfies the condition of Lemma 15. Hence we only have to check that the values of $v_{i,l}$ for all $i < l$ can be found in $\mathcal{S}_{i,l}$ which can be done in time $O(l)$. The delay is the sum of the complexity of deciding $\textsc{Closure}_{D_2}$ for each partial solution in a branch: $O(n^2)$. ◀

It turns out that Lemma 15 is a particular case of a general theorem of universal algebra which applies to all near unanimity terms. However we felt it was interesting to give the lemma and its proof to get a sense of how the following theorem is proved.

▶ **Theorem 18** (Baker-Pixley, adapted from [2])**.** *Let* $\mathcal{F}$ *be a clone which contains a near unanimity term of arity* k*, then* $v \in Cl_{\mathcal{F}}(\mathcal{S})$ *if and only if for all sets of indices* I *of size* $k-1$*,* $v_I \in Cl_{\mathcal{F}}(\mathcal{S})_I$.

This allows to settle the case of $D_1 =< maj, x+y+z >$ and of the two infinite families of clones of our restricted lattice $S^k_{10} =< Th^{k+1}_k, x \wedge (y \vee z) >$ and $S^k_{12} =< Th^{k+1}_k, x \wedge (y \to z) >$.

▶ **Corollary 19.** *If a clone* $\mathcal{F}$ *contains* Th^{k+1}_k *then* $\textsc{Closure}_{\mathcal{F}}$ *is solvable in* $O(mn^k)$*. In particular* $\textsc{Closure}(S^k_{10})$*,* $\textsc{Closure}(S^k_{12})$ *and* $\textsc{Closure}(D_1)$ *are in* P.

We have proved that the complexity of any closure problem in one of our infinite families is polynomial. Remark that we can use the method of Proposition 17 to obtain a delay $O(n^k)$ for enumerating the elements of a set closed by a near unanimity function of arity k. Notice that we could have applied Theorem 18 to the clones of Subsection 3.3 which all contain the maj function. However, it was relevant to deal with them separately to obtain a different algorithm with delay $O(n)$ rather than $O(n^2)$.

Notice that the complexity of $\textsc{Closure}_{\mathcal{F}}$ is increasing with the smallest arity of a near unanimity function in $\mathcal{F}$. We should thus investigate the complexity of the uniform problem when the clone is given as input. Let ClosureTreshold be the following problem: given a set $\mathcal{S}$ of vectors and an integer k decide whether the vector $\mathbf{1} \in Cl_{S_{10}^k}(\mathcal{S})$. It is a restricted version of the uniform problem, but it is already hard to solve because we can reduce the Hitting Set problem to its complement.

▶ **Theorem 20.** ***ClosureTreshold*** *is* coNP*-complete.*

In fact, the result is even stronger because our reduction preserves the value k. We cannot hope to get an FPT algorithm for ClosureTreshold parameterized by k since the Hitting Set problem parameterized by the size of the hitting set is W[2]-complete [6]. It means that if we want to significantly improve the delay of our enumeration algorithm for the clone S_{10}^k, we should drop the backtrack search since it relies on solving $\textsc{Closure}_{S_{10}^k}$.

3.5 Limits of the Infinite Parts

Here we deal with the two cases left which are the limits of the two infinite hierarchies of clones we have seen in the previous subsection. Let us begin with $S_{12} =< x \wedge (y \to z) >$.

▶ Remark. Let $\mathcal{S}$ be a vector set and assume that there exists some $i \in [n]$ such that for all $v \in \mathcal{S}$, $v_i = 1$ (resp. $v_i = 0$) then for all $w \in Cl_{S_{12}}(\mathcal{S})$ we have $w_i = 1$ (resp. $w_i = 0$). Then we will assume in this section that for all $i \in [n]$ there is at least a vector v in $\mathcal{S}$ with $v_i = 1$ and a vector w with $w_i = 0$.

▶ **Theorem 21.** *Let $\mathcal{S}$ be a vector set, a vector v belongs to $Cl_{S_{12}}(\mathcal{S})$ if and only if*

- *there exists $w \in \mathcal{S}$ such that $\nVdash(v) \subseteq \nVdash(w)$*
- *for all $(k,i) \in \nVdash(v) \times \nvdash(v)$ there exists $w \in \mathcal{S}$ with $w_{k,i} = (0,1)$ or $w_{k,i} = (1,0)$*

Proof. Let us start by proving the following claim.
Claim: Let $k, i \in [n]$. Then there exists $u \in Cl_{S_{12}}(\mathcal{S})$ such that $u_{k,i} = (1,0)$ if and only if there exists $v \in \mathcal{S}$ such that $v_{k,i} = (1,0)$ or $v_{k,i} = (0,1)$.

Assume first that there exists $v \in \mathcal{S}$ such that $v_{k,i} = (0,1)$. Let $x \in \mathcal{S}$ such that $x_k = 1$ and $y \in \mathcal{S}$ such that $y_i = 0$. Without loss of generality, such vectors exist by the assumption of Remark 3.5. Then $u := x \wedge (v \to y)$ has the sought property, i.e $u_{k,i} = (1,0)$. Assume now that for all $v \in \mathcal{S}$, $v_{k,i} \neq (1,0)$ and $v_{k,i} \neq (0,1)$. We show that this property is preserved by the application of $x \wedge (y \to z)$. For all $v \in \mathcal{S}$, $v_{k,i} = (1,1)$ or $v_{k,i} = (0,0)$. Since the function $x \wedge (y \to z)$ acts coordinate-wise on the vectors, if we consider $w = x \wedge (y \to z)$ with $x, y, z \in \mathcal{S}$ we must have $w_i = w_k$. Therefore $w_{k,i} \neq (1,0)$ and $w_{k,i} \neq (0,1)$ which implies by induction that there is no v with $v_{k,i} = (0,1)$ and $v \in Cl_{S_{12}}(\mathcal{S})$. This completes the proof of the claim and we now prove the theorem.
($\Longleftarrow$) We can simulate $w \wedge v$ with $w \wedge (w \to v)$. We will show that for all $i \in \nvdash(v)$ either there exists a vector $v^i \in \mathcal{S}$ such that $\nVdash(v) \subseteq \nVdash(v^i)$ and $v^i_i = 0$ or we can construct it. Notice that it is sufficient in order to prove that $v \in Cl_{S_{12}}(\mathcal{S})$ since we have $v = \bigwedge_{i \in \nvdash(v)} v^i$. So let $i \in \nvdash(v)$ and assume that for all $w \in \mathcal{S}$ such that $\nVdash(v) \subseteq \nVdash(w)$ we have $w_i = 1$. Let w be such a vector and let $\nVdash(v) = \{j_1, j_2, ..., j_k\}$. We will construct a sequence of vectors $(w^l)_{l \leq k}$ such that for all $l \leq k$ and for all $r \leq l$, $w^l_{j_r} = 1$ and $w^l_i = 0$. Let w^1 be the vector with $w^1_{j_1} = 1$ and $w^1_i = 0$. By the claim, such a vector exists in $Cl_{S_{12}}(\mathcal{S})$. Now for all $l \leq k$, let us define $w^l := w \wedge (u^l \to w^{l-1})$ where u^l is a vector such that $u^l_{j_l} = 0$ and $u^l_i = 1$ and there is such a vector in $Cl_{S_{12}}(\mathcal{S})$ by the claim. Since by induction we have $w^{l-1}_i = 0$, and since $u^l_i = 1$, we have $(u^l \to w^{l-1})_i = 0$ and thus $w^l_i = 0$. Now since $u^l_{j_l} = 0$ and $w_{j_l} = 1$ we have

$w^l_{j_l} = 1$. Finally, for all $r < l$, we have w_{j_r} and $w^{l-1}_{j_r} = 1$. Hence $w^l_{j_r} = 1$. We obtain that $\not\Vdash(v) \subseteq \not\Vdash(w^k)$ and $w^k_i = 0$.
($\Longrightarrow$) Let $v \in Cl_{S_{12}}(\mathcal{S})$. Notice that if $v = x \wedge (y \to z)$, then $\not\Vdash(v) \subseteq \not\Vdash(x)$. Thus, there exists $w \in \mathcal{S}$ such that $\not\Vdash(v) \subseteq \not\Vdash(w)$. Now, by the claim, for all $k, i \in [n]$ such that $v_{k,i} = (1,0)$ there exists $w \in \mathcal{S}$ such that $w_{k,i} = (1,0)$ or $w_{k,i} = (0,1)$ which conclude the proof. ◀

▶ **Corollary 22.** $\textsc{Closure}_{S_{12}} \in \mathsf{P}$.

Finally, we deal with the clone $S_{10} =< x \wedge (y \vee z) >$. The characterization of $Cl_{S_{10}}(\mathcal{S})$ and its proof are very similar to the one of $Cl_{S_{12}}(\mathcal{S})$.

▶ **Theorem 23.** *Let $\mathcal{S}$ be a vector set, a vector v belongs to $Cl_{S_{10}}(\mathcal{S})$ if and only if*

- *there exists $w \in \mathcal{S}$ such that $\not\Vdash(v) \subseteq \not\Vdash(w)$*
- *for all $(k,i) \in \not\Vdash(v) \times \not\Vdash(v)$ there exists $w \in \mathcal{S}$ with $w_{k,i} = (1,0)$*

▶ **Corollary 24.** $\textsc{Closure}_{S_{10}} \in \mathsf{P}$.

4 Larger Domains

In this section, we try to extend some results of the boolean domain to larger domains.

4.1 Tractable Closure

The first tractable case is an extension of the clones of Subsection 3.4. Indeed using Th. 18, we can get an equivalent to Corollary 19 and to Proposition 17 in any domain size.

▶ **Corollary 25.** *If $\mathcal{F}$ contains a near unanimity operation, then* $\textsc{Closure}_{\mathcal{F}} \in P$.

▶ **Proposition 26.** *If $\mathcal{F}$ contains a near unanimity term of arity k, then* $\textsc{EnumClosure}_{\mathcal{F}}$ *can be solved in delay $O(n^{k-1})$.*

The second tractable case is a generalization of Subsection 3.2.

▶ **Proposition 27.** *Let f be a commutative group operation over D, then* $\textsc{Closure}_{<f>} \in P$.

Proof. We want to solve $\textsc{Closure}_{<f>}$, given $\mathcal{S}$ a set of vectors and v a vector. Let A be the matrix which have the elements of $\mathcal{S}$ as rows. The vector v is in $\textsc{Closure}_{<f>}(\mathcal{S})$ if and only there is a vector x with coefficients in $\mathbb{Z}$ such that $Ax = v$. This equation is not over a field so we cannot solve it directly. We apply a classical group theorem to the finite commutative group (D, f), which states that D is a direct sum of cyclic groups $D_1, \dots, D_t$ whose order is the power of a prime. The equation $Ax = v$ can be seen as a set of equations over fields: $A_i x_i = v_i$, for $i \leq t$, where A_i, x_i and v_i are the projection of A, x and v over D_i. We can easily reconstruct an x which have the projections x_i on D_i by the Chinese remainder theorem. Therefore, deciding whether $v \in \textsc{Closure}_{<f>}(\mathcal{S})$ is equivalent to solving a set of linear systems and hence is in polynomial time. ◀

One natural generalization would be to allow the function f to be non commutative. In that case, we conjecture that $\textsc{Closure}_{<f>}$ is NP-hard.

4.2 A Limit to the Backtrack Search

The last case we would like to extend is the clone generated by the conjunction. A natural generalization is to fix an order on D and to study the complexity of $\textsc{Closure}_{<f>}$ with f monotone. Let f be the function over $D = \{0, 1, 2\}$ defined by $f(x, y) = \min(x + y, 2)$. This function is clearly monotone for the usual order. However we can prove that EXACT-3-COVER reduces to $\textsc{Closure}_{<f>}$.

▶ **Proposition 28.** $\textsc{Closure}_{<f>}$ *is* NP*-complete.*

This hardness result implies that we cannot use the backtrack search to solve the associated enumeration algorithm. However, if we allow a space proportional to the number of solutions, we can still get a polynomial delay algorithm for associative functions, a property satisfied by the function f of the last proposition. Remark that the space used can be exponential while the backtrack search only requires a polynomial space.

▶ **Proposition 29.** *If f is an associative function, then* $\textsc{EnumClosure}_{<f>} \in$ DelayP*.*

Proof. Let $\mathcal{S}$ be an instance of $\textsc{EnumClosure}_{<f>}$. Let G be the directed graph with vertices $Cl_{<f>}(\mathcal{S})$ and from each $v \in Cl_{<f>}(\mathcal{S})$, there is an arc to $f(v, s)$ for all $s \in \mathcal{S}$. Since f is associative, by definition of G, every vertex of $Cl_{<f>}(\mathcal{S})$ is accessible from a vertex in $\mathcal{S}$. Therefore we can do a depth-first traversal of the graph G to enumerate all solutions. A step of the traversal is in polynomial time: from an element v we generate its neighborhood: $f(v, s)$ for $s \in \mathcal{S}$. The computation of $f(v, s)$ is in time $O(n)$ and $|\mathcal{S}| = m$. We must also test whether the solution $f(v, s)$ has already been generated. This can be done in time $O(n)$ by maintaining a self balanced search tree containing the generated solutions, since there are at most $|D|^n$ solutions. In conclusion the delay of the enumeration algorithm is in $O(mn)$ thus polynomial. ◀

To obtain a polynomial space algorithm, we could try to use the *reverse search* method [1]. To do that, we want the graph G to be a directed acyclic graph, which is the case if we require the function to be monotone. The monotonicity also ensures that the depth of G is at most $n(|D| - 1)$. However we also need to be able to compute for each element of G a canonical ancestor in polynomial time and it does not seem to be easy even when f is monotone. We leave the question of finding a good property of f which ensures the existence of an easy to compute ancestor open for future research.

Acknowledgements. Authors have been partly supported by the ANR project Aggreg and we thank the members of the project and Mamadou Kanté for interesting discussions about enumeration. We also thank Florent Madelaine for his help with CSP and universal algebra.

References

1 David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1):21–46, 1996.

2 Kirby A Baker and Alden F Pixley. Polynomial interpolation and the chinese remainder theorem for algebraic systems. *Mathematische Zeitschrift*, 143(2):165–174, 1975.

3 Andrei A Bulatov, Víctor Dalmau, Martin Grohe, and Dániel Marx. Enumerating homomorphisms. *Journal of Computer and System Sciences*, 78(2):638–650, 2012.

4 Florent Capelli, Arnaud Durand, and Yann Strozecki. A note on polynomial delay and incremental delay, 2015. URL: `http://www.prism.uvsq.fr/~ystr`.

5 Nadia Creignou and Jean-Jacques Hébrard. On generating all solutions of generalized satisfiability problems. *Informatique théorique et applications*, 31(6):499–511, 1997.

6 Jörg Flum and Martin Grohe. Parameterized complexity theory, 2006.

7 David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

8 Leonid Khachiyan, Endre Boros, Khaled Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. On the complexity of some enumeration problems for matroids. *SIAM Journal on Discrete Mathematics*, 19(4):966–984, 2005.

9 Donald E Knuth. Combinatorial Algorithms, Part 1, volume 4A of The Art of Computer Programming, 2011.

10 Emil Leon Post. *The two-valued iterative systems of mathematical logic*. Princeton University Press, 1941.

11 Robert C Read and Robert E Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.

12 Steffen Reith and Heribert Vollmer. Optimal satisfiability for propositional calculi and constraint satisfaction problems. *Information and Computation*, 186(1):1–19, 2003.

13 Thomas J Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226. ACM, 1978.

14 Yann Strozecki. *Enumeration complexity and matroid decomposition*. PhD thesis, Université Paris Diderot – Paris 7, 2010.

15 Yann Strozecki. On enumerating monomials and other combinatorial structures by polynomial interpolation. *Theory Comput. Syst.*, 53(4):532–568, 2013.

Copyless Cost-Register Automata: Structure, Expressiveness, and Closure Properties

Filip Mazowiecki[1] **and Cristian Riveros**[2]

1 **University of Warsaw, Poland**
2 **Pontificia Universidad Católica de Chile, Chile**

Abstract

Cost register automata (CRA) and its subclass, copyless CRA, were recently proposed by Alur et al. as a new model for computing functions over strings. We study structural properties, expressiveness, and closure properties of copyless CRA. We show that copyless CRA are strictly less expressive than weighted automata and are not closed under reverse operation. To find a better class we impose restrictions on copyless CRA, which ends successfully with a new robust computational model that is closed under reverse and other extensions.

1998 ACM Subject Classification F.4.1. Computational logic

Keywords and phrases Cost Register Automata, Weighted Automata, Semirings

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.53

1 Introduction

Weighted automata (WA) are an expressible extension of finite state automata for computing functions over strings. They have been extensively studied since Schützenberger [17], and its decidability problems [12, 1], extensions [7], logic characterization [7, 11], and applications [14, 6] have been deeply investigated.

Unfortunately, there exists a lack of research on understanding subclasses of functions below the class of WA. Recently Alur et al. [2, 4] introduced the computational model of cost register automata (CRA), an alternative model for computing functions over strings. The idea of this model is to enhance deterministic finite automata with registers that can be combined by using operations over a fixed semiring. In contrast to previous models with counters, CRA blindly updates its registers on each transition by using values computed on the previous state. In [2], it was shown that CRA are strictly more expressive than WA. Interestingly, it was also shown that a natural subfragment of CRA is equally expressive to WA, which gives a new representation for understanding this class of functions.

A new representation for WA allows to study natural subclasses of functions that could not be proposed from the classical perspective. This is the case for the class of copyless CRA that where proposed in [2]. The idea of the so-called copyless restriction is to use each register at most once in every transition. Intuitively, this automaton model is register-deterministic in the sense that it cannot copy the content of each register, similar to a deterministic finite automaton that cannot make a copy of its current state. The copyless restriction was successfully used in the context of streaming tree transducers [3] for capturing MSO-transductions over trees and it was proposed as a natural restriction over CRA. Furthermore, copyless CRA are an excellent candidate for having good decidability properties. It was stated in [2] that the existing proofs of undecidability in WA rely on the unrestricted non-deterministic nature of the model and, thus, it is believed that copyless CRA might have good decidability properties. Despite that this is a natural and interesting model for computing


© Filip Mazowiecki and Cristian Riveros;
licensed under Creative Commons License CC-BY

functions, research on this line has not been pursued further and not much is known about copyless CRA.

In this paper we study the structure, expressiveness, and closure properties of copyless CRA. We start by developing a toolkit of structural properties for analysing copyless CRA. Towards this goal, we introduce a *normal form* on the registers of copyless CRA. We show that every copyless CRA can be put in this normal form which considerably simplifies the analysis of this model. With this restriction we provide further results that explain the flow and grow of registers content during a run. Specifically, we prove that from its normal form one can identify a subset of registers, called *stable* registers, that cannot be reset and are constantly growing during a run. We show that stable registers lead the behaviour of copyless CRA and that they are crucial to analyse the growing rate of loops.

Then we turn our attention on studying the expressivity of copyless CRA. As a proof of concept, we use the structural properties developed in this paper to compare the expressiveness of copyless CRA with the class of functions defined by WA. We show that copyless CRA are strictly less expressive than WA. Furthermore, we show that copyless CRA are strictly less expressive than regular cost functions, a class of functions introduced in [2]. It is important to stress that it was previously believed that copyless CRA was strictly less expressive than WA, but this is the first paper which proves this statement formally.

In the last sections, we focus on the robustness of copyless CRA in terms of its closure properties. The robustness of a computational model is usually measured in terms of how stable is the model when new operations or extensions are allowed. Deterministic finite automata are a good example for the previous statement: they are closed under several extensions/operations like set operations (e.g union, intersection), different flavours of non-determinism, regular look-ahead, two-directions (in particular, under reverse), etc. These properties are probably one of the reasons behind its fruitful connection with MSO logic or finite monoids [5, 16]. Unfortunately, this measure of robustness put copyless CRA in an undesirable position: our expressiveness result shows that copyless CRA are not closed under reverse and, furthermore, under any extensions regarding directions of its reading head. This implies that the behaviour of copyless CRA is asymmetric with respect to the input, which buried our expectations of a robust class for computing functions.

The lack of good closure properties for copyless CRA fuels our interest in its subclasses. We consider a natural fragment of copyless CRA, called *bounded alternation copyless* CRA (BAC). This class was previously introduced in [13] and characterized in terms of the so-called *Maximal Partition logic*. Interestingly, this fragment of copyless CRA does not lose much in terms of expressivity. In fact, most examples in [2] and in this paper are definable by BAC. Furthermore, all the structural toolkit introduced for copyless CRA also extend for this class. In contrast to copyless CRA, BAC are robust under several and natural extensions previously considered in [2, 3]. Specifically, we show that BAC are closed under unambiguous non-determinism, regular look-ahead and under reverse. These results emphasize that BAC is a promising computational model in the world of quantitative functions and show that there exists a rich theory of functions below the class of WA.

Organization. In Section 2 we introduce copyless CRA and some basic definitions. In Section 3 we introduce the normal form and analyze the content of registers during the runs of copyless CRA. Then we show in Section 4 that the class of copyless CRA is not closed under reverse. In Section 5 we define BAC and show some closure properties of this class. We conclude in Section 6 with possible directions for future research. Due to the page limit all full proofs are moved to the appendix, available online.

2 Preliminaries

In this section, we recall the definitions of cost register automata and the copyless restriction. We start with the definitions of expressions and substitutions over a semiring that are standard in this area.

Semirings and functions. A semiring is a structure $\mathbb{S} = (S, \oplus, \odot, \mathbb{0}, \mathbb{1})$ where $(S, \oplus, \mathbb{0})$ is a commutative monoid, $(S - \{\mathbb{0}\}, \odot, \mathbb{1})$ is a monoid, multiplication distributes over addition, and $\mathbb{0} \odot s = s \odot \mathbb{0} = \mathbb{0}$ for each $s \in S$. If the multiplication is commutative, we say that $\mathbb{S}$ is commutative. In this paper, we always assume that $\mathbb{S}$ is commutative. For the sake of simplicity, we usually denote the set of elements S by the name of the semiring $\mathbb{S}$. As standard examples of semirings we will consider the *semiring of natural numbers* $\mathbb{N}(+,\cdot) = (\mathbb{N}, +, \cdot, 0, 1)$, the *min-plus semiring* $\mathbb{N}_\infty(\min, +) = (\mathbb{N}_\infty, \min, +, \infty, 0)$ and the *max-plus semiring* $\mathbb{N}_{-\infty}(\max, +) = (\mathbb{N}_{-\infty}, \max, +, -\infty, 0)$ which are standard semirings in the field of weighted automata [8].

In this paper, we study the specification of functions from strings to values, namely, from Σ^* to $\mathbb{S}$. We say that a function $f : \Sigma^* \to \mathbb{S}$ is definable by a computational system $\mathcal{A}$ (e.g. weighted automaton, or CRA) if $f(w) = [\![\mathcal{A}]\!](w)$ for any $w \in \Sigma^*$, where $[\![\mathcal{A}]\!]$ is the semantics of $\mathcal{A}$ over strings. For any string w, we denote by w^r the reverse string. We say that a class of functions F is *closed under reverse* [2] if for every $f \in F$ there exists a function $f^r \in F$ such that $f^r(w^r) = f(w)$ for all $w \in \Sigma^*$.

Variables, expressions, and substitutions. Fix a semiring $\mathbb{S} = (S, \oplus, \odot, \mathbb{0}, \mathbb{1})$ and a set of variables $\mathcal{X}$ disjoint from S. We denote by $\mathrm{Expr}(\mathcal{X})$ the set of all syntactical *expressions* that can be defined with $\mathcal{X}$, constants in S, and the binary operations $\oplus, \odot$. For any expression $e \in \mathrm{Expr}(\mathcal{X})$ we denote by $\mathrm{Var}(e)$ the set of variables in e. We call an expression $e \in \mathrm{Expr}(\mathcal{X})$ a *ground expression* if $\mathrm{Var}(e) = \varnothing$. For any ground expression we define $[\![e]\!] \in \mathbb{S}$ to be the evaluation of e with respect to $\mathbb{S}$.

A *substitution* over $\mathcal{X}$ is defined as a mapping $\sigma : \mathcal{X} \to \mathrm{Expr}(\mathcal{X})$. We denote the set of all substitutions over $\mathcal{X}$ by $\mathrm{Subs}(\mathcal{X})$. A *ground substitution* σ is a substitution where the expression $\sigma(x)$ is ground for every $x \in \mathcal{X}$. Any substitution σ can be extended to a mapping $\hat{\sigma} : \mathrm{Expr}(\mathcal{X}) \to \mathrm{Expr}(\mathcal{X})$ such that, for every $e \in \mathrm{Expr}(\mathcal{X})$, $\hat{\sigma}(e)$ is the resulting expression $e[\sigma]$ of substituting each $x \in \mathrm{Var}(e)$ by the expression $\sigma(x)$. For example, if $\sigma(x) = 2x$ and $\sigma(y) = 3y$, and $e = x + y$, then $\hat{\sigma}(e) = 2x + 3y$. Using the extension $\hat{\sigma}$, we can define the composition substitution $\sigma_1 \circ \sigma_2$ of two substitutions σ_1 and σ_2 such that $\sigma_1 \circ \sigma_2(x) = \hat{\sigma}_1(\sigma_2(x))$ for each $x \in \mathcal{X}$.

A valuation is defined as a substitution of the form $\nu : \mathcal{X} \to \mathbb{S}$. We denote the set of all valuations over $\mathcal{X}$ by $\mathrm{Val}(\mathcal{X})$. Clearly, any valuation ν composed with a substitution σ defines a ground substitution, since $\mathrm{Var}(\nu \circ \sigma(x)) = \varnothing$ for all $x \in \mathcal{X}$.

In this paper, we say that two expressions e_1 and e_2 are equal (denoted by $e_1 = e_2$) if they are equal up to evaluation equivalence, that is, $[\![\nu \circ e_1]\!] = [\![\nu \circ e_2]\!]$ for every valuation $\nu \in \mathrm{Val}(\mathcal{X})$. Similarly, we say that two substitutions σ_1 and σ_2 are equal (denoted by $\sigma_1 = \sigma_2$) if $\sigma_1(x) = \sigma_2(x)$ for every $x \in \mathcal{X}$.

Cost register automata. A cost register automaton (CRA) over a semiring $\mathbb{S}$ [2] is a tuple $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu)$ where Q is a set of states, Σ is the input alphabet, $\mathcal{X}$ is a set of variables (we also call them registers), $\delta : Q \times \Sigma \to Q \times \mathrm{Subs}(\mathcal{X})$ is the transition function, q_0 is the initial state, $\nu_0 : \mathcal{X} \to \mathbb{S}$ is the initial valuation, and $\mu : Q \to \mathrm{Expr}(\mathcal{X})$ is the final

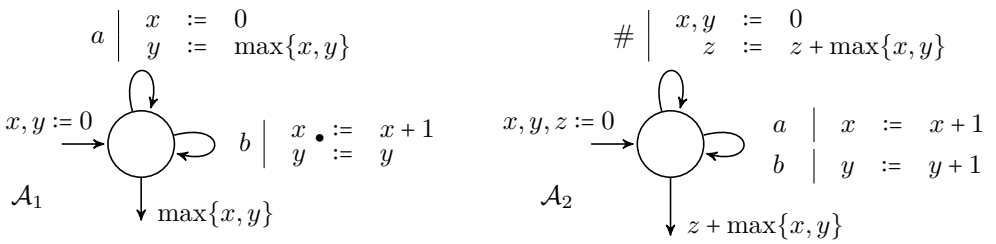


Figure 1 Examples of copyless cost-register automata.

output function. A configuration of $\mathcal{A}$ is a tuple (q, ν) where $q \in Q$ and $\nu \in \mathrm{Val}(\mathcal{X})$ represents the current values in the variables of $\mathcal{A}$. Given a string $w = a_1 \dots a_n \in \Sigma^*$, the run of $\mathcal{A}$ over w is a sequence of configurations: $(q_0, \nu_0) \xrightarrow{a_1} (q_1, \nu_1) \xrightarrow{a_2} \dots \xrightarrow{a_n} (q_n, \nu_n)$ such that, for every $1 \le i \le n$, $\delta(q_{i-1}, a_i) = (q_i, \sigma_i)$ and $\nu_i(x) = [\![\nu_{i-1} \circ \sigma_i(x)]\!]$ for each $x \in \mathcal{X}$. The output of $\mathcal{A}$ over w, denoted by $[\![\mathcal{A}]\!](w)$, is $[\![\nu_n \circ \mu(q_n)]\!]$.

The run of $\mathcal{A}$ over w can be equally defined in terms of ground expressions rather than values. A ground configuration of $\mathcal{A}$ is a tuple (q, ς) where $q \in Q$ and $\varsigma \in \mathrm{Subs}(\mathcal{X})$ is a ground substitution. Given a string $w = a_1 \dots a_n \in \Sigma^*$, the ground run of $\mathcal{A}$ over w is a sequence of ground configurations: $(q_0, \varsigma_0) \xrightarrow{a_1} \dots \xrightarrow{a_n} (q_n, \varsigma_n)$ such that for $1 \le i \le n$, $\delta(q_{i-1}, a_i) = (q_i, \sigma_i)$, $\varsigma_0 = \nu_0$ and $\varsigma_i(x) = \varsigma_{i-1} \circ \sigma_i(x)$ for each $x \in \mathcal{X}$. We denote the output ground expression of $\mathcal{A}$ over a string w by $|\mathcal{A}|(w) = \varsigma_n \circ \mu(q_n)$. Notice that, in contrast to ordinary runs, ground runs keep ground expressions as partial values of the run. It is easy to see that $[\![\mathcal{A}]\!](w) = [\![|\mathcal{A}|(w)]\!]$.

We define the transitive closure of the transition function $\delta^* : Q \times \Sigma^* \to Q \times \mathrm{Subs}(\mathcal{X})$, by induction over the word-length. Formally, $\delta^*(q, \epsilon) = (q, \mathrm{id})$ where ϵ is the empty word and $\mathrm{id}(x) = x$ for all $x \in \mathcal{X}$; and $\delta^*(q_1, w \cdot a) = (q_3, \sigma \circ \sigma')$, whenever $\delta^*(q_1, w) = (q_2, \sigma)$ and $\delta(q_2, a) = (q_3, \sigma')$. For a CRA $\mathcal{A}$ we define the set $\mathrm{Subs}(\mathcal{A})$ of all substitutions in $\mathcal{A}$ such that $\sigma \in \mathrm{Subs}(\mathcal{A})$ if, and only if, $\delta^*(p, w) = (q, \sigma)$ for some $p, q \in Q$ and $w \in \Sigma^*$.

Copyless restriction and copyless CRA. We say that an expression $e \in \mathrm{Expr}(\mathcal{X})$ is *copyless* if e uses every variable from $\mathcal{X}$ at most once. For example, $x \cdot (y + z)$ is copyless but $x \cdot y + x \cdot z$ is not copyless (because x is mentioned twice). Notice that the copyless restriction is a syntactical constraint over expressions. Furthermore, we say that a substitution σ is *copyless* if for every $x \in \mathcal{X}$ the expression $\sigma(x)$ is copyless and $\mathrm{Var}(\sigma(x)) \cap \mathrm{Var}(\sigma(y)) = \varnothing$ for every pair of different registers $x, y \in \mathcal{X}$. Copyless substitutions, similar to copyless expressions, are restricted in such a way that each variable is used at most once in the whole substitution.

A CRA $\mathcal{A}$ is called *copyless* if for every transition $\delta(q_1, a) = (q_2, \sigma)$ the substitution σ is copyless; and for every state $q \in Q$ the expression $\mu(q)$ is copyless, where μ is the output function of $\mathcal{A}$. In other words, every time $\mathcal{A}$ updates registers or outputs a value, each register can be used only once. It is straightforward that if $\mathcal{A}$ is copyless then all substitutions $\sigma \in \mathrm{Subs}(\mathcal{A})$ are also copyless. In the following, we give an example of a copyless CRA.

▶ **Example 1.** Let $\mathbb{S}$ be the max-plus semiring $\mathbb{N}_{-\infty}(\max, +)$ and $\Sigma = \{a, b\}$. Consider the function f_1 that for a given string $w \in \Sigma^*$ computes the longest substring of b's. This can be easily defined by the CRA $\mathcal{A}_1$ in Figure 1. The CRA $\mathcal{A}_1$ stores in the x-register the length of the last suffix of b's and in the y-register the length of the longest substring of b's seen so far. One can easily check that $\mathcal{A}_1$ is a copyless CRA. Indeed, each substitution is copyless and the final output expression $\max\{x, y\}$ is copyless as well.

▶ **Example 2.** Let $\mathbb{S}$ be the max-plus semiring $\mathbb{N}_{-\infty}(\max, +)$ and $\Sigma = \{a, b, \#\}$. Consider the function f_2 such that, for any $w \in \Sigma^*$ of the form $w_0 \# w_1 \# \dots \# w_n$ with $w_i \in \{a, b\}^*$, it

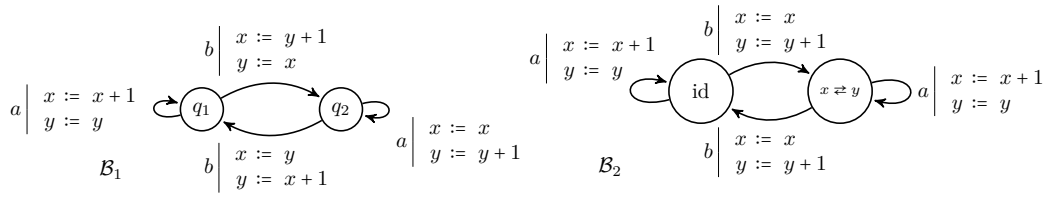


Figure 2 Examples of copyless cost-register automata regarding normal form.

computes the maximum number of a's or b's for each substring w_i (i.e. $\max\{|w_i|_a, |w_i|_b\}$) and then it sums these values over all substrings w_i, that is, $f_2(w) = \sum_{i=0}^{n} \max\{|w_i|_a, |w_i|_b\}$. One can check that the copyless CRA $\mathcal{A}_2$ in Figure 1 computes f_2. The copyless CRA $\mathcal{A}_2$ follows similar ideas to $\mathcal{A}_1$: the registers x and y count the number of a's and b's, respectively, in the longest suffix without # and the register z stores the partial output without considering the last suffix of a's and b's.

In the diagram of $\mathcal{A}_2$, we omit an assignment if a register is not updated (i.e. it keeps its previous value). For example, for the a-transition we omit the assignments $y := y$ and $z := z$ for the sake of presentation of the CRA. One should keep in mind these assignments because of the copyless restriction.

Trim assumption. For technical reasons, in this paper we assume that our finite automata and cost register automata are always *trim*, namely, all their states are reachable from some initial states (i.e., they are accessible) and they can reach some final state (i.e., they are co-accessible). It is worth noticing that verifying if a state is accessible or co-accessible is reduced to a reachability test in the transition graph [15] and this can be done in NLogSpace. Thus, we can assume without lost of generality that all our automata are trimmed.

3 Structural Properties of Copyless CRA

Fix a copyless CRA $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu)$. In this section we analyze the structure of $\mathcal{A}$ and develop some machinery that will be useful in Section 4. These results help to understand the internal structure of copyless CRA.

Normal form. Let $\mathcal{A}$ be a copyless CRA and let $\preceq$ be a predefined linear order over $\mathcal{X}$. We say that $\sigma \in \text{Subs}(\mathcal{A})$ is in normal form with respect to $\preceq$ if $x \preceq y$ for all $x \in X$ and all $y \in \text{Var}(\sigma(x))$. In other words, all variables mentioned in $\sigma(x)$ are greater or equal to x with respect to $\preceq$. Furthermore, we write that $\mathcal{A}$ is in *normal form* with respect to $\preceq$ if every $\sigma \in \text{Subs}(\mathcal{A})$ is in normal form with respect to $\preceq$. For example, the copyless CRA in Example 1 is in normal form with respect to the order $y \preceq x$. Throughout the paper we assume that every set of registers $\mathcal{X}$ is given with a linear order $\preceq_{\mathcal{X}}$. Instead of writing that $\mathcal{A}$ or $\sigma \in \text{Subs}(\mathcal{A})$ are in normal form with respect to $\preceq_{\mathcal{X}}$ we write in short that $\mathcal{A}$ is in normal form, and that σ is in normal form.

▶ **Example 3.** Consider the set of registers $\mathcal{X} = \{x, y\}$ with the order $x \preceq y$. The copyless CRA $\mathcal{B}_1$ in Figure 2 is not in normal form, because of the b-transitions. On the other hand, the copyless CRA $\mathcal{B}_2$ is in normal form. For both automata we omit the initial states, initial valuations and final output functions because they are not relevant for the discussion.

In the previous example, the automaton $\mathcal{B}_1$ uses the registers x and y to count the number of a's and b's. However, depending on the current state both registers have either

the number of a's or the number of b's. It is clear that one would like to avoid this type of behavior in a theoretical analysis. Intuitively, one register should always contain the number of a's and the other register the number of b's. One can clearly transform $\mathcal{B}_1$ to an automaton in normal form by exchanging the use of x and y in the transitions and encoding this permutation between registers in the states. This is precisely what the automaton $\mathcal{B}_2$ does. In the following result, we generalize this idea for all copyless CRA.

▶ **Proposition 4.** *For every copyless CRA $\mathcal{A}$ there exists a copyless CRA in normal form $\mathcal{A}'$ with the same set of registers such that they output the same ground expressions for all words and thus recognize the same function. The number of states in $\mathcal{A}'$ can be bounded exponentially in the size of the automaton $\mathcal{A}$.*

Proof (sketch). For a copyless CRA $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu)$, we define the set of states for $\mathcal{A}'$ by $Q' = Q \times \{\rho \mid \rho \text{ is a permutation of the set } \mathcal{X}\}$. Let $\delta(q, a) = (p, \sigma)$ be a transition. The idea is to find a permutation ρ such that the new substitution $\sigma'(x) = \sigma(\rho(x))$ is a substitution in normal form. Such a permutation always exists because $\mathcal{A}$ satisfies the copyless restriction. Intuitively, the substitutions σ' and σ store the same values in the registers, but the corresponding values might be in different registers. To enable $\mathcal{A}'$ to find the proper value of every register, we store the permutation ρ in the state p (i.e., the one defined by the substitution σ) and update the transition of $\mathcal{A}$ accordingly. ◀

Stable registers and reset substitutions. Let now $\mathcal{A}$ be a copyless CRA in normal form. During a run of $\mathcal{A}$ the content of its registers flows from higher to lower registers with respect to the total order $\preceq$. This does not necessarily mean that the content of all registers eventually reaches the $\preceq$-minimum register. For example, if all substitutions in $\mathcal{A}$ are of the form $\sigma(x) = x \oplus k$ for some $k \in \mathbb{S}$, then each register will store just its own content during the whole run. Intuitively, in this example each register is "stable" with respect to the content flow of $\mathcal{A}$, since each register never passes its value to lower registers. This idea motivates the notion of *stable registers* which are essential to understand the behaviour and output of copyless CRA. Let $\sigma \in \mathrm{Subs}(\mathcal{A})$ be a copyless substitution in normal form. We say that a register x is *σ-stable* (or stable on σ) if $x \in \mathrm{Var}(\sigma(x))$. More general, we write that a register x is *stable* in $\mathcal{A}$ if x is σ-stable for all substitutions $\sigma \in \mathrm{Subs}(\mathcal{A})$.

Stable registers play a crucial role in the behavior of copyless CRA. We show that they are the only registers whose value always depends on the whole word. Namely, we can always "reset" the value of non-stable registers to a constant. For instance, the automaton $\mathcal{A}_1$ in Example 1 resets the register x to 0 each time the symbol a is read. On the other side, the register y is stable and it cannot be reset to a constant. In fact its value only grows or remains the same during the run of $\mathcal{A}_1$.

We formalize this idea of reseting the content of registers as follows: a substitution $\sigma \in \mathrm{Subs}(\mathcal{A})$ is a *reset* substitution if $\mathrm{Var}(\sigma(x)) = \varnothing$ for all non σ-stable registers x. We say that a substitution $\sigma \in \mathrm{Subs}(\mathcal{A})$ is an *$\mathcal{A}$-reset substitution* if $\mathrm{Var}(\sigma(x)) = \varnothing$ for all non-stable registers in $\mathcal{A}$.

In the next result, we say that a copyless CRA is strongly connected if all states are mutually reachable in the transition graph of $\mathcal{A}$.

▶ **Proposition 5.** *Let $\mathcal{A}$ be a copyless and strongly connected CRA in normal form. Then for all $q, q' \in Q$ there exists $w^{q,q'} \in \Sigma^*$ and a substitution σ such that $\delta^*(q, w^{q,q'}) = (q', \sigma)$ and σ is an $\mathcal{A}$-reset substitution. Furthermore, there exists $w^{q,q'}$ containing all letters in Σ.*

The strongly connected restriction is a technical assumption to be sure that the result holds for every pair of states. Of course, the result also holds if we restrict to the strongly connected

components of $\mathcal{A}$. For instance in Example 1 it suffices to take the word $w = a$ given that the substitution defined by the a-transition is an $\mathcal{A}$-reset substitution.

Growing rate of stable registers in a cycle. The behavior of cycles in a computation model is always important; most of the decidability results can be derived from a good understanding of its cyclic behavior. Here, we study how the content of stables registers behaves through cycles. We say that a word $w \in \Sigma^*$ is a cycle over a state $q \in Q$ in $\mathcal{A}$ if $\delta^*(q, w) = (q, \sigma)$ for some substitution σ. Of course, the iteration of a cycle w (i.e. w^n for any $n \geq 1$) is also a cycle over q and it satisfies $\delta^*(q, w^n) = (q, \sigma^n)$ for any n. The next result shows that by iterating a cycle one can always "reset" the content of non σ-stable registers.

▶ **Lemma 6.** *Let $\mathcal{A}$ be in normal form and $\sigma \in \mathrm{Subs}(\mathcal{A})$ a copyless substitution in normal form. There exists $N \geq 0$ such that σ^N is a reset substitution.*

In the next proposition, we study the growing behavior of stable registers when a reset substitution is iterated. This will be useful to understand the behavior of copyless CRA inside their cycles. For this result we need an additional assumption that automata do not use $\mathbb{0}$ in their expressions. Moreover the expressions in our semirings that do not use $\mathbb{0}$ do not evaluate to $\mathbb{0}$. This is a technical assumption to avoid having $\mathbb{0}$ in the registers of CRA. Notice that the arctic-semiring $\mathbb{N}_{-\infty}(\max, +)$ satisfies $\mathbb{0} = -\infty$ and that the functions we defined in Section 4 do not output $-\infty$. Also in $\mathbb{N}_{-\infty}(\max, +)$ expressions defined with max, + and $\mathbb{N}$ do not evaluate to $-\infty$. Thus this assumption does not influence the results in Section 4.

▶ **Proposition 7.** *Let $\mathcal{A}$ be in normal form, $\sigma \in \mathrm{Subs}(\mathcal{A})$ a reset substitution and x a σ-stable register. Then there exist $c, d \in \mathbb{S}$ with $c \neq \mathbb{0}$ such that for every $i \geq 0$ we have:*

$$\sigma^{i+1}(x) \quad = \quad (c^i \odot \sigma(x)) \oplus \Big(d \odot \bigoplus_{j=0}^{i-1} c^j\Big).$$

In particular, for the semiring $\mathbb{N}_{-\infty}(\max, +)$ (i.e. $\odot = +$ and $\oplus = \max$) one can check that this is equivalent to $\sigma^{i+1}(x) = \max\{i \cdot c + \sigma(x),\ (i-1) \cdot c + d\}$, which shows that, intuitively, the σ-stable registers grow linearly when the substitution σ is cycling. For instance, consider the copyless CRA in Example 1 and let σ be the substitution defined by the b-transition. One can easily check that the register x is σ-stable since $\sigma(x) = x + 1$. Furthermore, $\sigma^{i+1}(x) = x + i + 1$, which is as in Proposition 7 (take $c = 1, d = 0$). This is precisely the expected behaviour of the copyless CRA on a long sequence of b's.

4 Inexpressibility of Copyless CRA

In this section, we use the techniques discussed previously to show a function that is not definable by any copyless CRA. This function can be defined by a weighted automaton, which will prove that copyless CRA are less expressive than weighted automata. Interestingly, the "reverse" of this function is definable by copyless CRA. From this, we will conclude that copyless CRA are not closed under reverse.

Consider the function $f_{\mathcal{B}}$ given by the copyless CRA $\mathcal{B}$ over $\Sigma = \{a, b, \#\}$ and $\mathbb{N}_{-\infty}(\max, +)$ in Figure 3. To understand $f_{\mathcal{B}}$, let us define the output of $\mathcal{B}$ formally. For any $w \in \Sigma^*$, let k be the number of #-symbols in w. Furthermore, for $0 < i < k$ let n_i and m_i be the number of a's and b's, respectively, between the i-th and $(i+1)$-th occurrence of # in w. Additionally, let n_0, m_0, n_k, m_k be the numbers of a's and b's before and after the first and last # in w.

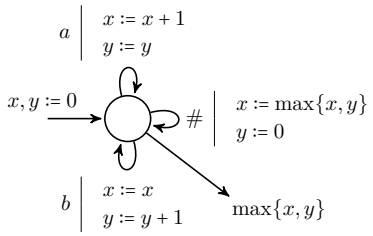


Figure 3 CRA $\mathcal{B}$.

By the definition of $\mathcal{B}$ in Figure 3, one can easily check that $f_{\mathcal{B}}$ is defined by $f_{\mathcal{B}}(\epsilon) = 0$, for the empty word ϵ, and

$$f_{\mathcal{B}}(w) \quad = \quad \max_{j \in \{-1,0,\ldots,k\}} \left\{ m_j + \sum_{i=j+1}^{k} n_i \right\} \tag{1}$$

for $w \neq \epsilon$, where $m_{-1} = 0$. From the above definition, one can also give a formal definition of $f^R_{\mathcal{B}}$, the reverse function of $f_{\mathcal{B}}$, which is given by reversing the role of n_i and m_j in (1). Formally, one can easily check that $f^R_{\mathcal{B}}$ is defined by $f^R_{\mathcal{B}}(\epsilon) = 0$, and

$$f^R_{\mathcal{B}}(w) \quad = \quad \max_{j \in \{0,\ldots,k,k+1\}} \left\{ \sum_{i=0}^{j-1} n_i + m_j \right\}, \tag{2}$$

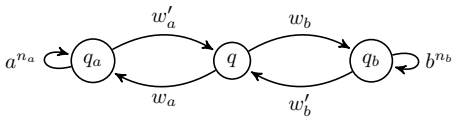
for $w \neq \epsilon$, where $m_{k+1} = 0$. The following theorem is the main result of this section.

▶ **Theorem 8.** *The function $f^R_{\mathcal{B}}$ is not recognizable by any copyless CRA.*

Proof (sketch). By contradiction, suppose that $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu)$ is a copyless CRA in normal form with respect to $\preceq$, which recognizes (2). To prove Theorem 8, we analyze the content of the registers after reading the word $w_0 \cdot w(j,s)$, where

$$w(j,s) = (w_a \cdot (a^{n_a \cdot j}) \cdot w'_a)^s \cdot w_b \cdot (b^{n_b \cdot j^2}) \cdot w'_b \cdot w_a \cdot (a^{n_a \cdot j}) \cdot w'_a.$$

for $j, s \in \mathbb{N}$. To understand the construction of $w(j,s)$, consider the following diagram, which is a fragment of $\mathcal{A}$.



First, the prefix w_0 is used to reach the connected component of $\mathcal{A}$ to make the Proposition 5 work and q is the state that $\mathcal{A}$ reaches after reading w_0. By a standard automata argument, one can find a state q_a such that for some number n_a the word a^{n_a} is a cycle over q_a. Similarly there is a state q_b and a number n_b such that b^{n_b} is a cycle over q_b. The words w_a and w_b go from q to the states q_a and q_b, respectively, and the words w'_a and w'_b go back to q. To analyze the run of $\mathcal{A}$ on $w(j,s)$ we need only the component above.

The next step is to study the asymptotic growing of $f^R_{\mathcal{B}}$ over $w_0 \cdot w(j,s)$ in terms of j and s. For this, the most important fragments of $w(j,s)$ are the subwords $a^{n_a \cdot j}$ and $b^{n_b \cdot j^2}$. To omit technical difficulties we do not discuss further here the remaining parts of $w(j,s)$. In a nutshell, the words w_a, w_b, w'_a, and w'_b are the reset words discussed in Proposition 5. Recall that the size of these words does not depend on j and s and they contain all symbols in Σ, in particular #.

We can estimate the output of $f_{\mathcal{B}}^R$ over $w_0 \cdot w(j,s)$ in terms of j and s by using (2). Given that the subwords $a^{n_a \cdot j}$ and $b^{n_b \cdot j^2}$ are separated by # (by Proposition 5), we conclude that the last sequence of a's is not included in the sum. This is because the maximum in (2) is achieved when m_j contains the subword $b^{n_b \cdot j^2}$ and thus the last sequence of a's will not be included in the final sum. Then one can prove that the value of the output of $f_{\mathcal{B}}^R$ over $w(j,s)$ is equal to

$$f_{\mathcal{B}}^R(w) = n_b \cdot j^2 + s \cdot n_a \cdot j + \mathcal{O}(1).$$

There are some a's and b's in the remaining parts of $w(j,s)$ and w_0 that contribute to the output, but these are constant terms that can be hidden in $\mathcal{O}(1)$.

By Lemma 6 we can take n_a big enough such that if $\delta^*(q_a, a^{n_a}) = (q_a, \sigma_a)$ then the substitution σ_a is a reset substitution. Let x be a stable register. By Proposition 7 we can approximate the expressions $\sigma_a^{j+1}(x)$ for j big enough. In the semiring $\mathbb{N}_{-\infty}(\max, +)$ the content of x when reading this loop can be estimated by

$$\sigma_a^{j+1}(x) = \max\{j \cdot c_x + \sigma_a(x) + \mathcal{O}(1),\ j \cdot c_x + \mathcal{O}(1)\}$$

for some constant c_x. Intuitively, this means that for j big enough after reading $a^{n_a \cdot j}$ the content of every stable register x is growing linearly to j or the content of x becomes linear in j. Similarly we can estimate the content of stable registers after reading b^{j^2}. The word $w(j,s)$ ends with w'_a, which resets the content of all non-stable registers to a constant, thus we can estimate their content with $\mathcal{O}(1)$.

Consider now j as a variable and s as a fixed parameter in $w(j,s)$. By the previous analysis the contents of the stable registers after reading $w(j,s)$ are quadratic functions in j, where the coefficient of j comes from the growth of registers when reading the subwords $a^{n_a \cdot j}$. We show that in every register either the coefficient of j is small compared to $s \cdot n_a$, or linear in $(s+1) \cdot n_a$. In both cases the output function cannot combine the registers to get a polynomial with $s \cdot n_a$ as a coefficient of j. Finally, for any $\mathcal{A}$ we can fix the parameter s to be big enough comparing to the number of registers. ◀

From Theorem 8 we immediately get the following corollary.

▶ **Corollary 9.** *There exists a semiring $\mathbb{S}$ such that the class of functions recognizable by copyless register automata over $\mathbb{S}$ is not closed under reverse.*

In [13] it is shown that copyless CRA are contained in weighted automata (WA). The previous results delimit the expressiveness of copyless CRA: they are strictly less expressive than WA. It is easy to define $f_{\mathcal{B}}^R$ by a polynomial ambiguous weighted automaton (i.e. a subclass of WA where the numbers of accepting runs is bounded by a polynomial function), which shows that copyless CRA is a strict subclass of weighted automata. Furthermore, in [2] Alur et al. introduced the class of *regular cost functions* defined in terms of streaming string-to-tree transducers (see [2] for more details). This class contained copyless CRA but it was left open whether this inclusion is strict or not. Since the class of regular cost functions is closed under reverse operation, Corollary 9 proves that copyless CRA are strictly contained in the class of regular cost functions.

▶ **Corollary 10.** *The class of functions defined by copyless CRA is strictly contained in the class of functions defined by weighted automata and in the class of regular cost functions.*

A natural question is whether the class of functions defined by copyless CRA is strictly contained in the class of functions defined by polynomial ambiguous weighted automata. We

conjecture that copyless CRA can express functions that are not expressible by polynomial ambiguous automata. A candidate function is the copyless CRA $\mathcal{A}_2$ in Example 2 for which it seems that one needs exponentially many runs in order to be computed by a weighted automaton.

5 Bounded Alternation Copyless CRA

Given that copyless CRA are not closed under reverse operation we look for a robust subclass of copyless CRA. The proof of Theorem 8 suggests that the alternation between semiring operations is the reason why copyless CRA cannot replicate the behavior of the CRA $\mathcal{B}$ in backward mode: $\mathcal{B}$ can sum the number of a- and b-symbols to maximize each time that a #-symbol is read, but it cannot do the same alternation of operations when the word is read in the other direction. This fact inspires the definition of *bounded alternation* copyless CRA, a strict subclass of copyless CRA where the output is restricted to expressions with bounded alternation. This class was proposed in [13] and characterized in terms of the so-called *Maximal Partition logic*. In this section, we show that bounded alternation copyless CRA has also good closure properties; this class is closed under unambiguous non-determinism, regular look-ahead and, moreover, under reverse.

The *alternation* of $e \in \mathrm{Expr}(\mathcal{X})$ is defined as the maximum number of switches between $\oplus$ and $\odot$ operations over all branches of the parse-tree of e. Formally, let $\otimes \in \{\oplus, \odot\}$ and $\bar{\otimes}$ be the dual operation of $\otimes$ in $\mathbb{S}$. We define the set of expressions $\mathrm{Expr}_0^{\otimes}(\mathcal{X})$ with 0-alternation by $\mathrm{Expr}_0^{\otimes} = \mathcal{X} \cup \mathbb{S}$. For any $N \geq 1$, we define the set of expressions $\mathrm{Expr}_N^{\otimes}(\mathcal{X})$ as the $\otimes$-closure of $\mathrm{Expr}_{N-1}^{\bar{\otimes}}(\mathcal{X})$, namely, $\mathrm{Expr}_N^{\otimes}(\mathcal{X})$ is the minimal set of expressions that contains $\mathrm{Expr}_{N-1}^{\bar{\otimes}}(\mathcal{X})$ and satisfies that $e_1 \otimes e_2 \in \mathrm{Expr}_N^{\otimes}(\mathcal{X})$ whenever $e_1, e_2 \in \mathrm{Expr}_N^{\otimes}(\mathcal{X})$. We define $\mathrm{Expr}_N(\mathcal{X}) = \mathrm{Expr}_N^{\oplus}(\mathcal{X}) \cup \mathrm{Expr}_N^{\odot}(\mathcal{X})$, namely, the set of all expressions with alternation bounded by N.

We say that a copyless CRA $\mathcal{A}$ has *bounded alternation* if the number of alternations of all ground expressions output by $\mathcal{A}$ is uniformly bounded by a constant, that is, there exists N such that $|\mathcal{A}|(w) \in \mathrm{Expr}_N(\mathcal{X})$ for every $w \in \Sigma^*$. A copyless CRA $\mathcal{A}$ is called a bounded alternation copyless CRA (BAC) if $\mathcal{A}$ has bounded alternation. All the examples of copyless CRA presented in Section 2 have bounded alternation. For example, one can easily check that the alternation of the copyless CRA in Example 1 is bounded by 2.

The alternation of any expression can be easily derived just by counting what is the maximum number of alternation between $\oplus$ and $\odot$. However, it is not directly clear how to check if a copyless CRA has bounded alternation from its definition. We show that this semantical property can be verified in NLogSpace in the size of a copyless CRA.

▶ **Proposition 11.** *The problem of deciding whether a copyless CRA has bounded alternation can be computed in* NLogSpace. *Furthermore, if a copyless CRA has bounded alternation, the alternation is bounded by* $|Q| \cdot \max\{\, \mathrm{alt}(\sigma) \mid \exists p, q \in Q.\ \delta(q, a) = (p, \sigma) \,\}$ *where* $\mathrm{alt}(\sigma)$ *is the alternation of* σ.

Closure under unambiguous non-determinism. We first extend the model of bounded alternation copyless CRA with non-determinism. The class CRA was designed as a deterministic model in contrast to weighted automata, where non-determinism plays a crucial role. Thus, we restrict non-determinism to be unambiguous, namely, we allow many runs over a word but at most one accepting run, which defines the output. Formally, a non-deterministic CRA is a tuple $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \Delta, I_0, V_0, F, \mu)$ where Q, Σ, $\mathcal{X}$, and μ are defined as before, $\Delta \subseteq Q \times \Sigma \times Q \times \mathrm{Subs}(\mathcal{X})$ is a finite transition relation, $I_0 \subseteq Q$ is a set of initial states,

$V_0 : I_0 \to \mathrm{Val}(\mathcal{X})$ assigns an initial valuation for each initial state, and F is the set of final states. Additionally, we assume that for every $q, q' \in Q$ and $a \in \Sigma$ there exists at most one $\sigma \in \mathrm{Subs}(\mathcal{X})$ such that $(q, a, q', \sigma) \in \Delta$. Given a string $w = a_1 \dots a_n \in \Sigma^*$, a run of $\mathcal{A}$ over w is a sequence of configurations: $(q_0, \nu_0) \xrightarrow{a_1} (q_1, \nu_1) \xrightarrow{a_2} \dots \xrightarrow{a_n} (q_n, \nu_n)$ such that $q_0 \in I_0$, $\nu_0 = V_0(q_0)$, and for $1 \leq i \leq n$, $(q_{i-1}, a_i, q_i, \sigma_i) \in \Delta$ and $\nu_i(x) = [\![\nu_{i-1} \circ \sigma_i(x)]\!]$ for each $x \in \mathcal{X}$. Furthermore, a run of $\mathcal{A}$ over w is an accepting run if $q_n \in F$. We say that $\mathcal{A}$ is unambiguous if for every $w \in \Sigma^*$ there exists exactly one accepting run of $\mathcal{A}$ over w. The output of $\mathcal{A}$ over w is defined as $[\![\mathcal{A}]\!](w) = [\![\nu_n \circ \mu(q_n)]\!]$ where (q_n, ν_n) is the final configuration of the only accepting run of $\mathcal{A}$ over w. The definitions of unambiguous copyless CRA and unambiguous BAC are straightforward restrictions of this definition.

We do not know whether for each unambiguous copyless CRA there is an equivalent deterministic copyless CRA. However, this is true when we assume bounded alternation.

▶ **Theorem 12.** *Let $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, I_0, V_0, F, \mu)$ be an unambiguous BAC whose alternation is bounded by N. There exists a deterministic BAC $\mathcal{A}'$ that computes the same function as $\mathcal{A}$, that is, $[\![\mathcal{A}]\!](w) = [\![\mathcal{A}']\!](w)$ for every $w \in \Sigma^*$. Furthermore, the number of states of $\mathcal{A}'$ is of size $2^{\mathcal{O}(|Q|^3 \cdot |\mathcal{X}|^5 \cdot N^2)}$ and the number of registers in $\mathcal{A}'$ is of size $\mathcal{O}(|Q| \cdot |\mathcal{X}|^2 \cdot N)$.*

Proof (sketch). The construction goes by storing in $\mathcal{A}'$ the tree of $\mathcal{A}$-runs over the input. Of course, if we would keep all possible runs, then the set of states of $\mathcal{A}'$ would be infinite. A well-known property of unambiguous CRA is that, for each prefix, there is at most $|Q|$ possible runs and, furthermore, the last state of each run is different. Thus, $\mathcal{A}'$ can keep a tree structure in the states representing the runs where each branch represents a run and the number of branches is bounded by $|Q|$.

Unfortunately, we cannot do the same trick by naively keeping copies of the registers for each $\mathcal{A}$-run (recall that $\mathcal{A}'$ must be copyless). To overcome this problem, $\mathcal{A}'$ will do partial evaluation of the runs and postpone the use of common registers whenever it is possible by exploiting the copyless restriction and bounded alternation of $\mathcal{A}$. The full construction is in the appendix, available online. ◀

Closure under regular look-ahead. Our next extension of the CRA model is based on regular look-ahead, namely, transitions that can check regular properties over the input. Regular look-ahead has been extensively studied for finite automata [9, 10] and has been stated as a key property of a model for computing non-boolean functions [3, 2]. Let REG_Σ be the set of all regular languages over Σ. A CRA with regular look-ahead (CRA-RLA) is a tuple $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \Delta, q_0, \nu_0, \mu)$ where Q, Σ, $\mathcal{X}$, q_0, ν_0, and μ are defined as before and $\Delta : Q \times \mathrm{REG}_\Sigma \rightharpoonup Q \times \mathrm{Subs}(\mathcal{X})$ is a partial transition function. Given a string $w = a_1 \dots a_n \in \Sigma^*$, the run of $\mathcal{A}$ over w is a sequence of configurations: $(q_0, \nu_0) \xrightarrow{L_1} (q_1, \nu_1) \xrightarrow{L_2} \dots \xrightarrow{L_n} (q_n, \nu_n)$ such that for $1 \leq i \leq n$, $\Delta(q_{i-1}, L_i) = (q_i, \sigma_i)$, $a_i \dots a_n \in L_i$ and $\nu_i(x) = [\![\nu_{i-1} \circ \sigma_i(x)]\!]$ for each $x \in \mathcal{X}$. The output of $\mathcal{A}$ over w is defined as usual, i.e. $[\![\mathcal{A}]\!](w) = [\![\nu_n \circ \mu(q_n)]\!]$. To keep determinism, we also restrict Δ as follows: for a fixed state q let $\Delta(q, L_1) = (q_1, \sigma_1), \Delta(q, L_2) = (q_2, \sigma_2), \dots, \Delta(q, L_k) = (q_k, \sigma_k)$ be all transitions with q in the first coordinate and $L_1, \dots, L_k \in \mathrm{REG}_\Sigma$. Then the languages $L_1, \dots, L_k$ are pairwise disjoint (i.e. $L_i \cap L_j = \varnothing$). Note that $\mathcal{A}$ is always deterministic, i.e. after reading the remaining suffix the automaton is forced to take at most one available transition. The restrictions to copyless and bounded alternation CRA-RLA are defined as expected.

Like for unambiguous copyless CRA, we do not know if extending copyless CRA with regular look-ahead results in a more expressive model. Assuming bounded alternation we prove that the resulting class of functions is the same.

▶ **Theorem 13.** *For every BAC $\mathcal{A}$ with regular look-ahead there exists a BAC $\mathcal{A}'$ without regular look-ahead that computes the same function, that is, $[\![\mathcal{A}]\!](w) = [\![\mathcal{A}']\!](w)$ for every $w \in \Sigma^*$. Furthermore, the number of states and the number of registers of $\mathcal{A}'$ is double-exponential and polynomial, respectively, in the size of $\mathcal{A}$.*

Proof (sketch). The proof goes by constructing an unambiguous BAC $\mathcal{A}''$ that guesses the sequence of transitions $\Delta(p, L) = (q, \sigma)$ and checks later that L is satisfied over the suffix. If L is given by a finite deterministic automaton $\mathcal{R}$, this check can be done unambiguously by keeping the current state of each $\mathcal{R}$ and removing repeated runs. By Theorem 12, we know that $\mathcal{A}''$ can be converted into an equivalent deterministic BAC $\mathcal{A}'$. ◀

Closure under reverse. We finish this section proving that, in contrast to copyless CRA (see Section 4), BAC are closed under reverse. Recall that a subclass $\mathcal{C}$ of CRA is closed under reverse if for every $\mathcal{A} \in \mathcal{C}$ there exists $\mathcal{A}' \in \mathcal{C}$ such that $[\![\mathcal{A}]\!](w) = [\![\mathcal{A}']\!](w^r)$ for every $w \in \Sigma^*$.

▶ **Theorem 14.** *For every BAC $\mathcal{A}$ there exists a BAC $\mathcal{A}'$ that computes the reverse function of $\mathcal{A}$, that is, $[\![\mathcal{A}]\!](w) = [\![\mathcal{A}']\!](w^r)$ for every $w \in \Sigma^*$. Furthermore, the number of states is double-exponential and the number of registers is polynomial in the size of $\mathcal{A}$.*

We conclude this section by stressing the robustness of bounded alternation copyless CRA: they are closed under unambiguous non-determinism, regular look-ahead, and reverse operation. Notice that all the structural properties studied in Section 3 also apply to this class, in particular, the results regarding normal form and stable registers.

6 Conclusions and Future Work

In this paper, we studied structural properties, expressiveness and closure properties of copyless CRA. In particular, we showed that the class of functions recognized by copyless CRA are not closed under reverse. To recover the closure properties of CRA, we proposed the subclass of bounded alternation copyless CRA (BAC). We prove that BAC are closed under unambiguous non-determinism, regular look-ahead, and reverse. For general copyless CRA, we do not know whether this class is closed under unambiguous non-determinism or regular look-ahead. A positive answer would be surprising, since unambiguous automata are often trivially closed under the reverse operation (e.g. finite automata or weighted automata).

To study whether a subclass of CRA is closed under reverse, one could approach the problem in a more general way. A natural extension of BAC is to enhance the model with the ability of moving in both directions. Our results do not give a straightforward argument that BAC is closed under two-way extension, but we believe that this could be shown by exploiting the copyless restriction and bounded alternation similar as in the proof of Theorem 13.

The most important task for future work is to study the decidability properties of copyless CRA or BAC. The classical questions for computational models, like boundedness or equivalence of two automata, remain unanswered. We hope that the machinery developed in Section 3 is a first step towards answering these questions.

Acknowledgments. We thank the anonymous referees for their helpful comments. The first author was partially supported by Poland's National Science Center grant 2013/09/B/ST6/01575. The last author was supported by CONICYT + PAI / Concurso Nacional Apoyo al Retorno de Investigadores/as desde el extranjero – Convocatoria 2013 + 821320001 and by the Millenium Nucleus Center for Semantic Web Research under grant NC120004.

References

1 Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? In *ATVA*, pages 482–491, 2011. doi:10.1007/978-3-642-24372-1_37.

2 Rajeev Alur, Loris D'Antoni, Jyotirmoy Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *LICS*, pages 13–22. IEEE Computer Society, 2013.

3 Rajeev Alur and Loris D'Antoni. Streaming tree transducers. In *Automata, Languages, and Programming*, pages 42–53. Springer, 2012.

4 Rajeev Alur and Mukund Raghothaman. Decision problems for additive regular functions. In *Automata, Languages, and Programming*, pages 37–48. Springer, 2013.

5 J. Richard Buchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960.

6 Karel Culik II and Jarkko Kari. Image compression using weighted finite automata. In *Mathematical Foundations of Computer Science 1993*, pages 392–402. Springer, 1993.

7 Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theor. Comput. Sci.*, 380(1-2):69–86, 2007.

8 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer, 1st edition, 2009.

9 Joost Engelfriet. Top-down tree transducers with regular look-ahead. *Mathematical Systems Theory*, 10(1):289–303, 1976.

10 Joost Engelfriet and Sebastian Maneth. Macro tree transducers, attribute grammars, and mso definable tree translations. *Information and Computation*, 154(1):34–91, 1999.

11 Stephan Kreutzer and Cristian Riveros. Quantitative monadic second-order logic. In *LICS*, pages 113–122. IEEE Computer Society, 2013.

12 Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. In *ICALP*, pages 101–112, 1992. doi:10.1007/3-540-55719-9_67.

13 Filip Mazowiecki and Cristian Riveros. Maximal Partition Logic: Towards a Logical Characterization of Copyless Cost Register Automata. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, volume 41 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 144–159, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2015.144.

14 Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

15 Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1993.

16 M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res. Dev.*, 3(2):114–125, April 1959. doi:10.1147/rd.32.0114.

17 M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.

Algorithmic Statistics, Prediction and Machine Learning*

Alexey Milovanov

Moscow State University, Moscow, Leninskie gory, MSU, Russia
almas239@gmail.com

Abstract

Algorithmic statistics considers the following problem: given a binary string x (e.g., some experimental data), find a "good" explanation of this data. It uses algorithmic information theory to define formally what is a good explanation. In this paper we extend this framework in two directions.

First, the explanations are not only interesting in themselves but also used for prediction: we want to know what kind of data we may reasonably expect in similar situations (repeating the same experiment). We show that some kind of hierarchy can be constructed both in terms of algorithmic statistics and using the notion of a priori probability, and these two approaches turn out to be equivalent (Theorem 5).

Second, a more realistic approach that goes back to machine learning theory, assumes that we have not a single data string x but some set of "positive examples" $x_1, \ldots, x_l$ that all belong to some unknown set A, a property that we want to learn. We want this set A to contain all positive examples and to be as small and simple as possible. We show how algorithmic statistic can be extended to cover this situation (Theorem 11).

1998 ACM Subject Classification H.1.1 Systems and Information Theory

Keywords and phrases algorithmic information theory, minimal description length, prediction, kolmogorov complexity, learning

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.54

1 Introduction and Notation

Let x be a binary string, and let A be a finite set of binary strings containing x. Considering A as an "explanation" (statistical model) for x, we want A to be as simple and small as possible (the smaller A is, the more specific the explanation is). This approach can be made formal in the framework of algorithmic information theory, where the notion of algorithmic (Kolmogorov) complexity of a finite object (a string or a set encoded as a binary string in a natural way) is defined.

The definition and basic properties of Kolmogorov complexity can be found in the textbooks [5], [8], for a short survey see [6]. Informally Kolmogorov complexity of a string x is defined as the minimal length of a program that produces x. This definition depends on the programming language, but there are optimal languages that make the complexity minimal up to a constant; we fix one of them and denote the complexity of x by $C(x)$.

We also use another basic notion of the algorithmic information theory, the *discrete a priory probability*. Consider a probabilistic machine A without input that outputs some binary string and stops. It defines a probability distribution on binary strings: $m_A(x)$ is

* This work was partially supported by RaCAF ANR-15-CE40-0016-01 grant and RFBR grant 16-01-00362.

© Alexey Milovanov;
licensed under Creative Commons License CC-BY

the probability to get x as the output of A. (The sum of $m_A(x)$ over all x can be less than 1 since the machine can also hang.) The functions m_A can be also characterized as lower semicomputable semimeasures (non-negative real-valued functions $m(\cdot)$ on binary strings such that the set of pairs (r, x) where r is a rational number, x is a binary string and $r < m(x)$, is computably enumerable, and $\sum_x m(x) \leqslant 1$). There exists a universal machine U such that m_U is maximal (up to $O(1)$-factor) among all m_A. We fix some U with this property and call $m_U(x)$ the *discrete a priori probability of* x, denoted as $\mathbf{m}(x)$. The function $\mathbf{m}$ is closely related to Kolmogorov complexity. Namely, the value $-\log_2 \mathbf{m}(x)$ is equal to $C(x)$ with $O(\log C(x))$-precision.

Now we can define two parameters that measure the quality of a finite set A as a model for its element x: the complexity $C(A)$ of A and the binary logarithm $\log |A|$ of its size. The first parameter measures how simple is our explanation; the second one measures how specific it is. We use binary logarithms to get both parameters in the same scale: to specify an element of a set of size N we need $\log N$ bits of information.

There is a trade-off between two parameters. The singleton $A = \{x\}$ is a very specific description, but its complexity may be high. On the other hand, for a n-bit string x the set $A = \mathbb{B}^n$ of all n-bit strings is simple, but it is large. To analyze this trade-off, following [3, 4], let us note that every set A containing x leads to a *two-part description of* x: first we specify A using $C(A)$ bits, and then we specify x by its ordinal number in A, using $\log |A|$ bits. In total we need $C(A) + \log |A|$ bits to specify x (plus logarithmic number of bits to separate two parts of the description). This gives the inequality

$$C(x) \leqslant C(A) + \log |A| + O(\log C(A))$$

(the length of the optimal description, $C(x)$, does not exceed the length of any two-part description). The difference

$$\delta(x, A) = C(A) + \log |A| - C(x)$$

is called *optimality deficiency* of A (as a model for x). As usual in algorithmic statistic, all our statements are made with logarithmic precision (with error tolerance $O(\log n)$ for n-bit strings), so we ignore the logarithmic terms and say that $\delta(x, A)$ is positive and measures the overhead caused by using two-part description based on A instead of the optimal description for x.

Note that this overhead $\delta(x, A)$ is zero for $A = \{x\}$, so the question is whether we can obtain A that is simpler than x but maintains $\delta(x, A)$ reasonably small. This trade-off is reflected by a curve called sometimes that the *profile* of x; this profile can be defined also in terms of randomness deficiency (the notion of (α, β)-stochasticity introduced by Kolmogorov, see [8], [9], [7]), and in terms of time-bounded Kolmogorov complexity (the notion of depth, see [4]).

In our paper we apply these notions to an analysis of the prediction and learning. In Section 2 we consider, for a given string x, all "good" explanations and consider their union. Elements of this union are strings that can be reasonably expected when the experiment that produced x is repeated. We show that this union has another equivalent definition in terms of a priori probability (Theorem 5).

In Subsection 2.5 we consider a situation where we start with several data strings $x_1, \ldots, x_l$ obtained in several independent experiments of the same type. We show that all the basic notions of algorithmic statistics can be extended (with appropriate changes) to this framework, as well as Theorem 5.

2 Prediction Hierarchy

2.1 Algorithmic Prediction

Assume that we have some experimental data represented as a binary string x. We look for a good statistical model for x and find some set A that has small optimality deficiency $\delta(x, A)$. If we believe in this model, we expect only elements from A as outcomes when the same experiment is repeated. The problem, however, is that many different models with small optimality deficiency may exist for a given x, and they may contain different elements. If we want to cover all the possibilities, we need to consider the union of all these sets, so we get the following definition. In the following definition we assume that x is a binary string of length n, and all the sets A also contain only strings of length n.

▶ **Definition 1.** Let $x \in \mathbb{B}^n$ be a binary string and let d be some integer. The union of all finite sets of strings $A \subset \mathbb{B}^n$ such that $x \in A$ and $\delta(x, A) \leqslant d$ is called *algorithmic prediction d-neighborhood of* x.

Obviously d-neighborhood increases as d increases. It becomes trivial (contains all n-bit strings) when $d = n$ (then $\mathbb{B}^n$ is one of the sets A in the union).

▶ **Example 2.** If $x = 0 \dots 0$ (the strings consisting of n zeros), then x' belongs to d-neighborhood of x iff $C(x') \lesssim d$

▶ **Example 3.** If x is a random string of length n (i. e. $C(x) \approx n$) then the d-neighborhood of x contains all strings of length n provided d is greater than some function of order $O(\log n)$.

2.2 Probabilistic Prediction

There is another natural approach to prediction. Since we treat the experiment as a black box (the only thing we know is its outcome x), we assume that the possible models $A \subset \mathbb{B}^n$ are distributed according to their a priori probabilities, and consider the following two-stage process. First, a finite set is selected randomly: a non-empty set A is chosen with probability $\mathbf{m}(A)$ (note that a priori probability can be naturally defined for finite sets via some computable encoding). Second, a random element x of A is chosen uniformly. In this process every string x is chosen with probability

$$\sum_{A \ni x} \mathbf{m}(A)/|A|,$$

and it is easy to see that this probability is equal to $\mathbf{m}(x)$ up to a $O(1)$-factor. Indeed, the formula above defines a lower semicomputable function of x, so it does not exceed $\mathbf{m}(x)$ more than by $O(1)$-factor. On the other hand, if we restrict the sum to the singleton $\{x\}$, we already get $\mathbf{m}(x)$ up to a constant factor. So this process gives nothing new in terms of the final output distribution on the outcomes x. Still the advantage is that we may consider, for a given pair of strings x and y, the conditional probability

$$p(y|x) = \Pr[y \in A \mid \text{the output of the two-stage process is } x]\,.$$

In other words, by definition

$$p(y|x) = \frac{\sum_{A \ni x,y} \mathbf{m}(A)/|A|}{\sum_{A \ni x} \mathbf{m}(A)/|A|}. \tag{1}$$

As we have said, the denominator equals $\mathbf{m}(x)$ up to $O(1)$-factor, so

$$p(y|x) = \frac{\sum_{A \ni x,y} \mathbf{m}(A)/|A|}{\mathbf{m}(x)} \tag{2}$$

up to $O(1)$-factor. Having some string x and some threshold d, we now can consider all strings y such that $p(y|x) \geqslant 2^{-d}$ (we use the logarithmic scale to facilitate the comparison with algorithmic prediction). These strings could be considered as plausible ones to appear when repeating the experiment of unknown nature that once gave x.

Our main result shows that this approach is essentially equivalent to the algorithmic prediction. By a technical reason we have to change slightly the random process that defines $p(y|x)$. Namely, it is strange to consider models that are much more complex than x itself, so we consider only sets A whose complexity does not exceed $\mathrm{poly}(n)$; any sufficiently large polynomial can be used here (in fact, $4n$ is enough). So we assume that the sums in (1) and (2), and in similar formulas in the sequel are always restricted to sets $A \subset \mathbb{B}^n$ that have complexity at most $4n$, and take this modified version of (1) as a final definition for $p(y|x)$.

▶ **Definition 4.** Let x be a binary string and let d be an integer. The set of all strings y such that $p(y|x) \geqslant 2^{-d}$ is called *probabilistic prediction d-neighborhood of x*.

We are ready to state the main result of this section.

▶ **Theorem 5.**

(a) *For every n-bit string x and for every d the algorithmic prediction d-neighborhood is contained in probabilistic prediction $d + O(\log n)$-neighborhood.*

(b) *For every n-bit string x and for every d the probabilistic prediction d-neighborhood of x is contained in algorithmic prediction $d + O(\log n)$-neighborhood.*

The next section contains the proof of this result; later we show some its possible extensions.

2.3 The Proof of the Theorem 5

Proof of (a). This direction is simple. Assume that some string y belongs to the algorithmic prediction d-neighborhood of x, i.e., there is a set A containing x and y such that $C(A) + \log|A| \leqslant C(x) + d$. We may assume without loss of generality that $d \leqslant 2n$ otherwise all n-bit string belong to probabilistic prediction d-neighborhood of x (take $A = \mathbb{B}^n$). Then the inequality for $C(A) + \log|A|$ implies that complexity of A does not exceed $4n$, so the set A is included in the sum. This inequality implies also that

$$\frac{\mathbf{m}(A)/|A|}{\mathbf{m}(x)} \geqslant 2^{-d-O(\log n)}$$

(as we have said, $-\log \mathbf{m}(u)$ equals $C(u) + O(\log C(u))$). This fraction is one of terms in the sum that defines $p(y|x)$, so y belongs to the probabilistic prediction $d+O(\log n)$-neighborhood of x. ◀

Before proving the second part (b), we need to prove a technical lemma. It is inspired by [11, Lemma 6] where it was shown that if a string belongs to many sets of bounded complexity, then one of them has even smaller complexity. We generalize that result as follows.

▶ **Lemma 6.** *Assume that sets L and R consist of finite objects (in particular, Kolmogorov complexity $C(v)$ is defined for $v \in L$). Assume that R is has at most 2^n elements. Let G be*

a finite bipartite graph where L and R are the sets of its left and right nodes, respectively. Assume that a right node x has at least 2^k neighbors of Kolmogorov complexity at most i. Then x has a neighbor of complexity at most $i - k + O(C(G) + \log(k + i + n))$. Here $C(G)$ stands for the length of the shortest program that given any $v \in L$ outputs a list of its neighbors.

Proof. Let us enumerate left nodes that have complexity at most i. We start a selection process: some of them are marked (=selected) immediately after they appear in the enumeration. This selection should satisfy the following requirements:

- at any moment every right node that has at least 2^k neighbors among enumerated nodes, has a marked neighbor;
- the total number of marked nodes does not exceed $2^{i-k}p(i,k,n)$ for some polynomial p (fixed in advance).

If we have such a selection strategy of complexity $C(G) + O(\log(i + k + n))$, this implies that the right node x has a neighbor of complexity at most

$$i - k + O(C(G) + \log(k + i + n)),$$

namely, any its marked neighbor (that marked neighbor can be specified by its number in the list of all marked nodes).

To prove the existence of such a strategy, let us consider the following game. The game is played by two players, who alternate moves. The maximal number of moves is 2^i. At each move the first player plays a left node, and the second player replies saying whether she marks that node or not. The second player loses if the number of marked nodes exceeds $2^{i-k+1}(n+1)\ln 2$ or if after some of her moves there exists a right node y that has at least 2^k neighbors among the nodes chosen by the first player but has no marked neighbor. (The choice of the bound $2^{i-k+1}(n+1)\ln 2$ will be clear from the probabilistic estimate below.) Otherwise she wins.

Assume first that the set L of left nodes is finite (recall that the set of right nodes is finite by assumption). Then our game is a finite game with full information, an hence one of the players has a winning strategy. We claim that the second player can win. If it is not the case, the first player has a winning strategy. We get a contradiction by showing that the second player has a probabilistic strategy that wins with positive probability against any strategy of the first player. So we assume that some strategy of the first player is fixed, and consider the following simple probabilistic strategy of the second player: every node presented by the first player is marked with probability $p = 2^{-k}(n+1)\ln 2$. The expected number of marked nodes is $p2^i = 2^{i-k}(n+1)\ln 2$. By Markov's inequality, the number of marked nodes exceeds the expectation by a factor of 2 with probability less than $\frac{1}{2}$. So it is enough to show that the second bad case (after some move there exists a right node y that has 2^k neighbors among the nodes chosen by first player but has no marked neighbor) happens with probability at most $\frac{1}{2}$.

For that, it is enough to show that for every node right node y the probability of this bad event is less than $\frac{1}{2}$ divided by the number $|R|$ of right nodes. Let us estimate this probability. If y has 2^k (or more) neighbors, the second player had (at least) 2^k chances to mark its neighbor (when these 2^k nodes were presented by the first player), and the probability to miss all 2^k these chances is at most $(1-p)^{2^k}$. The choice of p guarantees that this probability is less than $2^{-n-1} \leqslant (1/2)/|R|$. Indeed, using the bound $1 - x \leqslant e^{-x}$, it is easy to show that

$$(1-p)^{2^k} \leqslant e^{\ln 2 \cdot (-n-1)} = 2^{-n-1}.$$

We have proven that the winning strategy exists but have not yet estimated is complexity. A winning strategy can be found be an exhaustive search among all the strategies. The set of all strategies is finite and the game is specified by G, i and k. Therefore the complexity of the first found winning strategy is at most $C(G) + O(\log(i + k))$.

Thus the Lemma 6 is proven in the case when L is a finite set. To extend the proof to general case, notice that the winning condition depends only on the neighborhood of each left node. The worst graph for the the second player is the following "model" graph. It has 2^{2^n+i} left nodes and 2^n right nodes and each of 2^{2^n} possible neighborhoods is shared by 2^i left nodes. A winning strategy for such a graph can be found from n, i and k and hence its complexity is logarithmic in $n + i + k$. That strategy can be translated to the game associated with the initial graph, this translation increases the complexity by $C(G)$, as we have to translate each left node played by the first player to a left node of the model graph. ◀

Having in mind future applications in Subsection 2.4, we will consider in the next statement an arbitrary decidable family $\mathcal{A}$ of finite sets though in this section we need only the case when $\mathcal{A}$ contains all finite sets.

▶ **Corollary 7.** *Let $\mathcal{A}$ be a decidable family of finite sets. Assume that $x_1, \ldots, x_l$ are strings of length n. Denote by $\mathcal{A}^n_m$ all subsets of $\mathbb{B}^n$ of complexity at most m. Then the sum*

$$S := \sum_{A \in \mathcal{A}^n_m,\ x_1,\ldots,x_l \in A} \frac{\mathbf{m}(A)}{|A|}$$

equals to its maximal term up to a factor of $2^{O(\log(n+m+l))}$.

Proof of the corollary. Let M denote the maximal term in the sum S. Obviously the sum S is equal to the sum over $i \leqslant m$ and $j \leqslant n$ of sums

$$\sum_{\substack{A \in \mathcal{A}^n_m \\ C(A)=i \\ \log|A|=j \\ x_1,\ldots,x_l \in A}} \frac{\mathbf{m}(A)}{|A|}. \tag{3}$$

As there are $(m + 1)(n + 1)$ such sums, we only need to prove that each sum (3) is at most $M \cdot 2^{O(\log n+m+l)}$. In other words, we have to show that for all i, j there is a set $H \in \mathcal{A}^n_m$ with $x_1, \ldots, x_l \in A$ such that $\frac{\mathbf{m}(H)}{|H|}$ is greater than the sum (3) up to a factor of $2^{O(\log(n+m+l))}$.

To this end fix i and j. Since $\mathbf{m}(u) = 2^{-C(u)-O(\log C(u))}$, the sum (3) equals

$$\sum_{\substack{A \in \mathcal{A}^n_m \\ C(A)=i \\ \log|A|=j \\ x_1,\ldots,x_l \in A}} 2^{-C(A)-\log|A|+O(\log(n+m))} = \sum_{\substack{A \in \mathcal{A}^n_m \\ C(A)=i \\ \log|A|=j \\ x_1,\ldots,x_l \in A}} 2^{-i-j+O(\log(n+m))} \tag{4}$$

All the terms in the sum (4) coincide and thus the sum (4) is equal to $2^{-i-j+O(\log(n+m))}$ times the number of sets $A \in \mathcal{A}^n_m$ with $C(A) = i$, $\log|A| = j$, $x_1, \ldots, x_l \in A$. Let k denote the floor of the binary logarithm of that number.

Consider the bipartite graph whose left nodes are finite subsets from $\mathcal{A}^n$ of cardinality at most 2^j, right nodes are l-tuples of n-bit strings and a left node A is adjacent to a right node $\langle x_1, \ldots, x_l\rangle$ if all $x_1, \ldots, x_l$ are in A. The complexity of this graph is $O(\log(n + l + j))$ and the logarithm of the number of right nodes is nl. By Lemma 6 there is a set $H \in \mathcal{A}^n_m$ of

log-size j and complexity at most $i-k+O(\log(i+j+k+n+l))=i-k+O(\log(l+m+n))$ with $x_1,\dots,x_l\in A$. The fraction $\frac{\mathbf{m}(H)}{|H|}$ is equal to $2^{-(i-k)-j}$ up to a factor of $2^{O(\log(n+m+l))}$. Recall that the sum (4) equals to 2^k2^{-i-j} up to the same factor and thus we are done. ◀

▶ Remark. Consider the following case of Corollary 7: $\mathcal{A}$ is the family off all finite subsets, $l=1$. As was shown in Subsection 2.2 the sum $\sum_{A\ni x}\mathbf{m}(A)/|A|$, is equal to $\mathbf{m}(x)$ up to a *constant* factor.

By this reason, we expect that the accuracy in the corollary can be improved.

Proof of (b). Let y be some string that belongs to probability prediction d-neighborhood for x. According to (2), it implies that

$$\sum_{A\ni x,y}\frac{m(A)}{|A|}\geqslant\mathbf{m}(x)2^{-d-O(\log n)}=2^{-d-C(x)-O(\log n)}.$$

Now we will use Corollary 7 for $l=2$, $x_1=x$, $x_2=y$, $m=4n$ and the family of all sets as $\mathcal{A}$. By this corollary there is a set $A\ni x,y$ such that $\mathbf{m}(A)/|A|=2^{-d-C(x)-O(\log n)}$, so: $C(A)+\log|A|-C(x)\leqslant d+O(\log n)$, i. e. y belongs to the algorithmic prediction $d+O(\log n)$-neighborhood of x. ◀

2.4 Sets of Restricted Type

In some cases we know *a priori* what sets could be possible explanations, and are interested only in models from this class. To take this into account, we consider some family $\mathcal{A}$ of finite sets, and look for sets A in $\mathcal{A}$ that contain the data string x and are "good models" for x. This approach was used in [11]; it turns out that many results of algorithmic statistics can be extended to this case (though sometimes we get weaker versions with more complicated proofs).

In this section we show that Theorem 5 also has an analog for arbitrary decidable family $\mathcal{A}$. The family of all subsets of $\mathbb{B}^n$ that belong to $\mathcal{A}$ is denoted by $\mathcal{A}^n$.

First we consider the case when for each string x the set $\mathcal{A}$ contains the singleton $\{x\}$.

Let us define probability prediction neighborhood for a n-bit string x with respect to $\mathcal{A}$. Again we consider a two-stage process: first, some set of n-bit strings from $\mathcal{A}$ is chosen with probability $\mathbf{m}(A)$. Second, a random element in A is chosen uniformly. Again, we have to assume that we choose sets whose complexity is not greater than $4n$. A value $p_{\mathcal{A}}(y|x)$ is then defined as the conditional probability of $y\in A$ with the condition "the output of the two-stage process is x":

$$p_{\mathcal{A}}(y|x)=\frac{\sum_{A\ni x,y}\mathbf{m}(A)/|A|}{\sum_{A\ni x}\mathbf{m}(A)/|A|}\tag{5}$$

Here the sum is taken over all sets in $\mathcal{A}^n$ that have complexity at most $4n$.

Again as in Subsection 2.2 the denominator equals $\mathbf{m}(x)$ up to $O(1)$-factor (because $\{x\}\in\mathcal{A}$), so:

$$p_{\mathcal{A}}(y|x)=\frac{\sum_{A\ni x,y}\mathbf{m}(A)/|A|}{\mathbf{m}(x)}\tag{6}$$

up to $O(1)$-factor.

Then *$\mathcal{A}$-probabilistic prediction d-neighborhood* is defined naturally: a string y belongs to this neighborhood if $p_{\mathcal{A}}(y|x)\geqslant 2^{-d}$. The *$\mathcal{A}$-algorithmic prediction d-neighborhood* for x is

defined as follows: a string y belong to it if there is a set $A \ni x, y$ that belongs to $\mathcal{A}^n$ such that $\delta(x, A) \leqslant d$.

Now we are ready to state an analog of Theorem 5:

▶ **Theorem 8.** *Let $\mathcal{A}$ be a decidable family of binary strings containing all singletons. Then:*

(a) *For every n-bit string x and for every d the $\mathcal{A}$-algorithmic prediction d-neighborhood is contained in $\mathcal{A}$-probabilistic prediction $d + O(\log n)$-neighborhood.*

(b) *For every n-bit string x and for every d the $\mathcal{A}$-probabilistic prediction d-neighborhood of x is contained in $\mathcal{A}$-algorithmic prediction $d + O(\log n)$-neighborhood.*

Proof of (a). The proof is similar to the proof of Theorem 5 (a). Assume that a string y belongs to the algorithmic prediction d-neighborhood for x, i.e., there is a set $A \in \mathcal{A}^n$ containing x and y such that $C(A) + \log |A| \leqslant C(x) + d$. If $d > 3n$, then the statement is trivial. Indeed, there is a set $A' \in \mathcal{A}^n$ that contains x and y such that $\delta(x, A') \leqslant 3n$. To prove this, we can not set $A' = \mathbb{B}^n$ any more, as this set may not belong to $\mathcal{A}$. However we may let A' be the first set in $\mathcal{A}^n$, that contains x and y. The complexity of this set is not greater than $|x| + |y| \leqslant 2n$ and log-size is not greater than n. Thus $\delta(x, A') \leqslant 3n$. The rest of the proof is completely similar to the proof of Theorem 5 (a). ◀

Proof of (b). The proof is similar to the proof of Theorem 5 (b). ◀

Now we state and prove Theorem 8 in general case (for families $\mathcal{A}$ that may not contain all singletons). In the case $x \in \bigcup \mathcal{A}^n$, where $n = |x|$, the definition of $\mathcal{A}$-probability prediction neighborhood remains the same. Otherwise, if $x \notin \bigcup \mathcal{A}^n$, the string x can not appear in the two-stage process, so in this case we define $\mathcal{A}$-probability prediction d-neighborhood for x as the empty set for every d. Notice, that now we can not rewrite (5) as (6) because $\{x\}$ may not belongs to $\mathcal{A}$.

Now we define $\mathcal{A}$-algorithmic prediction neighborhood. There is a subtle point that should be taken into account: it may happen that there is no set $A \in \mathcal{A}$ containing x such that $\delta(x, A) \approx 0$. By this reason we include in the algorithmic prediction neighborhood of x the union of all sets A in $\mathcal{A}$, such that $\delta(x, A)$ is as small as it is possible:

▶ **Definition 9.** Let $x \in \mathbb{B}^n$ be a binary string, let d be some integer and let $\mathcal{A}$ be some family of sets. The union of all finite sets in $\mathcal{A}^n$ such that $x \in A$ and every $B \in \mathcal{A}^n$ that contains x satisfies the inequality: $\delta(x, A) \leqslant \delta(x, B) + d$ is called *$\mathcal{A}$-algorithmic prediction d-neighborhood of x*. (In other words, d-neighborhood includes all sets A whose $\delta(x, A)$ is at most d more than the minimum.)

▶ **Theorem 10.** *Let $\mathcal{A}$ be a decidable family of binary strings. Then:*

(a) *For every n-bit string x and for every d the $\mathcal{A}$-algorithmic prediction d-neighborhood is contained in $\mathcal{A}$-probabilistic prediction $d + O(\log n)$-neighborhood.*

(b) *For every n-bit string x and for every d the $\mathcal{A}$-probabilistic prediction d-neighborhood of x is contained in $\mathcal{A}$-algorithmic prediction $d + O(\log n)$-neighborhood.*

Notice that if $x \notin \bigcup \mathcal{A}^n$ then both algorithmic and prediction neighborhoods are empty and the statement is trivial. Therefore in the proof we will assume that this is not the case.

Proof of (a). The proof is completely similar to the proof of Theorem 8. ◀

Proof of (b). Let y be some strings that belongs to probability prediction d-neighborhood for x, that is,

$$\sum_{A \ni x,y} \frac{\mathbf{m}(A)}{|A|} \geqslant 2^{-d} \sum_{A \ni x} \frac{\mathbf{m}(A)}{|A|} \tag{7}$$

Let

$$A_x = \arg\max\{\mathbf{m}(A)/|A| \mid x \in A \in \mathcal{A}^n\}$$

and

$$A_{xy} = \arg\max\{\mathbf{m}(A)/|A| \mid x, y \in A \in \mathcal{A}^n\}.$$

Recall that $\frac{\mathbf{m}(A)}{|A|} = 2^{-C(A)-\log|A|}$ (up to a $2^{O(\log n)}$ factor) and by Corollary 7 the sums in both parts of the equality are equal to their largest terms (again up to $2^{O(\log n)}$ factor). Therefore,

$$2^{-C(A_{x,y})-\log|A_{x,y}|} \geqslant 2^{-d-O(\log n)} 2^{-C(A_x)-\log|A_x|},$$

which means that $\delta(x, A_{x,y}) \leqslant \delta(x, A_x) + d + O(\log n)$. Hence y belongs $\mathcal{A}$-algorithmic prediction $d + O(\log n)$-neighborhood of x. ◀

2.5 Prediction for Several Examples

Consider the following situation: we have not one but several strings $x_1, \ldots, x_l \in \mathbb{B}^n$ that are experimental data. We know that they were drawn independently with respect to the uniform probability distribution in some unknown set A. We want to explain these observation data, i. e. to find an appropriate set A. Again we measure the quality of explanations by two parameters: $C(A)$ and $\log|A|$.

In this section we will extend previous results to this scenario. Again we assume that we know a priori which sets could be possible explanations. So, we consider only sets from a decidable family of sets $\mathcal{A}$.

Let $\overrightarrow{x}$ denote the tuple $x_1, \ldots, x_l$. Let $A \subset \mathbb{B}^n$ be a set that contains all strings from $\overrightarrow{x}$. Then we can restore $\overrightarrow{x}$ from A and indexes of strings from $\overrightarrow{x}$ in A and hence we have :

$$C(\overrightarrow{x}) \leqslant C(A) + l\log|A| + O(\log n).$$

Therefore it is natural to define the *optimality deficiency* of $A \ni \overrightarrow{x}$ by the formula

$$\delta(\overrightarrow{x}, A) := C(A) + l\log|A| - C(\overrightarrow{x}).$$

The definitions of the $\mathcal{A}$-algorithmic prediction d-neighborhood of the tuple $\overrightarrow{x}$ is obtained from Definition 9 by changing x to $\overrightarrow{x}$.

In a similar way we modify the definition of the $\mathcal{A}$-probabilistic prediction neighborhood. Again we consider a two-stage process: first, a set of n-bit strings from $\mathcal{A}$ is chosen with probability $\mathbf{m}(A)$. Second, l random elements in A are chosen uniformly and independently on each other. Again, by technical reason, we assume, that we consider only sets whose complexity is not greater then $(l+3)n$. The value $p_{\mathcal{A}}(y|\overrightarrow{x})$ is defined as the conditional probability of $y \in A$ under the condition [the output of this two-stage process is equal to $\overrightarrow{x}$]:

$$p_{\mathcal{A}}(y|\overrightarrow{x}) = \frac{\sum_{A \ni \overrightarrow{x},y} \mathbf{m}(A)/|A|^l}{\sum_{A \ni \overrightarrow{x}} \mathbf{m}(A)/|A|^l}.$$

Here both sums are taken over all sets $A \in \mathcal{A}^n$ that have complexity at most $n(l+3)$. (If no such set contains x then $p_{\mathcal{A}}(y|\overrightarrow{x}) = 0$.) By definition, a string y belongs to $\mathcal{A}$-probabilistic prediction d-neighborhood for $\overrightarrow{x}$ if $p_{\mathcal{A}}(y|\overrightarrow{x}) \geqslant 2^{-d}$.

Now we are ready to state an analog of Theorem 10:

▶ **Theorem 11.** *Let $\mathcal{A}$ be a decidable family of binary strings. Then:*

(a) *For every l n-bit strings $\overrightarrow{x}$ and for every d the $\mathcal{A}$-algorithmic prediction d-neighborhood is contained in $\mathcal{A}$-probabilistic prediction $d + O(\log(n+l))$-neighborhood of $\overrightarrow{x}$.*

(b) *For every l n-bit strings $\overrightarrow{x}$ and for every d the $\mathcal{A}$-probabilistic prediction d-neighborhood of $\overrightarrow{x}$ is contained in $\mathcal{A}$-algorithmic prediction $d + O(\log(n+l))$-neighborhood of $\overrightarrow{x}$.*

Proof. The proof is entirely similar to the proof of Theorem 10, but now Corollary 7 is applied for l and $l+1$ strings so the accuracy becomes $O(\log(n+l))$. ◀

3 Non-uniform Probability Distributions

We have considered so far only uniform probability distributions as statistical hypotheses. The paper [10, Appendix II] justifies such a restriction: it was observed there that for every data string x and for probability distribution P there is a finite set $A \ni x$ that is not worse than P as an explanation for x (with logarithmic accuracy). However, if the data consists of more than one string, then this is not the case. Now, we will explain this in more details.

The quality of a probability distribution P as an explanation for the data $x_1, \ldots, x_l$ is measured be the following two parameters:

- the complexity $C(P)$ of the distribution P,
- $-\log(P(x_1)\ldots P(x_l))$ (the smaller this parameter is the larger is the likelihood to get the tuple $\overrightarrow{x}$ by independently drawing l strings with respect to P).

We consider only distributions over finite sets such that the probability of every outcome is a rational number. The complexity of such a distribution is defined as the complexity of the set of all pairs $\langle y, P(y)\rangle$ ordered lexicographically.

If P is a uniform distribution over a finite set A then the first parameter becomes $C(A)$ and the second one becomes $-l\log|A|$. If $l = 1$ then for every pair x, P there is a finite set $A \ni x$ such that both $C(A), \log|A|$ are at most $C(P), -\log P(x)$ with the accuracy $O(\log|x|)$. Indeed, let $A = \mathbb{B}^n$ if $P(x) \geqslant 2^{-n}$ and

$$A = \{x \in \mathbb{B}^n \mid P(x) \geqslant 2^{-i}\}$$

if $2^{-i} \leqslant P(x) < 2^{-i+1} \leqslant 2^{-n}$. In both cases we have $C(A) \leqslant C(P) + O(\log n)$ and $\log|A| \leqslant -\log P(x) + 1$.

For $l = 2$ this is not the case:

▶ **Example 12.** Let x_1 be a random string of length $2n$ and $x_2 = 00\ldots0y$ be a string of length $2n$ where y is a random string of length n independent of x_1 (that is, $C(x_1, x_2) = 3n + O(1)$). A plausible explanation of such data is the following: the strings x_1, x_2 were drawn independently with the respect the distribution P where half of the probability is uniformly distributed over all strings of length $2n$ and the remaining half is uniformly distributed over all strings of length $2n$ starting with n zeros. The complexity of this distribution P is negligible ($O(\log n)$) and the second parameter $-\log(P(x_1)P(x_2))$ is about $3n$. On the other hand there is no simple set A containing both strings x_1, x_2 with $2\log|A|$ being close to $3n$. Indeed, for every set A containing x_1 we have $C(A) + \log|A| \geqslant 3n - O(\log n)$ and hence $2\log|A| \geqslant 6n - 2C(A) - O(\log n) \gg 3n$ (the last inequality holds provided $C(A)$ is small).

Therefore we will not restrict the class of statistical hypotheses to uniform distributions. We will show that the main result of [11] (Theorem 16 below) translates to the case of several strings, i.e., to the case $l > 1$ (Theorem 17 below).

3.1 The Profile of a Tuple $x_1, \ldots, x_l$

Fix $x_1, \ldots, x_l \in \mathbb{B}^n$. As above, we will denote by $\overrightarrow{x}$ the tuple $x_1, \ldots, x_l$. The optimality deficiency is defined by the formula

$$\delta(\overrightarrow{x}, P) = C(P) - \log(P(x_1) \ldots P(x_l)) - C(\overrightarrow{x}).$$

This value is non-negative up to $O(\log(n + l))$, since given P and l we can describe the tuple $\overrightarrow{x}$ in $-\log(P(x_1) \ldots P(x_l)) + O(1)$ bits, using the Shannon-Fano code.

▶ **Definition 13.** The profile $P_{\overrightarrow{x}}$ of the tuple $\overrightarrow{x}$ is defined as the set of all pairs $\langle a, b\rangle$ of naturals such that there is a probability distribution P of Kolmogorov complexity at most a with $\delta(\overrightarrow{x}, P) \leqslant b$.

Loosely speaking, a tuple of strings $\overrightarrow{x}$ is called *stochastic* if there is a simple distribution P such that $\delta(\overrightarrow{x}, P) \approx 0$. In other words, if $(a, b) \in P_{\overrightarrow{x}}$ for $a, b \approx 0$. Otherwise it is called *non-stochastic*. In one-dimensional case non-stochastic objects were studied, for example, in [7], [11]. However, in the one-dimensional case we can not present explicitly a non-stochastic object. In the two-dimensional case the situation is quite different: let x_1 be a random string of length n and let $x_2 = x_1$. For such pair x_1, x_2 there is no simple distribution P with small $\delta(\langle x_1, x_2\rangle, P)$. Indeed, for any probability distribution P we have $C(P) - \log P(x_i) \geqslant C(x_i) = n$ for $i = 1, 2$ (with accuracy $O(\log n)$). Adding these inequalities we get

$$2C(P) - \log(P(x_1)P(x_2)) \geqslant 2n\,.$$

Hence $\delta(\langle x_1, x_2\rangle, P) \geqslant 2n - C(P) - C(x_1, x_2) = n - C(P)$, which is very large provided $C(P) \ll n$.

In general, if strings x_1 and x_2 have much *common information* (i. e. $C(x_1, x_2) \ll C(x_1) + C(x_2)$), then the pair $\langle x_1, x_2\rangle$ is non-stochastic. There is also a non-explicit example of a non-stochastic pair of strings: consider any pair whose first term is non-stochastic. There is no good explanation for the first term, hence there is no good explanation for the whole pair.

The first example suggests the following question: is the profile of the pair of strings x_1, x_2 determined by $C(x_1), C(x_2), C(x_1, x_2)$, P_{x_1}, P_{x_2} and $P_{[x_1,x_2]}$? Here $[x_1, x_2]$ denotes the concatenation of strings x_1 and x_2. Notice that $P_{[x_1,x_2]}$ denotes the 1-dimensional profile of the string $[x_1, x_2]$ and is not to be confused with P_{x_1,x_2}, which is the 2-dimensional profile of the pair of strings x_1, x_2. The following theorem is the main result of Section 3. It provides a negative answer to this question.

▶ **Theorem 14.** *For every n there are strings x_1, x_2, y_1 and y_2 of length $2n$ such that:*

1. *The sets P_{x_1} and P_{y_1}, P_{x_2} and P_{y_2}, $P_{[x_1,x_2]}$ and $P_{[y_1,y_2]}$ are at most $O(\log n)$ apart.*
2. *$C(x_1) = C(y_1) + O(\log n)$, $C(x_2) = C(y_2) + O(\log n)$, $C(x_1, x_2) = C(y_1, y_2) + O(\log n)$.*
3. *However the distance between P_{x_1,x_2} and P_{y_1,y_2} is greater than $0.5n - O(\log n)$. (We say that the distance between two sets R and Q is at most ε if R is contained in ε-neighborhood, with respect to L_∞-norm, of Q, and vice versa.)*

Sketch of proof. Our example is borrowed from [1], where there are several examples of *stochastic* pairs of strings with non-extractable common information.

Consider a finite field $\mathbb{F}$ of cardinality 2^n and a plane (two-dimensional vector space) over $\mathbb{F}$. Let y_1 be a random line on this plane, and y_2 be a random point on this line. Then

$$C(y_1) = 2n, C(y_2) = 2n, C(y_1, y_2) = 3n$$

(everything with logarithmic accuracy). These strings y_1, y_2 have about n bits of common information. On the other hand [1, Theorem 8] states that this common information is non-extractable. By this reason $(1.5n, 0.5n - O(\log n)) \notin P_{y_1,y_2}$.

It is easy to construct another pair of strings x_1, x_2 that has the same properties except that the pair $(n + O(1), O(1))$ is inside P_{x_1,x_2}. To this end let x_1, x_2 be random strings of length $2n$ that share first n bits: $x_1 = x^*x_1^*$, $x_2 = x^*x_2^*$ and $C(x^*x_1^*x_2^*) = 3n + O(1)$. ◀

3.2 Randomness Deficiency

In this subsection we introduce multi-dimensional randomness deficiency and show that the main result of [11] relating 1-dimensional randomness deficiency and optimality deficiency translates to any number of strings.

The 1-dimensional randomness deficiency of a string x in a finite set A was defined by Kolmogorov as $d(x|A) = \log|A| - C(x|A)$. It is always non-negative (with $O(\log|x|)$ accuracy), as we can find x from A and the index of x in A. For most elements x in any set A the randomness deficiency of x in A is negligible. More specifically, the fraction of x in A with randomness deficiency greater than β is less than $2^{-\beta}$. The randomness deficiency measures how non-typical looks x in A.

▶ **Definition 15.** The set of all pairs (a, b) such that there is a set $A \ni x$ of complexity at most a and $d(x|A) \leqslant b$ is called the *stochasticity profile* of x and is denoted by Q_x

To distinguish profiles P_x and Q_x we will call P_x the *optimality profile* in the sequel. Surprisingly, the sets P_x and Q_x almost coincide:

▶ **Theorem 16** ([11]). *For every string x of length n the distance between P_x and Q_x is at most $O(\log n)$.*

The multi-dimensional randomness deficiency is defined in the following way. For a tuple of strings $\overrightarrow{x} = x_1, \ldots, x_l$ and a distribution P let

$$d(\overrightarrow{x}|P) = -\log(P(x_1)\ldots P(x_l)) - C(x_1, \ldots, x_l|P)\,.$$

If $l = 1$ and P is a uniform distribution in a finite set then this definition is equivalently to the one-dimensional case. The randomness deficiency measures how implausible is to get $x_1, \ldots, x_l$ as a result of l independent draws from A. The set off all pairs (a, b) such that there is a distribution P of complexity at most a and $d(\overrightarrow{x}|P) \leqslant b$ is called the *l-dimensional stochasticity profile of $\overrightarrow{x}$* and is denoted by $Q_{\overrightarrow{x}}$.

It turns out that Theorem 16 translates to multi-dimensional case:

▶ **Theorem 17.** *For every tuple $\overrightarrow{x} = x_1, \ldots, x_l$ of strings of length n the distance between sets $P_{\overrightarrow{x}}$ and $Q_{\overrightarrow{x}}$ is at most $O(\log(n + l))$.*

▶ Remark. Theorem 17 is basically an analog of Theorem 16 for a restricted class of distributions, namely, for product distributions Q on l-tuples, i.e., distributions of the form $Q(x_1, \ldots, x_l) = P(x_1)\cdots P(x_l)$. A natural question is whether Theorem 16 can be generalized to any decidable class of distributions. This is indeed the case and the proof is very similar to the proof of Theorem 17.

An Open Question

Can we improve the accuracy in Corollary 7 from $2^{O(\log(n+m+l))}$ to $2^{O(\log(n+l))}$?

Acknowledgments. I would like to thank Alexander Shen and Nikolay Vereshchagin for useful discussions, advice and remarks.

References

1 A. Chernov, An. Muchnik, A. Romashchenko, A. Shen, and N. Vereshchagin. Upper semilattice of binary strings with the relation "x is simple conditional to y". Theoretical Computer Science 271 (2002) 69–95.

2 P. Gács, J. Tromp, P.M.B. Vitányi. Algorithmic statistics, *IEEE Trans. Inform. Th.*, 47:6 (2001), 2443–2463.

3 A.N. Kolmogorov, Talk at the Information Theory Symposium in Tallinn, Estonia, 1974.

4 Moshe Koppel *Complexity, depth and sophistication* Complex Systems 1, pp. 87–91

5 Li M., Vitányi P., *An Introduction to Kolmogorov complexity and its applications*, 3rd ed., Springer, 2008 (1 ed., 1993; 2 ed., 1997), xxiii+790 pp. ISBN 978-0-387-49820-1.

6 A. Shen *Around Kolmogorov complexity: basic notions and results* `http://arxiv.org/abs/1504.04955`

7 A. Shen *The concept of* (α, β)*-stochasticity in the Kolmogorov sense, and its properties. Soviet Mathematics Doklady, 271(1):295–299, 1983*

8 A. Shen, V. Uspensky, N. Vereshchagin *Kolmogorov complexity and algorithmic randomness*. MCCME, 2013 (Russian). English translation: `http://www.lirmm.fr/~ashen/kolmbook-eng.pdf`

9 N. Vereshchagin, A. Shen *Algorithmic statistics revisited* http://arxiv.org/abs/1504.04950

10 N. Vereshchagin and P. Vitányi *"Kolmogorov's Structure Functions with an Application to the Foundations of Model Selection" . IEEE Transactions on Information Theory 50:12 (2004) 3265-3290. Preliminary version: Proc. 47th IEEE Symp. Found. Comput. Sci., 2002, 751–760.*

11 N.K. Vereshchagin, P.M.B. Vitányi *Rate Distortion a nd Denoising of Individual Data Using Kolmogorov Complexity* IEEE Transactions on Information Theory,56:7 (2010), 3438–3454

Polynomial Kernels for Deletion to Classes of Acyclic Digraphs*

Matthias Mnich[1] and Erik Jan van Leeuwen[2]

1 **Universität Bonn, Bonn, Germany**
mmnich@uni-bonn.de
2 **Max-Planck-Institut für Informatik, Saarbrücken, Germany**
erikjan@mpi-inf.mpg.de

Abstract

We consider the problem to find a set X of vertices (or arcs) with $|X| \leq k$ in a given digraph G such that $D = G - X$ is an acyclic digraph. In its generality, this is Directed Feedback Vertex Set or Directed Feedback Arc Set respectively. The existence of a polynomial kernel for these problems is a notorious open problem in the field of kernelization, and little progress has been made.

In this paper, we consider both deletion problems with an additional restriction on D, namely that D must be an out-forest, an out-tree, or a (directed) pumpkin. Our main results show that for each of these three restrictions the vertex deletion problem remains NP-hard, but we can obtain a kernel with $k^{O(1)}$ vertices on general digraphs G. We also show that, in contrast to the vertex deletion problem, the arc deletion problem with each of the above restrictions can be solved in polynomial time.

1998 ACM Subject Classification F2.2 Nonnumerical Algorithms and Problems

Keywords and phrases directed feedback vertex/arc set, parameterized algorithms, kernels

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.55

1 Introduction

In this paper, we study the problem of removing a (small) subset of vertices X from a graph G such that the resulting graph $G - X$ is acyclic. On undirected graphs, this translates immediately to the property that $G - X$ is a forest or (if we insist that $G - X$ is connected) a tree. The problem to decide whether a given undirected graph G has a set $X \subseteq V(G)$ of size at most a given integer k such that $G - X$ is a forest or a tree is known as Feedback Vertex Set and Tree Deletion Set respectively.

Over the past years, we have gotten to understand the complexity of Feedback Vertex Set and Tree Deletion Set quite well. Both problems are NP-hard [14, 20]. It is long known that the minimization version of Feedback Vertex Set admits a polynomial-time 2-approximation algorithm [2] and that Feedback Vertex Set admits a polynomial kernel parameterized by k [19] (see e.g. [8, 11] for background on kernelization). The minimization version of Tree Deletion Set, in contrast, cannot be polynomial-time approximated within a factor $O(n^{1-\epsilon})$ for any $\epsilon > 0$ [20], unless P = NP. However, Tree Deletion Set was recently shown to admit a polynomial kernel (when parameterized by k) [13].

The usual way to generalize Feedback Vertex Set and Tree Deletion Set to digraphs is to insist that the resulting digraph has no directed cycle. Indeed, the problem to

* Research of M.M. was supported by ERC Starting Grant 306465 (BeyondWorstCase).

© Matthias Mnich and Erik Jan van Leeuwen;
licensed under Creative Commons License CC-BY

decide whether a given digraph G has a set $X \subseteq V(G)$ of size at most a given integer k such that $G - X$ is a (connected) acyclic digraph is known as Directed Feedback Vertex Set (Connected DAG Vertex Deletion Set). In contrast to their undirected counterparts, the complexity situations for Directed Feedback Vertex Set and Connected DAG Vertex Deletion Set are very much unclear.

It is known that Directed Feedback Vertex Set is NP-hard [14], even on tournaments [18]. Connected DAG Vertex Deletion Set is NP-hard and cannot be polynomial-time approximated within a factor $O(n^{1-\epsilon})$ for any $\epsilon > 0$ [20], but we are not aware of any results on the parameterized complexity of this problem. Directed Feedback Vertex Set is polynomial-time approximable within a factor of $O(\log |V(G)| \log \log |V(G)|)$ on general digraphs [9, 17], but it is open whether this is best possible. Directed Feedback Vertex Set has a kernel of exponential size $k^{O(k)}$, as the problem was shown fixed-parameter tractable by Chen et al. [4], but it is unknown whether a polynomial kernel exists. In fact, this question remains open despite being posed several times [4, 10, 8, 6, 5].

There is limited insight into whether Directed Feedback Vertex Set could admit a polynomial kernel. Abu-Khzam [1] (see also Dom et al. [7]) showed that Directed Feedback Vertex Set admits a polynomial kernel if the given digraph is a (bipartite) tournament and Bang-Jensen et al. [3] recently extended this to generalizations of tournaments. We are not aware of polynomial kernels for Directed Feedback Vertex Set on other restricted classes of digraphs. This suggests to explore other roads towards an answer to the open question of a polynomial kernel for Directed Feedback Vertex Set.

Our Contributions

We study a different translation of Feedback Vertex Set and Tree Deletion Set to digraphs. Instead of transferring the property that the resulting graph should be acyclic to digraphs, we transfer the property that the resulting graph should be a forest or tree. To this end, we consider the notion of an *out-tree*, which is a digraph where each vertex has in-degree at most 1 and the underlying (undirected) graph is a tree. An *out-forest* is a disjoint union of out-trees. This leads to the following parameterized problems:

Out-Forest/Out-Tree Vertex Deletion Set
Input: A digraph G and an integer k.
Question: Is there a set $X \subseteq V(G)$ with $|X| \leq k$ s.t. $G - X$ is an out-forest/out-tree?

Note that these problems can also be viewed as a restricted version of Directed Feedback Vertex Set and Connected DAG Vertex Deletion Set. Here, instead of restricting the input graph G as Abu-Khzam [1] and Dom et al. [7] did when they considered tournaments, we consider general digraphs G as input but restrict what kind of acyclic digraph the resulting digraph $G - X$ should be.

Thinking further in this direction, we consider another restriction on the resulting digraph, namely that it should be a pumpkin. A digraph is a *pumpkin* if it consists of a source vertex s and a sink vertex t ($s \neq t$), together with a collection of internally vertex-disjoint induced directed paths from s to t. Note that the underlying graph of a pumpkin is not acyclic, in contrast to out-forests and out-trees. This leads to the following parameterized problem:

Pumpkin Vertex Deletion Set
Input: A digraph G and an integer k.
Question: Is there a set $X \subseteq V(G)$ with $|X| \leq k$ s.t. $G - X$ is a pumpkin?

We consider all three problems on general digraphs, and observe that each is NP-hard, even on acyclic digraphs. More importantly, we show that all three problems admit a polynomial kernel.

▶ **Theorem 1.** Out-Tree Vertex Deletion Set *is* NP*-hard, even on acyclic digraphs, but admits a kernel with* $O(k^3)$ *vertices.*

▶ **Theorem 2.** Out-Forest Vertex Deletion Set *is* NP*-hard, even on acyclic digraphs, but admits a kernel with* $O(k^3)$ *vertices.*

▶ **Theorem 3.** Pumpkin Vertex Deletion Set *is* NP*-hard, even on acyclic digraphs, but admits a kernel with* $O(k^{18})$ *vertices.*

The polynomial kernel for Out-Tree Vertex Deletion Set, presented in Sec. 2, relies on a large set of reduction rules that heavily exploit that vertices of out-trees have in-degree at most 1. Although this 'forbidden structure' seems to lend itself naturally to a characterization by forbidden induced subgraphs that can be attacked through a standard approach using the Sunflower Lemma (cf. [12, Lemma 3.2]), the demand that the resulting digraph be *connected* means that substantially different methods must be employed to obtain meaningful structure.

After applying our set of reduction rules, the underlying graph of the resulting digraph G' contains a small feedback vertex set F. It then remains to show that the forest $G' - F$ has bounded size by analyzing the interaction of $G' - F$ with F. In particular, we argue that the leaves of $G' - F$ can be split into four types, and bound the number of leaves of each type. In the analysis of one of the types (the fourth), we adapt some of the rules of the polynomial kernel for Tree Deletion Set by Giannopoulou et al. [13] to the directed case; the analysis for the other cases is new and specific to Out-Tree Vertex Deletion Set.

The kernel for Out-Forest Vertex Deletion Set follows the same lines, but requires an additional reduction rule, presented in Sec. 3. We believe that the Sunflower Lemma could yield an alternative road to a polynomial kernel for Out-Forest Vertex Deletion Set (as connectedness is no longer an issue), but we chose to instead present our simple extension of the kernel for Out-Tree Vertex Deletion Set.

The polynomial kernel for Pumpkin Vertex Deletion Set, presented in Sect. 4, uses completely different methods. We first show that there are only $\text{poly}(k)$ candidates for the source and sink of the pumpkin. Therefore, we can split the instance into $\text{poly}(k)$ new instances with an annotated source and sink, each of which we subsequently kernelize. The resulting kernelized instances can be seen as a Turing kernel of the problem. Instead of being satisfied with this, we show that we can modify and combine these kernelized instances into a single instance of Pumpkin Vertex Deletion Set, which forms the final kernel.

The NP-hardness results and missing proofs are deferred to the full version of this paper.

Edge and Arc Deletion Problems

Instead of deleting vertices to get an acyclic graph, we also consider the problem of deleting edges or arcs from a given (di)graph to obtain an acyclic (di)graph. On undirected graphs, the problem to delete edges to obtain a forest or tree can easily be shown to be polynomial-time solvable by reducing to finding a spanning forest/tree. On digraphs, however, the complexity situation is quite different. In fact, it can be readily shown that the problem of deleting arcs from a digraph to remove all cycles must have the same complexity as Directed Feedback Vertex Set [9]. Therefore, one wonders how this affects the three problems of this paper.

The arc-deletion versions of our problems are defined as follows. Given a subset B of the arcs of a digraph G, the digraph *induced* by B is the graph with vertex set equal to the set of endpoints of the arcs in B and arc set equal to B.

Out-Forest/Out-Tree/Pumpkin Arc Deletion Set	
Input:	A digraph G and an integer k.
Question:	Is there a set $X \subseteq A(G)$ with $\|X\| \leq k$ s.t. the arcs of $A(G) - X$ induce an out-forest/out-tree/pumpkin?

▶ **Theorem 4.** Out-Forest Arc Deletion Set, Out-Tree Arc Deletion Set, *and* Pumpkin Arc Deletion Set *can be solved in polynomial time.*

A shortened proof of this theorem is in Sect. 5; full proofs are deferred to the full version.

Throughout this paper, we consider digraphs G with vertex set $V(G)$ and arc set $A(G)$. For a digraph G, we denote its underlying undirected graph by $\langle G \rangle$. The in-degree and the out-degree of a vertex $v \in V(G)$ is denoted $d^-(v)$ resp. $d^+(v)$.

For a digraph G and distinct vertices $u, v \in V(G)$, call $P = [w_0, \ldots, w_\ell]$ an *induced directed u,v-path* (of *length* ℓ) if $u = w_0$, $v = w_\ell$, (w_i, w_{i+1}) are arcs of G for $i = 0, \ldots, \ell - 1$ and $d^-(w_i) = d^+(w_i) = 1$ for $i = 1, \ldots, \ell - 1$.

2 Polynomial Kernel for Out-Tree Vertex Deletion

For a digraph G, call a set $S \subseteq V(G)$ an *out-tree vertex deletion set* if $G - S$ is an out-tree. To obtain a polynomial kernel for Out-Tree Vertex Deletion Set parameterized by solution size k, we first generalize the problem to its vertex-weighted variant, defined as:

Weighted Out-Tree Vertex Deletion Set	
Input:	A digraph G with weight function $w : V(G) \to \mathbb{N}$; an integer k.
Question:	Is there a set $S \subseteq V(G)$ of weight $w(S) = \sum_{v \in S} w(v) \leq k$ so that $G - S$ is an out-tree?

Allowing vertices to carry weights will allow for more flexible reduction rules. Using four relatively standard reduction rules (described in the full version), we can impose the following structure on instances of Weighted Out-Tree Vertex Deletion Set:

▶ **Lemma 5.** *Given an instance (G, w, k) of* Weighted Out-Tree Vertex Deletion Set, *in polynomial time we can construct an instance (G', w', k') such that:*

1. $d^-(v) \leq k + 1$ *for each* $v \in V(G')$;
2. $w'(v) \leq k + 1$ *for each* $v \in V(G')$;
3. *no vertex* $v \in V(G')$ *has* $d^-(v) = 1$ *and* $d^+(v) = 0$;
4. *every induced directed path of* G' *has length at most* 4;
5. (G, w) *has an out-tree vertex deletion set of weight at most* k *if and only if* (G', w') *has an out-tree vertex deletion set of weight at most* k'

We now define several novel reduction rules that are specific to Weighted Out-Tree Vertex Deletion Set. A *collision* in G is an ordered triple (u, m, v) of distinct vertices u, m, v such that $(u, m), (v, m) \in A(G)$. A collision only demands that $(u, m), (v, m) \in A(G)$; it does not specify anything about other arcs between u, m, v.

▶ **Lemma 6** (Collision Star Rule). *Let $(u_1, m_1, v), \ldots, (u_{k+1}, m_{k+1}, v)$ be collisions that pairwise intersect only in v. Let $G' = G - v$, let $k' = k - w(v)$ and $w' = w_{|V(G')}$. Then (G, w) has an out-tree vertex deletion set of weight at most k if and only if (G', w') has an out-tree vertex deletion set of weight at most k'.*

Proof. Let S be an out-tree vertex deletion set of (G, w) of weight at most k. We argue that $v \in S$; this then implies immediately that $S \setminus \{v\}$ is an out-tree vertex deletion set of (G', w') with weight at most $k' = k - w(v)$. To this end, suppose that $v \notin S$; then from each collision (u_i, m_i, v), at least one of u_i, m_i belongs to S, since S must intersect all collisions (out-trees are free of collisions). As each of u_i, m_i has weight at least 1 and the pairs $(u_1, m_1), \ldots, (u_{k+1}, m_{k+1})$ are pairwise vertex-disjoint, this contradicts that $w(S) \leq k$; hence, $v \in S$.

Conversely, let S' be an out-tree vertex deletion set of (G', w') of weight at most k'. As out-tree vertex deletion sets are closed under taking supersets, $S = S' \cup \{v\}$ is an out-tree vertex deletion set of (G, w). Further, the weight of S is equal to $w(S) = w'(S') + w(v) \leq k' + w(v) = k$.

We can implement the rule in polynomial time by considering each v in turn and then running a maximum matching algorithm. For a vertex v, let N_v^+ denote its set of out-neighbors. Let N_v^* denote the set of in-neighbors of the vertices in N_v^+, where we ensure that v is not placed in N_v^*. Now consider an auxiliary graph H_v on $N_v^+ \cup N_v^*$, were $u \in N_v^+$ and $w \in N_v^*$ are adjacent if and only if there is an arc from w to u in G. Then any set of collisions that pairwise intersect only in v (as in the lemma statement) induces a matching in H_v, and conversely, any matching in H_v induces a set of collisions that pairwise intersect only in v (as in the lemma statement). Hence, it suffices to find, for each $v \in V(G)$ a matching of size at least k in H_v, which can be done in polynomial time. ◀

Note again that the rule only specifies that the arcs (u_i, m_i) and (m_i, v) for $i = 1, \ldots, k+1$ should belong to $A(G)$; it does not specify anything about other arcs between these vertices.

▶ **Lemma 7** (Source Rule). *If G contains at least $k+2$ vertices of in-degree 0, then (G, w) does not contain an out-tree vertex deletion set of weight at most k.*

We apply the above rules exhaustively, before continuing to the next rule.

▶ **Lemma 8** (Feedback Vertex Set Rule). *If a 2-approximate minimum size undirected feedback vertex set of $\langle G \rangle$ has size more than $2k$, then (G, w) does not admit an out-tree vertex deletion set of weight at most k.*

This rule is correct, because $\langle G - S \rangle$ is acyclic, where S is an out-tree vertex deletion set of (G, w). We thus assume that $\langle G \rangle$ has a feedback vertex set F of size at most $2k$.

We next argue that if $G - F$ has many vertex-disjoint collisions, then G does not admit an out-tree vertex deletion set of low weight, because an out-tree does not contain any collisions. Observe that finding the maximum number of vertex-disjoint collisions in a digraph is in general an NP-hard problem, because of a straightforward reduction from the P_2-matching problem, which is known to be NP-hard [15]. However, here we only need to solve this task in $G - F$, which is a forest. We are then able to employ a greedy algorithm that 'cuts away' collisions in a bottom-up fashion and finds a largest set of vertex-disjoint collisions in $G - F$ in polynomial time.

▶ **Lemma 9** (Disjoint Collisions Rule). *If $G - F$ contains more than k vertex-disjoint collisions, then (G, w) does not admit an out-tree vertex deletion set of weight at most k.*

Assume now that Lemma 9 has been applied, and that F (as before) is a feedback vertex set of $\langle G \rangle$. Let $T_1, \ldots, T_c$ denote the set of connected components of $G - F$; then each underlying undirected component $\langle T_i \rangle$ is a tree. Let $\mathcal{L}$ denote the set of leaves of $\langle T_1 \rangle, \ldots, \langle T_c \rangle$. Our strategy to prove a polynomial kernel will be to first bound $|\mathcal{L}|$, and therefrom bound both c and $\sum_{i=1}^{c} |V(T_i)|$.

Bounding the Number of Leaves

We consider four types of leaves. Throughout, we assume that a leaf of type i is not of type $i-1,\ldots,1$.

Type 1: Leaves with an arc to a vertex of F.

By Lemma 5(1), $\sum_{v\in F} d^-(v) \leq |F|(k+1) \leq 2k(k+1)$. Hence, the number of leaves with an arc to a vertex of F is at most $2k(k+1)$.

Type 2: Leaves without any arc to or from a vertex of F.

By Lemma 5(3), such a leaf must be a source of G. We can bound the number of leaves of type 2 by $k+1$, since Lemma 7 does not apply.

Type 3: Leaves with an arc from a vertex of F whose unique incident arc in $G-F$ is an out-arc.

Let $v_1,\ldots,v_\ell$ denote those leaves. An *F-disjoint path* is a directed path P in G between two vertices $u,v\in V(G)$ such that no vertex of P belongs to F; in particular, $u,v\notin F$. Call a vertex $m\notin F$ a *mixer for i,j* (where i,j are distinct) if there exist F-disjoint paths from v_i to m and from v_j to m; we also say that i,j get *mixed* at m. If $m\notin F$ is a mixer for some i,j, then we simply call m a *mixer*. Observe that for any $m\notin F$, there is an F-disjoint path in G to some v_i only if v_i and m are part of the same tree of $\langle G\rangle - F$; therefore, if such a path exists, then it is unique.

We now construct a set M of mixers, as follows. Initialize $M=\emptyset$ and $I=\{1,\ldots,\ell\}$. Iteratively find a mixer m for some $\{i,j\}\subseteq I$ such that for any $\{i',j'\}\subseteq I$ that get mixed at m, the F-disjoint path from v_i to m or F-disjoint path from v_j to m is free of mixers for i,j', and i',j, and i',j' as internal vertex. Then add m to M, and remove all indices from I that get mixed at m. Denote by I' the set of indices in I at the end of this procedure.

We can show that the set of mixers induces a set of vertex-disjoint collisions of size $|M|$ by using that two unique indices i,j get mixed at each $m\in M$; the F-disjoint paths from v_i and v_j to m then ensure that m has two unique in-neighbors. Hence, the following lemma follows from Lemma 9:

▶ **Lemma 10.** *For the constructed set M of mixers, $|M|\leq k$.*

Consider any index $i\in\{1,\ldots,\ell\}\setminus I'$, and let m denote the mixer that was added to M at the step that i was removed from I. Let P_i be an F-disjoint path from v_i to m that is guaranteed by m being a mixer. It follows from the construction of M that $\{P_i \mid i\in\{1,\ldots,\ell\}\setminus I'\}$ forms a collection of F-disjoint paths that are pairwise disjoint except possibly for their end vertices. By Lemma 5(1), $d^-(m)\leq k+1$ for each $m\in M$. Hence, $\ell - |I'| \leq |M|(k+1)\leq k(k+1)$.

To bound $|I'|$, we find a set of inclusion-wise maximal directed paths starting in the vertices in $\{v_i \mid i\in I'\}$. The ends (not equal to v_i) of these paths yield a set of collisions, which are almost disjoint. Then Lemmas 6 and 9 combined imply:

▶ **Lemma 11.** $|I'|\leq 3k^2+2k$.

Adding up, we have that $\ell\leq 3k^2+2k+k(k+1)=4k^2+3k$.

Type 4: Leaves with an arc from a vertex of F whose arc in $G-F$ is an in-arc.

To bound the number of leaves of Type 4, we need an extra reduction rule. This rule is similar to one applied by Giannopoulou et al. [13] in a different context. The general idea is

to translate the constraints on leaves of Type 4 into a set of linear equations, then kernelize this set of equations, and use that kernel to reduce the number of leaves of Type 4. Since this works in the same way as in the paper by Giannopoulou et al. [13], we defer the details to the full version.

▶ **Lemma 12.** *The number of leaves of Type 4 is at most* $2k(k+1)^2 + k + 1$.

After analyzing all types of leaves, we can conclude that $|\mathcal{L}| \leq 2k^3 + 10k^2 + 9k + 2$. Using standard arguments about trees in conjunction with the properties ensured by Lemma 5(4), we can show the following:

▶ **Lemma 13.** *The number of vertices in* $T_1, \ldots, T_c$ *is* $O(k^3)$ *in total.*

Proof. Since we already bounded the number of leaves of $T_1, \ldots, T_c$, it suffices to bound the number of internal vertices. Let D denote the set of internal vertices of $T_1, \ldots, T_c$ that have degree at least 3 in $\langle G \rangle$. Since the number $|\mathcal{L}|$ of leaves is $O(k^3)$, we have $|D| = O(k^3)$ as well by standard arguments on trees.

It remains to bound the number of internal vertices of degree 2 in $\langle G \rangle$. The number of such vertices that neighbor a leaf is $O(k^3)$, so it remains to consider vertices that have distance at least 2 to every leaf in $\langle G \rangle$.

We start by bounding the number of such vertices involved in a collision. First, consider collisions (u, m, v) for which m has degree at least 3 in $\langle G \rangle$. Such collisions involve at most $\sum_{v \in D} d_{\langle G \rangle}(v)$ vertices of degree 2 in $\langle G \rangle$. Since this sum is bounded by $2|\mathcal{L}|$, we obtain a bound of $O(k^3)$ for such vertices.

Now consider collisions (u, m, v) for which m has degree 2 in $\langle G \rangle$ but at least one of u, v has degree at least 3 in $\langle G \rangle$. The number of vertices of degree 2 involved in such collisions is at most $2\sum_{v \in D} d_{\langle G \rangle}(v) = O(k^3)$.

It remains to count vertices involved in collisions that do not touch an internal vertex of degree at least 3 in $\langle G \rangle$. Note that any such collision can overlap at most two others. Hence, by Lemma 9, at most $7k$ vertices can be involved, or we can reject (G, k) as a "no"-instance. Therefore, $O(k^3)$ internal vertices of degree 2 are involved in a collision.

Any internal vertex of degree 2 that is not involved in a collision must lie on a directed path between two vertices that are either of degree at least 3 or are involved in a collision. By another rule deferred to the full version of this paper, all directed paths have length at most 4; this leads to a bound of $4((\sum_{v \in D} d(v)) + O(k^3)) = O(k^3)$. ◀

Now, by Lemma 5(2), each vertex has weight at most $k + 1$, and thus this kernel can be encoded with $O((k^3) \log k)$ bits. This completes the proof of Theorem 1.

3 Polynomial Kernel for Out-Forest Vertex Deletion

Given a digraph G, we call a set $S \subseteq V(G)$ an *out-forest vertex deletion set* if $G - S$ is an out-forest. To obtain the polynomial kernel for Out-Forest Vertex Deletion Set, we proceed similarly as in the case of Out-Tree Vertex Deletion Set. Namely, we can argue that almost all reduction rules that apply to Out-Tree Vertex Deletion Set also work for Out-Forest Vertex Deletion Set. More precisely, one can argue that Lemmas 5, 6, 8, and 10, and 12 also apply. This way, we can bound the number of Type 1, Type 3 and Type 4 leaves by a polynomial in k in instances of Out-Forest Vertex Deletion Set that have been reduced by their respective rules.

However, Lemma 7 does not apply. Therefore, we cannot bound the number of Type 2 leaves by a polynomial in k in instances of Out-Forest Vertex Deletion Set that have

been reduced by all rules except the one of Lemma 7. Instead, we propose the following additional rule:

▶ **Lemma 14** (Out-Forest Source Rule). *Let (G, w, k) be an instance of* Out-Forest Vertex Deletion Set. *If G contains a vertex v of in-degree 0 with unique out-neighbor u that itself has v as its unique in-neighbor, then let $G' = G - v$ and let $w' = w_{|V(G')}$. Then (G, w) has an out-forest vertex deletion set of weight at most k if and only if (G', w') has an out-forest vertex deletion set of weight at most k.*

Crucially, the rule is invalid for Out-Tree Vertex Deletion Set, because if u is part of the deletion set for the resulting instance, then we might also need to add v to the deletion set in the original instance.

Once an instance of Out-Forest Vertex Deletion Set has been reduced by all previous rules except the one of Lemma 7 and additionally by the one of Lemma 14, we can also bound the number of Type 2 leaves by a polynomial in k.

▶ **Lemma 15.** *There are at most $3k^2(k+1)$ leaves of Type 2.*

Proof. We perform a marking scheme. Initially, no vertices are marked. Consider any unmarked leaf u of Type 2 for which its unique out-neighbor m is also unmarked. Since the rule of Lemma 14 cannot be performed, m has at least one other in-neighbor besides u. Let v be an arbitrary in-neighbor of m that is not u. Store the collision (u, m, v), and mark u, m, and v. Perform this procedure until no longer possible. At the end, each leaf of Type 2 has a marked vertex as its out-neighbor. Since each marked vertex has in-degree at most $k+1$ by Lemma 5(1), it suffices to bound the number of marked vertices.

Consider all collisions $Z = \{(u_i, m_i, v_i)\}$ that were stored during the marking scheme. Whenever we added (u, m, v), the vertices u and m were unmarked. Therefore, for each $i \neq j$, $\{u_i, m_i\} \cap \{u_j, m_j\} = \emptyset$. However, the v-vertices might coincide between the different stored collisions. Let $Y = \{v_i \mid (u_i, m_i, v_i) \in Z\}$. Since Lemma 9 does not apply, $|Y| \leq k$. Now consider the set Y_v of collisions that were stored for a fixed vertex $v \in Y$, i.e., $Y_v := \{(u_i, m_i, v_i) \in Z \mid v_i = v\}$. Since Lemma 6 cannot be applied, $|Y_v| \leq k$. Therefore, $|Z| \leq |Y| \cdot \max_{v \in Y}\{|Y_v|\} \leq k^2$, and thus the number of marked vertices is at most $3k^2$. ◀

4 Polynomial Kernel for Pumpkin Vertex Deletion Set

We first give a simple property of the problem, and use it to give our first reduction rule. Let (G, k) be an instance of Pumpkin Vertex Deletion Set. Let $\mathrm{HI} = \{v \in V(G) \mid d^-(v) \geq k+2\}$ and $\mathrm{HO} = \{v \in V(G) \mid d^+(v) \geq k+2\}$.

▶ **Lemma 16** (High Degree Rule). *If $|\mathrm{HI}| > k+1$ or $|\mathrm{HO}| > k+1$, then (G, k) is a "no"-instance.*

We now assume that (G, k) is an instance of Pumpkin Vertex Deletion Set to which Lemma 16 does not apply; that is, $|\mathrm{HI}|, |\mathrm{HO}| \leq k+1$. We call such an instance *primary-reduced*. We now split (G, k) into $(k+2)^2$ instances of the following problem.

Annotated Pumpkin Vertex Deletion Set	
Input:	A digraph G, an integer k, a set $S \subseteq V(G)$ with $\lvert S\rvert \leq 1$ and a set $T \subseteq V(G)$ with $\lvert T\rvert \leq 1$ and $S \cap T = \emptyset$, such that each vertex $v \in V(G) \setminus (S \cup T)$ satisfies $\max\{d^-(v), d^+(v)\} \leq k+1$.
Question:	Is there a set $X \subseteq V(G)$ with $\lvert X\rvert \leq k$ for which $G - X$ is a pumpkin with source s and sink t s.t. $S \subseteq \{s\}$ and $T \subseteq \{t\}$?

The *annotation* of the problem is a source and sink (when $S \neq \emptyset$ or $T \neq \emptyset$). We call it Fully Annotated Pumpkin Vertex Deletion Set if both S and T are non-empty.

Now, for each $S \subseteq \mathrm{HO}$ with $|\mathrm{HO} \cap S| \leq 1$ and for each $T \subseteq \mathrm{HI} \setminus S$ with $|\mathrm{HI} \cap T| \leq 1$, let $W_{S,T} = (\mathrm{HI} \cup \mathrm{HO}) \setminus (S \cup T)$ and construct the instance $I_{S,T} = (G - W_{S,T}, k - |W_{S,T}|, S, T)$ of Annotated Pumpkin Vertex Deletion Set. Let $\mathcal{I}$ denote the set of all such instances; we call this the *primary split* of (G, k). Observe that $|\mathcal{I}| \leq (k+2)^2$ since $|\mathrm{HI}|, |\mathrm{HO}| \leq k+1$.

▶ **Lemma 17.** *(G, k) is a "yes"-instance if and only if at least one of the instances of* Annotated Pumpkin Vertex Deletion Set *of the primary split of (G, k) is a "yes"-instance.*

4.1 Polynomial Kernel for Annotated Pumpkin Vertex Deletion Set

We start with several reduction rules. Let (G', k', S', T') be an instance of Annotated Pumpkin Vertex Deletion Set.

▶ **Lemma 18** (Source-Sink Rule)**.** *If G' has more than $k'+1$ vertices v with $d^-(v) = 0$ or more than $k'+1$ vertices v with $d^+(v) = 0$, then (G', k', S', T') is a "no"-instance.*

▶ **Lemma 19** (Long Path Rule)**.** *For distinct vertices $u, v \in V(G')$, let $P = \{w_0, \ldots, w_\ell\}$ be an induced directed u,v-path of G' of length $\ell > k'+2$ so that $\{w_1, \ldots, w_{\ell-1}\} \cap (S \cup T) = \emptyset$. Obtain G'' from G' by removing w_1 and adding the arc (w_0, w_2). Then (G', k', S', T') is a "yes"-instance if and only if (G'', k', S', T') is.*

▶ **Lemma 20** (Parallel Paths Rule)**.** *Let u, v be distinct vertices. If there are ℓ induced directed u,v-paths that do not contain a vertex of S' and $\ell > k'+2$, then let G'' be obtained from G' by removing all vertices of $\ell - (k'+2)$ arbitrary such paths. Then (G'', k', S', T') is a "yes"-instance if and only if (G', k', S', T') is.*

We now assume that (G', k', S', T') is an instance of Annotated Pumpkin Vertex Deletion Set on which Lemmas 18, 19, and 20 have no effect. We may thus assume that G has at most $k'+1$ vertices v with $d^-(v) = 0$ and that G has at most $k'+1$ vertices v with $d^+(v) = 0$. Moreover, each induced directed path has length at most $k'+2$. Finally, there are at most $k'+2$ induced directed paths between any (ordered) pair of vertices. We call such an instance *reduced*.

▶ **Lemma 21.** *Let (G', k', S', T') be a reduced instance of* Annotated Pumpkin Vertex Deletion Set*. If $|V(G')| > 2(k'+3)^2(2k'+3)^3$, then (G', k', S', T') is a "no"-instance.*

Proof. The rule can clearly be executed in linear time, so it remains to prove correctness. Suppose that (G', k', S', T') is a "yes"-instance, and let $X' \subseteq V(G')$ be any set such that $G' - X'$ is a pumpkin with source s' (where $S' \subseteq \{s'\}$) and sink t' (where $T' \subseteq \{t'\}$), and that $|X'| \leq k'$. We will say that a vertex $v \in V(G')$ has *low degree* if $\max\{d^-(v), d^+(v)\} \leq k'+1$.

We perform an iterative marking scheme. Initially, no vertices are marked. As long as this is possible, find an unmarked low-degree vertex $v \in V(G')$ such that v has at least two unmarked in-neighbors or at least two unmarked out-neighbors; then, mark v and its low-degree in- and out-neighbors.

We claim that at most $(k'+2)(2k'+3)$ vertices are marked. Since in each iteration we mark a low-degree vertex and (some of) its in- and out-neighbors, the number of vertices that are marked in an iteration is at most $2k'+3$. Hence, it suffices to bound the number of iterations. Consider any iteration and let v denote the vertex for this iteration. We count two iterations for the case that $v = s'$ or $v = t'$, and thus may assume that $v \neq s'$ and $v \neq t'$.

Without loss of generality assume that v has at least two unmarked in-neighbors. Then either $v \in X'$ or at least one of the unmarked in-neighbors of v must be in X'. Since the marking scheme proceeds iteratively, this means that after $k'+3$ iterations, we know that X' contains at least $k'+1$ vertices, a contradiction. Hence, there are at most $k'+2$ iterations, and the claim follows.

We now bound the total number of vertices v with $\max\{d^-(v), d^+(v)\} > 1$. Let v be any vertex such that $\max\{d^-(v), d^+(v)\} > 1$. Since the marking scheme was exhaustive, v has at least one marked neighbor. Since each marked vertex has low degree and there are at most $(k'+2)(2k'+3)$ marked vertices by the above claim, there can be at most $(k'+2)(2k'+3)^2$ vertices v with $\max\{d^-(v), d^+(v)\} > 1$.

To complete the proof, we notice that since the instance is reduced and thus Lemma 18 cannot be applied, there are at most $2k'+2$ vertices v with $\min\{d^-(v), d^+(v)\} = 0$. Let C denote the set of vertices v ($v \neq s', t'$) satisfying $\min\{d^-(v), d^+(v)\} = 0$ or $\max\{d^-(v), d^+(v)\} > 1$. We just proved that $|C| \leq (k'+3)(2k'+3)^2$. As any vertex $v \in V(G') \setminus (C \cup \{s', t'\})$ satisfies $d^-(v) = d^+(v) = 1$, all vertices of $V(G') \setminus (C \cup \{s', t'\})$ are on induced directed paths between vertices of $C \cup \{s', t'\}$. Since the instance is reduced and thus Lemma 19 cannot be applied, any such induced directed path has length at most $k'+2$. Moreover, because each vertex of C has low degree, a vertex of C is incident upon at most $2k'+2$ induced directed paths. Hence, there at most $|C|(2k'+2)(k'+1)$ vertices $v \in V(G') \setminus (C \cup \{s', t'\})$ in induced directed paths that have at least one vertex of C as an endpoint. Finally, there might be induced directed paths that have s' and t' as endpoints. However, since the instance is reduced and thus Lemmas 19 and 20 cannot be applied, there are at most $k'+2$ induced directed s', t'-paths, each of at most $k'+1$ internal vertices. Therefore, $|V(G')| \leq 2 + (k'+1)(k'+2) + |C| + |C|(2k'+2)(k'+1) \leq 2(k'+3)^2(2k'+3)^3$ and the lemma follows. ◀

▶ **Theorem 22.** Annotated Pumpkin Vertex Deletion Set *has a polynomial kernel with at most* $2(k'+3)^2(2k'+3)^3 = O(k'^5)$ *vertices.*

4.2 Polynomial Kernel for Pumpkin Vertex Deletion Set

Let (G, k) be an instance of Pumpkin Vertex Deletion Set. Find the primary split $\mathcal{I}$ of (G, k), and kernelize each of the resulting instances of Annotated Pumpkin Vertex Deletion Set using Theorem 22. Let $\mathcal{I}'$ denote the set of resulting instances of Annotated Pumpkin Vertex Deletion Set. Let $p(k) = 2(k+3)^2(2k+3)^3$; that is, $p(k)$ is the bound of Theorem 22.

To obtain the kernel, we need to know the source and sink of the pumpkin. Therefore, for each instance $I'_{S,T} \in \mathcal{I}'$ where $|S| = 0$ or $|T| = 0$, we create at most $(p(k))^{2-|S|-|T|}$ new instances of Fully Annotated Pumpkin Vertex Deletion Set as follows. Let $I'_{S,T} = (G', k', S, T)$. If $|S| = 0$ and $|T| = 1$, then for each $v \in V(G') \setminus T$, create a new instance $J_{v,T} = (G', k', \{v\}, T)$. We create similar instances if $|S| = 1$ and $|T| = 0$. If $|S| = 0$ and $|T| = 0$, then for each ordered pair $u, v \in V(G')$, create a new instance $J_{u,v} = (G', k', \{u\}, \{v\})$. Let $\mathcal{J}$ denote the set of these new instances and of all instances $I'_{S,T} \in \mathcal{I}'$ where $|S|, |T| > 0$. We call $\mathcal{J}$ the *secondary split* of (G, k). Observe that $|\mathcal{J}| \leq (k+2)^2(p(k))^2$, since each instance in $\mathcal{I}'$ has at most $p(k)$ vertices by Theorem 22. Moreover, each instance of $\mathcal{J}$ is indeed an instance of Fully Annotated Pumpkin Vertex Deletion Set. Similar to Lemma 17, we can now prove the following lemma.

▶ **Lemma 23.** (G, k) *is a "yes"-instance if and only if at least one of the instances of* Fully Annotated Pumpkin Vertex Deletion Set *of the secondary split of* (G, k) *is a "yes"-instance.*

Now consider the instances of the secondary split $\mathcal{J}$ of (G,k). Let (G',k',S',T') be such an instance. As argued before, $|V(G')| \leq p(k)$. Moreover, $|S'| = |T'| = 1$. We now add vertices such that the resulting graph has exactly $p(k)+3k+4$ vertices and that the source and sink are forced, even if we remove the annotation.

Let $(G',k',S',T') \in \mathcal{J}$. Since the instance is part of the secondary split, we know that $S' = \{s'\}$ for some $s' \in V(G')$ and that $T' = \{t'\}$ for some $t' \in V(G')$. Let A denote an arbitrary set of $\min\{k'+1, d^+(s')\}$ out-arcs of s'. For each arc $a \in A$, replace the arc by an induced directed path of length $p(k) - |V(G')| + k + k' + 3$. Then, add $k+2$ induced directed paths of length 2 from s' to t', and add $k-k'$ new vertices with an incoming arc from s'. Let G'_a denote the resulting graph. Observe that G'_a has exactly $p(k)+3k+4$ vertices, by construction. The resulting instance then is the instance (G'_a,k) of Pumpkin Vertex Deletion Set. Let $\mathcal{K}$ denote the set of these created instances for all instances of $\mathcal{J}$ combined. We call this the *tertiary split* of (G,k). Observe that, by construction, $|\mathcal{K}| \leq (k+1)(k+2)^2(p(k))^2$, since $k' \leq k$ by construction.

▶ **Lemma 24.** *(G,k) is a "yes"-instance if and only if at least one of the instances of* Pumpkin Vertex Deletion Set *of the tertiary split of (G,k) is a "yes"-instance.*

We can now complete the proof of Theorem 3 by simply taking a disjoint union of the instances of the tertiary split of (G,k) and setting the parameter to $k' = (|\mathcal{K}|-1)(p(k)+3k+4)+k$; the full proof can be found in the full version.

5 Arc Deletion Problems

We give part of the proof of Theorem 4 by giving the polynomial-time algorithm for Out-Forest Arc Deletion Set and a sketch of the polynomial-time algorithm for Pumpkin Arc Deletion Set. The polynomial-time algorithm for Out-Tree Arc Deletion Set boils down to running breadth-first search from each vertex of the graph and is deferred to the full version.

Out-Forest Arc Deletion Set. Notice that for any out-forest T of G, the graph $(V(G),T)$ has exactly $|V(G)|-|T|$ vertices of in-degree 0, which we refer to as the *roots* of $(V(G),T)$. Therefore, if $|T| = |V(G)|-1$, then $(V(G),T)$ is an out-tree.

Let $\mathcal{M}_1 = (E(G),\mathcal{I}_1)$ be the graphic matroid of $\langle G \rangle$, and let $\mathcal{M}_2 = (E(G),\mathcal{I}_2)$ be the partition matroid of G in which a set of arcs $I \subseteq E(G)$ is independent if and only if each vertex $v \in V(G)$ has at most one incoming arc in I. It follows that the set of out-forests of G is exactly the matroid intersection $\mathcal{M}_1 \cap \mathcal{M}_2$, that is, $\mathcal{M}_1 \cap \mathcal{M}_2 = (E(G), \mathcal{I}_1 \cap \mathcal{I}_2)$. (Notice that $\mathcal{M}_1 \cap \mathcal{M}_2$ is not generally a matroid itself.)

Using Edmond's Theorem, the matroid intersection polytope has an efficient separation oracle which consists of sequentially checking both $\mathcal{M}_1, \mathcal{M}_2$ separation oracles. Using the ellipsoid method to convert a separation oracle into an optimization algorithm allows us to construct a polynomial-time algorithm for optimization over the intersection polytope $P(\mathcal{M}_1 \cap \mathcal{M}_2)$. Linear programming duality combined with the matroid intersection theorem implies that we can find a maximum independent set of $\mathcal{M}_1 \cap \mathcal{M}_2$ in polynomial time. We refer to Schrijver's book [16, Theorem 41.1] for further explanation.

Therefore, we can find a maximum size out-forest of G in polynomial time. This proves that Out-Forest Arc Deletion Set can be solved in polynomial time.

Pumpkin Arc Deletion Set. We need the following auxiliary lemma.

▶ **Lemma 25.** *Given a digraph G and a pair of distinct vertices $s, t \in V(G)$, in polynomial time we can either find a set $Z \subseteq A(G)$ of smallest size such that the arcs of $G - Z$ induce a pumpkin with source s and sink t, or correctly answer that no $Z \subseteq A(G)$ exists such that the arcs of $G - Z$ induce a pumpkin with source s and sink t.*

Proof (Sketch). We perform the following algorithm to construct the set $Z \subseteq A(G)$; the proof of its correctness is deferred to the full version. First, add all incoming arcs of s and all outgoing arcs of t to Z, and remove these arcs from G. Now replace each vertex $v \in V(G) \setminus \{s, t\}$ by two vertices v^- and v^+, add an arc a_v from v^- to v^+, direct all incoming arcs of v towards v^-, and direct all outgoing arcs of v to start at v^+. Let G' denote the resulting graph. Let w be a cost function that assigns cost 0 to a_v for each $v \in V(G) \setminus \{s, t\}$ and cost (-1) to each other arc of G', and let c be a capacity function that assigns capacity 1 to each arc. Now find a minimum-cost s, t-flow f in G'; this takes polynomial time and yields a flow f that assigns flow 0 or 1 to each arc of G' [16, p. 177–181]. Add all arcs of $A(G)$ that are assigned flow 0 by f to Z. If the value of the flow f is 0, then answer "no". ◀

Now let (G, k) be an instance of Pumpkin Arc Deletion Set. For each pair of distinct vertices $s, t \in V(G)$, apply the algorithm of Lemma 25. Return "yes" if the size of the smallest set X found over all choices of s, t is at most k, and "no" otherwise. The correctness of the algorithm is immediate from Lemma 25. This proves that Pumpkin Arc Deletion Set can be solved in polynomial time.

6 Conclusions

In this paper, we took a different approach to generalizing Feedback Vertex Set and Tree Deletion Set to digraphs: instead of generalizing the property that the resulting graph should be 'acyclic', we generalized the property that the resulting graph should be a 'forest' and 'tree' respectively. The corresponding problems, Out-Forest Vertex Deletion Set and Out-Tree Vertex Deletion Set, were both shown to admit a polynomial kernel. We also considered Pumpkin Vertex Deletion Set, which in contrast to the previous two problems asks for the deletion to a digraph for which the underlying graph is not acyclic. We showed that Pumpkin Vertex Deletion Set admits a polynomial kernel as well.

In the past, efforts to find a polynomial kernel for Directed Feedback Vertex Set were aimed at considering restricted classes of digraphs [1, 7]. We believe that our work establishes a different line of attack that could help to resolve this longstanding open problem. In particular, all three studied problems are of the form "delete k vertices to an acyclic digraph that is a Π", where in our case Π is 'out-forest', 'out-tree', or 'pumpkin'. Therefore, we ask for which other properties Π does this problem have a polynomial kernel (parameterized by k) on general digraphs?

An interesting next step in the suggested research program would be to consider the problem to delete k vertices to obtain a *planar* acyclic digraph with a single source and a single sink. On the one hand, as evidenced by our polynomial kernel for Pumpkin Vertex Deletion Set, the restriction to a single source and a single sink can be quite helpful. On the other hand, there is no restriction on the (in-)degrees of the vertices, which neutralizes most of the reduction rules presented in this paper. Therefore, we believe that resolving Single-Source&Sink Planar Acyclic Digraph Vertex Deletion Set might yield crucial insights. (Note that without planarity condition, this problem is equivalent to Directed Feedback Vertex Set.)

Of course, this conclusion is not complete without asking the question whether Directed Feedback Vertex Set has a polynomial kernel.

References

1. F. N. Abu-Khzam. A kernelization algorithm for d-hitting set. *J. Comput. System Sci.*, 76(7):524–531, 2010. doi:10.1016/j.jcss.2009.09.002.
2. V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discrete Math.*, 12(3):289–297, 1999. doi:10.1137/S0895480196305124.
3. J. Bang-Jensen, A. Maddaloni, and S. Saurabh. Algorithms and kernels for feedback set problems in generalizations of tournaments. *Algorithmica*, to appear.
4. J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):21:1–21:19, November 2008. doi:10.1145/1411509.1411511.
5. M. Cygan, F. Fomin, B.M.P. Jansen, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. Open problems for FPT school. Technical report, Warsaw University, 2014. URL: http://fptschool.mimuw.edu.pl/opl.pdf.
6. M. Cygan, L. Kowalik, and M. Pilipczuk. Open problems from workshop on kernels. Technical report, Warsaw University, 2013. URL: http://worker2013.mimuw.edu.pl/slides/worker-opl.pdf.
7. M. Dom, J. Guo, F. Hüffner, R. Niedermeier, and A. Truß. Fixed-parameter tractability results for feedback set problems in tournaments. *J. Discrete Algorithms*, 8(1):76–86, 2010. doi:10.1016/j.jda.2009.08.001.
8. R.G. Downey and M.R. Fellows. *Fundamentals of Parameterized Complexity Theory*. Springer, 2013.
9. G. Even, J. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998. doi:10.1007/PL00009191.
10. M.R. Fellows, J. Guo, D. Marx, and S. Saurabh. Data reduction and problem kernels (Dagstuhl seminar 12241). Technical Report 2(6), Dagstuhl Reports, 2012.
11. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.
12. F.V. Fomin, S. Saurabh, and Y. Villanger. A polynomial kernel for proper interval vertex deletion. *SIAM J. Discrete Math.*, 27(4):1964–1976, 2013. doi:10.1137/12089051X.
13. A.C. Giannopoulou, D. Lokshtanov, S. Saurabh, and O. Suchy. Tree deletion set has a polynomial kernel (but no $OPT^{O(1)}$ approximation). In *Proc. FSTTCS 2014*, volume 29 of *LIPIcs*, pages 85–96, 2014.
14. R.M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pages 85–103, New York, 1972. Plenum.
15. D.G. Kirkpatrick and P. Hell. On the completeness of a generalized matching problem. In *Proc. STOC 1978*, pages 240–245, New York, 1978. ACM.
16. A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.
17. P.D. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15(2):281–288, 1995. doi:10.1007/BF01200760.
18. E. Speckenmeyer. On feedback problems in digraphs. In Manfred Nagl, editor, *Proc. WG 1990*, volume 411 of *Lecture Notes Comput. Sci.*, pages 218–231. Springer, 1990. doi:10.1007/3-540-52292-1_16.
19. S. Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2), 2010. doi:10.1145/1721837.1721848.
20. M. Yannakakis. The effect of a connectivity requirement on the complexity of maximum subgraph problems. *J. ACM*, 26(4):618–630, 1979. doi:10.1145/322154.322157.

Size-Treewidth Tradeoffs for Circuits Computing the Element Distinctness Function*

Mateus de Oliveira Oliveira

Institute of Mathematics – Czech Academy of Sciences, Zitná 25, 115 67 Praha 1, Czech Republic
mateus.oliveira@math.cas.cz

Abstract

In this work we study the relationship between size and treewidth of circuits computing variants of the element distinctness function. First, we show that for each n, any circuit of treewidth t computing the element distinctness function $\delta_n : \{0,1\}^n \to \{0,1\}$ must have size at least $\Omega(\frac{n^2}{2^{O(t)} \log n})$. This result provides a non-trivial generalization of a super-linear lower bound for the size of Boolean formulas (treewidth 1) due to Nečiporuk. Subsequently, we turn our attention to read-once circuits, which are circuits where each variable labels at most one input vertex. For each n, we show that any read-once circuit of treewidth t and size s computing a variant $\tau_n : \{0,1\}^n \to \{0,1\}$ of the element distinctness function must satisfy the inequality $t \cdot \log s \geq \Omega(\frac{n}{\log n})$. Using this inequality in conjunction with known results in structural graph theory, we show that for each fixed graph H, read-once circuits computing τ_n which exclude H as a minor must have size at least $\Omega(n^2/\log^4 n)$. For certain well studied functions, such as the *triangle-freeness* function, this last lower bound can be improved to $\Omega(n^2/\log^2 n)$.

1998 ACM Subject Classification F.2.3 Tradeoffs Between Complexity Measures

Keywords and phrases non-linear lower bounds, treewidth, element distinctness

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.56

1 Introduction

The problem of explicitly defining a function in NP which requires super-linear circuit size has proven to be notoriously hard. Currently, the best known lower bound for a function in NP is of the order[1] of $3n - o(1)$ for circuits with arbitrary fan-in-2 gates [4, 8], and of the order of $5n - o(1)$ for circuits with gates from the binary De-Morgan basis [18, 21]. In the particular case of boolean formulas, Nečiporuk proved an $\Omega(n^2/\log n)$ lower bound for the size of boolean formulas over the full binary basis computing the n-bit element distinctness function [24]. Intuitively, the element distinctness function $\delta_n : \{0,1\}^n \to \{0,1\}$ takes as input a sequence of m numbers $s_1, s_2, \ldots, s_m \in \{1, \ldots, m^2\}$ encoded as binary strings with $2 \log m$ bits, and returns 1 if and only if all numbers in this sequence are distinct. Remarkably, Nečiporuk's lower bound has resisted improvements during the last four decades, and remains the strongest known lower bound for the size of formulas over the full binary basis. In the restricted setting of formulas over the De-Morgan basis, a size lower bound of $n^{3-o(1)}$ was obtained by Håstad [15] using different techniques.

In this work, we consider the problem of proving circuit size lower bounds for circuits of low treewidth. During the past decade a considerable amount of research has been devoted to the

* This work was supported by the European Research Council, grant number 339691, in the context of the project Feasibility, Logic and Randomness (FEALORA).

[1] Recently, this lower bound was improved to $(3 + 1/86)n - o(n)$ [10].

© Mateus de Oliveira Oliveira;
licensed under Creative Commons License CC-BY

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 56; pp. 56:1–56:14

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

study of the computational power and the combinatorial properties of circuits parameterized by treewidth [1, 2, 6, 11, 12, 16, 19]. In our first result we generalize Nečiporuk's lower bound to the context of circuits of low treewidth.

▶ **Theorem 1.** *Let $\mathcal{C}$ be a circuit of treewidth t computing the element distinctness function δ_n. Then $\mathcal{C}$ has size $\Omega(\frac{n^2}{2^{O(t)}\log n})$.*

Here the size of a circuit $\mathcal{C}$ is defined as its wire-complexity, i.e., the total number of edges in $\mathcal{C}$. Therefore, our lower bound holds for circuits containing unbounded fan-in AND and OR gates, and more generally, unbounded fan-in associatively constructible gates, which we will define in Section 2. Theorem 1 generalizes Nečiporuk's non-linear lower bound, from the context of Boolean formulas (that is to say, circuits of treewidth 1) to the context of circuits of low treewidth. In particular, our result implies an $\Omega(n^2/\log n)$ lower bound for the size of circuits whose underlying undirected graph belongs to several interesting classes, such as trees (treewidth at most 1), TTSP series-parallel graphs (treewidth at most 2), outer-planar graphs (treewidth at most 2), Halin graphs (treewidth at most 3), k-outerplanar graphs for fixed k (treewidth at most $O(k)$), etc. Additionally, Theorem 1 implies non-linear lower bounds even for circuits of treewidth $o(\log n)$.

It is worth comparing our result with another prominent restricted family of circuits for which no non-linear lower bound is known, namely, circuits whose underlying graph belongs to the class of Valiant Series-Parallel graphs [31]. We refer to [7] for a clear definition of this class. It can be shown that the class of Valiant-series-parallel graphs strictly contains the class of TTSP-series-parallel graphs (which have treewidth 2). Nevertheless, Valiant-series-parallel graphs are incomparable with graphs of treewidth k, for $k \geq 3$. On the one hand, there are Valiant-series-parallel graphs of treewidth at least k for every $k \in \mathbb{N}$. For instance, the $k \times k$ grid-graph is Valiant-series-parallel but has treewidth k. On the other hand, it is easy to construct graphs of treewidth 3 which are not Valiant-series-parallel. Proving a non-linear lower bound for Valiant-series-parallel circuits remains a major open problem in circuit complexity [25, 28].

Next, we turn our attention to *read-once* circuits, which are circuits where each variable labels at most one input vertex. These circuits have also been known in the VLSI literature as *semilective* circuits [17]. Read-once circuits parameterized by treewidth have been studied by the SAT-solving and proof-complexity communities. Part of the interest in these circuits is due to the fact that the satisfiability problem for read-once circuits size s and treewidth t can be solved in time $2^{O(t)} \cdot s^{O(1)}$ [1, 2, 6, 12]. Questions related to the design of optimal VLSI circuits have motivated the study of the complexity of *planar* read-once circuits computing explicit functions (i.e. functions in NP). Within this line of research, quadratic lower bounds have been obtained for the size of planar read-once circuits computing both multiple-output functions [22] and single-output functions [29]. We contrast these quadratic lower bounds with the fact that for multilective planar circuits, i.e., planar circuits in which variables can label arbitrarily many input gates, the best known lower bounds are of the order of $O(n \log n)$ for single-output functions and of the order of $O(n^{3/2})$ for multiple-output functions [30].

In this work we introduce the *symmetric non-deterministic state complexity* (symmetric-NSC) of a Boolean function, a complexity measure that is lower-bounded by the size of the smallest read-once oblivious branching program computing the function in question. We show that if $\mathcal{C}$ is a read-once circuit of size s and treewidth t computing a function $f_n : \{0,1\}^n \to \{0,1\}$ of symmetric-NSC $snsc(f_n)$, then $t \cdot \log s \geq \Omega(\log snsc(f_n))$. Using this tradeoff in conjunction with known results from structural graph theory, we show that for

each fixed graph H, read-once H-minor-free[2] circuits computing f_n must have size at least $\Omega\left(\frac{\log^2 snsc(f_n)}{\log^2 n}\right)$. Subsequently, we introduce a variant $\tau_n : \{0,1\}^n \to \{0,1\}$ of the element distinctness function and show that its symmetric-NSC is lower bounded by $2^{\Omega(n/\log n)}$. From these results we have that read-once H-minor-free circuits computing τ_n require size $\Omega(n^2/\log^4 n)$. Near-quadratic lower bounds can also be obtained for the size of read-once H-minor-free circuits computing certain well studied functions, such as the *triangle-freeness* function Δ_n, and the *triangle-parity* function $\bigoplus \text{Clique}_{3,n}$. A result from [9] implies that the symmetric-NSC of these functions is lower-bounded by $2^{\Omega(n)}$. Therefore, read-once H-minor-free circuits computing both Δ_n and $\bigoplus \text{Clique}_{3,n}$ require size $\Omega(\frac{n^2}{\log^2 n})$.

2 Preliminaries

Let Σ be a finite set of symbols. A k-ary gate over Σ is a function $g : \Sigma^k \to \Sigma$. For $k \geq 3$, we say that a k-ary gate g is associatively constructible if there exists an associative operation $\oplus : \Sigma \times \Sigma \to \Sigma$ such that $g(x_1, \ldots, x_k) = x_1 \oplus \ldots \oplus x_k$. Alternatively, we say that g is a $\oplus$-gate of fan-in k. Unbounded fan-in AND and OR gates are clearly associatively constructible. An unbounded fan-in MOD_r gate can be simulated by an associatively constructible gate g that computes the sum of its inputs modulo r, together with a unary gate $g' : \Sigma \to \Sigma$ that returns 0 if this sum is congruent to 0 mod r, and which returns 1 otherwise.

An associatively constructible circuit with n inputs is a directed acyclic graph $\mathcal{C} = (V, E, \mathbf{g})$ where V is a set of vertices, E is a set of directed edges, and $\mathbf{g}$ is a function that labels each vertex $v \in V$ with a symbol from Σ, a gate over Σ, or a variable from $\{x_1, \ldots, x_n\}$. The function $\mathbf{g}$ must satisfy the following conditions:

1. If the in-degree of v is 0, then $\mathbf{g}(v)$ is either an element of Σ or a variable in $\{x_1, \ldots, x_n\}$.
2. If the in-degree of v is k, then $\mathbf{g}(v)$ is a k-ary gate over Σ. Additionally, if $k \geq 3$, then $\mathbf{g}(v)$ is associatively constructible.

Vertices of in-degree 0 are called *inputs*. An input is *initialized* if it is labeled with an element of Σ, and *uninitialized* if it is labeled with a variable. A formula is a circuit whose underlying graph is a tree. We say that a circuit $\mathcal{C} = (V, E, \mathbf{g})$ is *read once* if no two input vertices are labeled with the same variable. Since our circuits may contain associatively constructible gates of unbounded fan-in and unbounded fan-out, we define the size $|\mathcal{C}|$ of a circuit $\mathcal{C}$ as the number of edges in $\mathcal{C}$.

Below, we define the notion of *rooted carving decomposition* of a circuit, a variant of the notion of carving decomposition defined in [27]. If T is a tree, we denote by $nodes(T)$ the set of all nodes of T, and by $leaves(T)$ the set of all leaves of T. For each node $u \in nodes(T)$, we let $T[u]$ denote the subtree of T rooted at u.

▶ **Definition 2** (Carving Decomposition). A *rooted carving decomposition* of a circuit $\mathcal{C} = (V, E, \mathbf{g})$ is a pair (T, γ) where T is a binary tree and $\gamma : leaves(T) \to V$ is a bijection mapping each leaf $u \in leaves(\mathcal{T})$ to a single vertex $\gamma(u) \in V$.

Observe that the internal nodes of a carving decomposition $\mathcal{T}$ are unlabeled. Given a node $u \in nodes(T)$, we let $V(u) = \gamma(leaves(T[u])) = \{\gamma(v) \mid v \in leaves(T[u])\}$ be the image of the leaves of $T[u]$ under γ. For two distinct subsets V_1, V_2 of vertices of a circuit $\mathcal{C}$ we let

[2] We say that a circuit $\mathcal{C}$ is H-minor-free if its underlying undirected graph excludes H as a minor.

$E(V_1, V_2)$ denote the set of edges in G with one endpoint in V_1 and another endpoint in V_2. The width $carw(\mathcal{T})$ of the carving decomposition $\mathcal{T}$ is defined as

$$\max\{|E(V(u), V \backslash V(u))| : u \in nodes(T)\}.$$

The carving width $carw(\mathcal{C})$ of a circuit $\mathcal{C}$ is the minimum width of a carving decomposition of $\mathcal{C}$. The following lemma, whose proof is based on a result from [23], relates carving width and treewidth of a circuit.

▶ **Lemma 3** (From Tree-Decompositions to Carving Decompositions). *Let $\mathcal{C} = (V, E, \mathbf{g})$ be an associatively constructible circuit of treewidth t. There is a circuit $\mathcal{C}'$ of size $|\mathcal{C}'| \leq 2 \cdot |\mathcal{C}|$, maximum degree 3, and carving width at most $3t + 3$ such that $\mathcal{C}$ and $\mathcal{C}'$ compute the same function.*

3 Nečiporuk's Method

In this section we briefly describe Nečiporuk's method for proving non-linear lower bounds on the size of Boolean formulas over the complete binary basis. For our purposes, it will be convenient to divide this method into three steps. Our first main result (Theorem 1) follows from a generalization of Step 1 given below. A complete proof of Nečiporuk's theorem can be found in [20].

Step 1: Let $X = \{x_1, \ldots, x_n\}$ be a set of variables, $f : \{0,1\}^X \to \{0,1\}$ be a Boolean function on X, and $Y \subseteq X$ be a subset of variables of X. We denote by $N_f(Y)$ the number of distinct functions that can be obtained by initializing all variables in $X \backslash Y$ with values in $\{0,1\}$. The first step in the proof of Nečiporuk's theorem consists in providing an upper bound for $N_f(Y)$. If f can be computed by a Boolean formula F, such upper bound can be given in terms of the number of inputs of F labeled with variables in Y.

▶ **Proposition 4.** *Let $f : \{0,1\}^X \to \{0,1\}$ be a function computable by a boolean formula F. Let $Y \subseteq X$ be a subset of variables such that at most l inputs of F are labeled with variables in Y. Then $N_f(Y)$ is at most $2^{O(l)}$.*

Note that if $g : \{0,1\}^Y \to \{0,1\}$ is a function obtained from f by initializing all variables in $X \backslash Y$, then g can be represented by a boolean formula F_g with l uninitialized inputs which is obtained from F by initializing all inputs labeled with variables in $X \backslash Y$. The proof of Proposition 4 follows by noting that the Boolean formula F_g can be simplified into a Boolean formula F'_g also computing the function g, in such a way that F'_g has at most l inputs, all of which are uninitialized, and in which all internal nodes have fan-in 2. This implies that F'_g has at most $l - 1$ internal nodes. Since there are 16 possible Boolean functions of fan-in 2, there are at most 16^{l-1} choices for g.

Step 2: The second step consists in exhibiting an explicit Boolean function with many sub-functions. Intuitively, a function $f : \{0,1\}^X \to \{0,1\}$ has many sub-functions if the quantity $N_f(Y)$ is large for some subsets $Y \subseteq X$ of suitable size. Let $X = \{x_1, \ldots, x_n\}$ be a set of $n = 2m \log m$ distinct variables partitioned into m blocks $Y_1, Y_2, \ldots, Y_m$, where each block Y_i has $2 \log m$ variables. The *element distinctness* function $\delta_n : \{0,1\}^X \to \{0,1\}$ is defined as follows for each assignment $s_1, s_2, \ldots, s_m$ of the blocks $Y_1, Y_2, \ldots, Y_m$ respectively.

$$\delta_n(s_1, s_2, \ldots, s_m) = \begin{cases} 1 & \text{if } s_i \neq s_j \text{ for } i \neq j, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The following lemma states that the element distinctness function defined in Equation 1 has many sub-functions.

▶ **Lemma 5** (See [20], Section 6.5). *Let $\delta_n : \{0,1\}^X \to \{0,1\}$ be the element distinctness function defined in Equation 1, where $|X| = n$ and $X = Y_1 \dot\cup Y_2 \dot\cup \ldots \dot\cup Y_m$ with $|Y_i| = 2 \log m$. Then for each $i \in \{1, \ldots, m\}$, $N_{\delta_n}(Y_i) \geq 2^{\Omega(n)}$.*

Step 3: In the third step, we combine Proposition 4 with Lemma 5 to obtain a non-linear lower bound for the size of Boolean formulas computing the element distinctness function $\delta_n : \{0,1\}^X \to \{0,1\}$ defined in Equation 1. Let F be a Boolean formula computing δ_n. Let l_i denote the number of inputs of F labeled with some variable in Y_i. By Proposition 4, we have that $N_{\delta_n}(Y_i) \leq 2^{O(l_i)}$. On the other hand, by Lemma 5, $N_{\delta_n}(Y_i) \geq 2^{\Omega(n)}$. Combining these two inequalities, we have that

$$2^{O(l_i)} \geq N_{\delta_n}(Y_i) \geq 2^{\Omega(n)}. \tag{2}$$

This implies that $l_i \geq \Omega(n)$. In other words, there are $\Omega(n)$ inputs of F labeled with variables from Y_i. Since there are $m = \Omega(\frac{n}{\log n})$ blocks Y_i, we have that the number of inputs of F labeled with variables in X is at least $\Omega(\frac{n^2}{\log n})$. □

3.1 Generalizing Nečiporuk's Theorem

In this section we will generalize Nečiporuk's non-linear lower bound to the context of circuits of low treewidth (Theorem 1). We call attention to the fact that this lower bound concerns circuits in which each variable can label arbitrarily many input vertices. The following lemma, which generalizes Proposition 4, is the main technical result towards the proof of Theorem 1.

▶ **Lemma 6.** *Let $f : \{0,1\}^X \to \{0,1\}$ be a function computable by a boolean circuit $\mathcal{C}$ of treewidth t. Let $Y \subseteq X$ be a subset of variables such that at most l inputs of $\mathcal{C}$ are labeled with variables in Y. Then $N_f(Y)$ is at most $2^{l \cdot 2^{O(t)}}$.*

The next two subsections will be dedicated to the proof of Lemma 6. Before, we show how Lemma 6 can be used to prove Theorem 1.

Proof of Theorem 1. Let $|X| = n = 2m \log m$, and $Y_1, \ldots, Y_m$ be a partition of the variables in X, where for each i, $|Y_i| = 2 \log m$. Let l_i be the number of inputs of $\mathcal{C}$ labeled with a variable from Y_i. By Lemma 5, $N_{\delta_n}(Y_i) \geq 2^{\Omega(n)}$. On the other hand, by Lemma 6, $N_{\delta_n}(Y_i) \leq 2^{l_i \cdot 2^{O(t)}}$. Therefore, by combining these two inequalities, we have $2^{l_i \cdot 2^{O(t)}} \geq N_{\delta_n}(Y_i) \geq 2^{\Omega(n)}$. This implies that $l_i \geq \Omega(n/2^{O(t)})$. Since there are $m = \Omega(\frac{n}{\log n})$ blocks of variables Y_i, we have that the number of inputs of $\mathcal{C}$ is at least $\frac{n^2}{2^{O(t)} \cdot \log n}$. ◀

3.2 Defining Relations via Constraint Satisfaction Problems

In this section we will introduce some terminology and basic results which will be used in the proof of Lemma 6. Let X be a set of variables. An *assignment* of X is a function $a : X \to \{0,1\}$ that associates with each variable $x \in X$ a value $a(x) \in \{0,1\}$. We let $\{0,1\}^X$ denote the set of all assignments of X. A *relation* over X is any subset $R \subseteq \{0,1\}^X$. We say that each variable $x \in X$ is *constrained* by R. In some places we write $var(R)$ to denote the set of variables constrained by R. If a is an assignment of X, and $Y \subseteq X$, then we let $a|_Y$ denote the *restriction* of a to Y. More precisely, for each $x \in Y$, $a|_Y(x) = a(x)$. We say that an assignment $a \in \{0,1\}^X$ *satisfies a relation* R over $Y \subseteq X$ if $a|_Y \in R$. If

$R \subseteq \{0,1\}^X$ is a relation over X, and $Y \subseteq X$, then the restriction of R to Y is the relation $R|_Y = \{a|_Y \mid a \in R\}$. The following immediate observation says the result of restricting a relation $R \subseteq \{0,1\}^X$ to a subset X' and subsequently to a subset $Y \subseteq X'$ is equivalent to restricting R directly to Y.

▶ **Observation 1.** *Let R be a relation over X and let $Y \subseteq X' \subseteq X$. Then $R|_Y = (R|_{X'})|_Y$.*

Below, we define the notion of *constraint satisfaction problem* over X.

▶ **Definition 7.** A constraint satisfaction problem (CSP) over a set of variables X is a set of relations

$$K = \{R_1, R_2, \ldots, R_r\} \tag{3}$$

where for each $i \in \{1, \ldots, r\}$, R_i is a relation over some subset $X_i \subseteq X$ of variables.

We note that two relations R_i and R_j in K in which $var(R_i) \neq var(R_j)$ are considered to be different. A CSP K over a set of variables X can be used to define a relation $R(K)$ over X. Intuitively, the relation $R(K)$ consists of all assignments over X that satisfy each relation in K.

$$R(K) = \{a \in \{0,1\}^X \mid a|_{X_i} \in R_i \text{ for } i \in \{1, \ldots, r\}.\} \tag{4}$$

Let K be a CSP over a set of variables X and let $S \subseteq K$. We denote by $c(S)$ the set of variables that are simultaneously constrained by some relation in S and some relation $K \backslash S$. We say that $c(S)$ is the *cutset* of S with respect to K. Given a CSP K over a set of variables X, and a subset $Y \subseteq X$, we will deal with the problem of obtaining a CSP K' with less relations than K, but with the property that $R(K)|_Y = R(K')|_Y$. The following simple lemma will be crucial for this goal.

▶ **Lemma 8.** *Let $K = \{R_1, \ldots, R_r\}$ be a CSP over a set of variables X, let $Y \subseteq X$, and $S \subseteq K$ be such that $var(S) \cap Y \subseteq c(S)$. Consider the CSP*

$$K' = (K \backslash S) \cup \{R(S)|_{c(S)}\}.$$

Then $R(K)|_Y = R(K')|_Y$.

3.3 Circuits vs CSPs

In this section we prove Lemma 6. The idea behind the proof is the following. Let $\mathcal{C}$ be a circuit of carving width w computing a function $f : \{0,1\}^Y \to \{0,1\}$. As a fist step, we associate with $\mathcal{C}$ a CSP $K(\mathcal{C})$ over a set of variables $Y \cup \{x_e \mid e \in E\}$. This CSP has the property that $R(K(\mathcal{C}))|_Y$ consists precisely of those assignments that cause $\mathcal{C}$ to evaluate to 1. In a second step, we use the fact that $\mathcal{C}$ has carving width w to obtain a new CSP K' such that each relation in K' constrains at most w variables, and such that the number of relations in K' is proportional to the number of uninitialized inputs of $\mathcal{C}$. This new CSP K' has the property that $R(K')|_Y = R(K(\mathcal{C}))|_Y$. Finally, if we are given a function $f : \{0,1\}^X \to \{0,1\}$ and a subset $Y \subseteq X$ of variables, then we will have that there are at most $2^{2^{O(w)}}$ distinct functions arising by restricting all variables in $X \backslash Y$ to values in $\{0,1\}$.

▶ **Definition 9** (CSP Derived from a Circuit)**.** Let $\mathcal{C} = (V, E, \mathbf{g})$ be a circuit whose inputs are labeled with variables from Y. We let $K(\mathcal{C}) = \{R_v \mid v \in V\}$ be the CSP over the variables $Y \cup \{x_e \mid e \in E\}$ which is defined as follows.

1. If v is an input vertex labeled by $\mathbf{g}$ with a variable x, and v is the source of edges $e_1, \ldots, e_k$, then R_v is a relation over the variables $Y_v = \{x, x_{e_1}, \ldots, x_{e_k}\}$, and an assignment $a : Y_v \to \Sigma$ is in R_v if and only if

   $$a(x) = a(x_{e_1}) = \ldots = a(x_{e_k}).$$

2. If v is an internal vertex labeled with a gate $\mathbf{g}(v)$, v is the target of edges $e_1, \ldots e_k$, and v is the source of edges $e'_1, \ldots, e'_{k'}$, then R_v is a relation over the variables $Y_v = \{x_{e_1}, \ldots, x_{e_k}, x_{e'_1}, \ldots, x_{e'_{k'}}\}$, and an assignment $a : Y_v \to \Sigma$ is in R_v if and only if

   $$\mathbf{g}(v)(a(x_{e_1}), \ldots, a(x_{e_k})) = a(x_{e'_1}) = \ldots = a(x_{e'_{k'}}).$$

3. If v is the output vertex of $\mathcal{C}$, and v is the target of edges $e_1, \ldots, e_k$, then R_v is a relation over the variables $Y_v = \{x_{e_1}, \ldots, .x_{e_k}\}$ and $a : \{x_{e_1}, \ldots, x_{e_k}\} \to \Sigma$ is in R_v if and only if

   $$\mathbf{g}(v)(x_{e_1}, \ldots, x_{e_k}) = 1.$$

Intuitively, the variables Y are input variables of the circuit $\mathcal{C}$, while the variables $\{x_e \mid e \in E\}$ are used to keep track of the evaluation of the circuit $\mathcal{C}$ when the variables in Y are initialized. The relation $R(K(\mathcal{C}))$ associated with the CSP $K(\mathcal{C})$ contains all assignments of $Y \cup \{x_e \mid e \in E\}$ which encode an initialization of the input variables together with an evaluation of the gates of the circuits which evaluate to 1. If we restrict the relation $R(K(\mathcal{C}))$ to the variables in Y, then we recover precisely the set of assignments that cause $\mathcal{C}$ to evaluate to 1.

▶ **Observation 2.** *Let $\mathcal{C}$ be a circuit computing a function $f : \{0,1\}^Y \to \{0,1\}$. Let $K(\mathcal{C})$ be the CSP associated with $\mathcal{C}$. Then $R(K(\mathcal{C}))|_Y = \{a \in \{0,1\}^Y \mid f(a) = 1\}$.*

We note that the number of relations in $R(K(\mathcal{C}))$ is precisely the number of gates of $\mathcal{C}$, and therefore there is no a priori correspondence between the number of relations in $R(K(\mathcal{C}))$ and the number of inputs of $\mathcal{C}$ labeled by variables in Y. The following theorem says that if $\mathcal{C}$ is a circuit of carving width w then one can construct a CSP K whose size is proportional to the number of inputs of $\mathcal{C}$ labeled with variables in Y, in such a way that the number of variables constrained by each relation in K is proportional to w, and such that $R(K)|_Y = R(K(\mathcal{C}))|_Y$.

▶ **Theorem 10** (CSP Reduction)**.** *Let $\mathcal{C}$ be a circuit of carving width w computing a function $f : \{0,1\}^Y \to \{0,1\}$. Let $l \geq |Y|$ be the number of inputs of $\mathcal{C}$ labeled with variables in Y. Then there exits a CSP $K = \{R_1, \ldots, R_k\}$ with $k \leq 3 \cdot l$ such that for each $i \in \{1, \ldots, k\}$, R_i constrains at most $2 \cdot w$ variables and such that $R(K)|_Y = \{a \in \{0,1\}^Y \mid f(a) = 1\}$.*

It is worth noting that the CSP K in Theorem 10 is obtained from the CSP $K(\mathcal{C})$ by applying several non-trivial simplification steps. Before proving Theorem 10 we show how this theorem can be used to prove Lemma 6.

Proof of Lemma 6. Let $f : \{0,1\}^X \to \{0,1\}$ be a boolean function which is computable by a circuit $\mathcal{C}$ of treewidth t. By Lemma 3, there exists a circuit $\mathcal{C}'$ of size $|\mathcal{C}'| \leq 2 \cdot |\mathcal{C}|$ such that $\mathcal{C}'$ of maximum degree 3 and carving width at most $w = 3(t+1)$ such that $\mathcal{C}'$ computes f. Now let $Y \subseteq X$. Then each initialization of the variables in $X \backslash Y$, gives rise to a circuit $\mathcal{C}''$ in which all l uninitialized inputs are labeled with variables in Y. By Theorem 10, there is a CSP K with at most $3 \cdot l$ relations, such that each relation R in K constrains at most $2 \cdot w = 6(t+1)$ variables, and such that $R(K)|_Y$ is precisely the set of assignments of the variables in Y which cause the circuit $\mathcal{C}''$ to evaluate to 1. This implies that there are at most $2^{2^{6(t+1)} \cdot 3 \cdot l}$ possible distinct functions arising from the restrictions of variables not in Y. ◀

In the remainder of this section, we prove Theorem 10. Let $\mathcal{C} = (V, E, \mathbf{g})$ be a circuit of carving-width w computing a function $f : \Sigma^Y \to \Sigma$. Let $K = K(\mathcal{C}) = \{R_v \mid v \in V\}$ be the CSP associated with $\mathcal{C}$. We note that $Y \cup \{x_e \mid e \in E\}$ is the set of variables constrained by relations in K. We say that a relation R_v is a Y-relation if R_v constrains some variable in Y. Let (T, γ) be a carving decomposition of (V, E). For a node u of T we let $\mathit{leaves}(T[u], Y)$ denote the set of leaves u' of $T[u]$ such that the relation $R_{\gamma(u')}$ is a Y-relation. We say that a node $u \in \mathit{nodes}(T)$ is a Y-node if u is either a leaf such that $R_{\gamma(u)}$ is a Y-relation, or if u is an internal node $u \in \mathit{nodes}(T)$ such that $\mathit{leaves}(T[u.l], Y) \neq \emptyset$ and $\mathit{leaves}(T[u.r], Y) \neq \emptyset$. If u is a Y-node, then we say that a node $u' \neq u$ is the Y-parent of u if u' is the ancestor of u at minimal distance from u with the property that u' is itself a Y-node. We let $\mathit{nodes}(T, Y)$ denote the set of all Y-nodes of T.

▶ **Lemma 11.** $|\mathit{nodes}(T, Y)| = 2 \cdot |\mathit{leaves}(T, Y)| - 1$.

Intuitively, the idea of the proof of Lemma 11 consists in showing that the set of all Y-nodes of T induces a binary tree. Since a binary tree with $|\mathit{leaves}(T, Y)|$ leaves has $|\mathit{leaves}| - 1$ internal nodes, the total number of Y-nodes is $2 \cdot |\mathit{leaves}(T, Y)| - 1$. Now let $T' = T \backslash \mathit{nodes}(T, Y)$ be the forest which is obtained by deleting from T all of its Y-nodes. We note that the number of connected components in the forest T' is at most $|\mathit{nodes}(T, Y)| = 2|\mathit{leaves}(T, Y)| - 1$. We let $T_1, \ldots, T_k$, for $k \leq |\mathit{nodes}(T, Y)|$ be the connected components of T'. For each $i \in \{1, \ldots, k\}$, let $S_i = \{R_v \mid \exists u \in \mathit{leaves}(T_i),\ \gamma(u) = v\}$ be the sub-CSP of $K(\mathcal{C})$ formed by the relations associated to vertices of $\mathcal{C}$ that label the leaves of the connected component T_i. Let $c(S_i) = \mathit{var}(K(\mathcal{C}) \backslash S_i) \cap \mathit{var}(S_i)$ be the cut-set of S_i with respect to $K(\mathcal{C})$. In other words, $c(S_i)$ is the set of variables that are constrained by some relation in S_i, and another relation in $K \backslash S_i$. Note that $c(S_i) \cap Y = \emptyset$. The fact that (T, γ) is a carving decomposition of $\mathcal{C}$ of width w implies that the number of variables in $c(S_i)$ is at most $2 \cdot w$. Let $R_i = R(S_i)|_{c(S_i)}$. Then we define our CSP as follows.

$$K = \left(K(\mathcal{C}) \backslash \bigcup_{i=1}^{k} S_i \right) \cup \bigcup_{i=1}^{k} \{R_i\} \tag{5}$$

Note that each subset of relations $S_i \subseteq K(\mathcal{C})$ corresponding to the connected component T_i is replaced by a unique relation R_i. By Lemma 11 there are at most $2 \cdot |\mathit{leaves}(T, Y)| - 1$ connected components in T'. Therefore, the number of relations in K is upper-bounded by $|\mathit{leaves}(T, Y)| + 2|\mathit{leaves}(T, Y)| - 1 = 3|\mathit{leaves}(T, Y)|$. We claim that $R(K)|_Y = R(K(\mathcal{C}))|_Y$. To prove this claim, let $K_0, K_1, \ldots, K_k$ be a sequence of CSPs where $K_0 = K(\mathcal{C})$, and for each $i \in \{1, \ldots, k\}$, $K_i = (K_{i-1} \backslash S_i) \cup \{R_i\}$. Then clearly we have that $K = K_k$. We claim that for each $j \in \{0, \ldots, k\}$, $K_j|_Y = K(\mathcal{C})|_Y$. In the base case, $k = 0$, and the claim follows trivially. Now assume that $K_j|_Y = K(\mathcal{C})|_Y$. By Lemma 8, we have that $K_j|_Y = K_{j+1}|_Y$. □

4 Symmetric Non-deterministic State Complexity

Let Σ be a finite set of symbols. In this section we introduce the notion of symmetric non-deterministic state complexity of functions of the form $f : \Sigma^n \to \{0, 1\}$ and of finite languages included in Σ^n. We note that this notion is intimately related with the size of the smallest non-deterministic oblivious, read-once branching program [26] computing f. A non-deterministic finite automaton (NFA) over Σ is a 5-tuple $\mathcal{A} = (Q, \Sigma, \mathfrak{R}, Q_0, F)$ where Q is a set of states, $Q_0 \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of final states and $\mathfrak{R} \subseteq Q \times \Sigma \times Q$ is a transition relation. We write $q \xrightarrow{a} q'$ to denote that the triple (q, a, q') belongs to $\mathfrak{R}$. We say that a string $w = w_1 w_2 \ldots w_n \in \Sigma^n$ is accepted by $\mathcal{A}$ if there is a

sequence $q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} \ldots \xrightarrow{w_n} q_n$ such that $q_0 \in Q_0$ and $q_n \in Q_F$. We denote by $\mathcal{L}(\mathcal{A})$ the set of all strings accepted by $\mathcal{A}$.

Let $\mathcal{L} \subseteq \Sigma^n$ be a set of length-n strings over Σ. The *non-deterministic* state complexity (NSC) of $\mathcal{L}$, denoted $nsc(\mathcal{L})$, is defined as the minimum number of states of a NFA accepting $\mathcal{L}$. Let $f : \Sigma^n \to \{0,1\}$ be a function. We denote by $\mathcal{L}(f)$ the set of all strings $w \in \Sigma^n$ for which $f(w) = 1$. We define the non-deterministic state complexity of f as $nsc(f) := nsc(\mathcal{L}(f))$. For each positive integer n, we define $[n] = \{1, \ldots, n\}$. We denote by $Perm(n)$ the set of all permutations of the set $[n]$. If $\pi : [n] \to [n]$ is a permutation in $Perm(n)$ and $w \in \Sigma^n$, then we let $\pi(w)$ be the string in Σ^n that is defined by setting $\pi(w)_{\pi(j)} = w_j$ for each $j \in [n]$. Intuitively, the j-th position of w is mapped to the position $\pi(j)$ of $\pi(w)$. If $\mathcal{L} \subseteq \Sigma^n$ is a set of length-n strings over Σ, then we denote by $\pi(\mathcal{L})$ the language obtained from $\mathcal{L}$ by permuting the coordinates of each string in $\mathcal{L}$ according to π. More precisely,

$$\pi(\mathcal{L}) = \{\pi(w) \mid w \in \mathcal{L}\}. \tag{6}$$

The symmetric non-deterministic state complexity (symmetric-NSC) of a language $\mathcal{L} \subseteq \Sigma^n$ is defined as the minimum non-deterministic state complexity of a permuted version of $\mathcal{L}$.

▶ **Definition 12** (Symmetric Nondeterministic State Complexity)**.** Let $\mathcal{L} \subseteq \Sigma^n$. The symmetric non-deterministic state complexity of $\mathcal{L}$ is defined as

$$snsc(\mathcal{L}) = \min_{\pi \in Perm(n)} nsc(\pi(\mathcal{L})). \tag{7}$$

The symmetric-NSC of a function $f : \Sigma^n \to \{0,1\}$ is defined as $snsc(f) = snsc(\mathcal{L}(f))$.

We note that the symmetric-NSC of a function $f : \{0,1\}^n \to \{0,1\}$ is lower-bounded by the size of the smallest non-deterministic oblivious read-once $|\Sigma|$-way branching program computing f. Therefore, functions requiring exponential size branching programs of this particular form have exponential symmetric non-deterministic state complexity. Two examples of such functions are the *triangle-freeness* function $\Delta_n : \{0,1\}^n \to \{0,1\}$, and the *triangle-parity* function $\bigoplus \text{Clique}_{3,n} : \{0,1\}^n \to \{0,1\}$. Both functions take as input an array $x = (x_{ij})_{1 \leq i < j \leq m}$ consisting of $n = \binom{m}{2}$ Boolean variables representing an undirected graph $G(x)$ on m vertices $\{1, \ldots, m\}$. The graph $G(x)$ has an edge connecting vertices i and j, with $i < j$, if and only if $x_{ij} = 1$. The triangle-freeness function Δ_n returns 1 on an input x if and only if the graph $G(x)$ does not contain a triangle. The triangle-parity function $\bigoplus \text{Clique}_{3,n}$ returns 1 if and only if the parity of the number of the triangles in $G(x)$ is odd. In [9] it was shown that read-once non-deterministic branching programs computing the functions Δ_n and $\bigoplus \text{Clique}_{3,n}$ require size $2^{\Omega(n)}$. Therefore, the same lower bound holds for the symmetric-NSC of these functions.

▶ **Theorem 13** ([9])**.** *$snsc(\Delta_n) \geq 2^{\Omega(n)}$ and $snsc(\bigoplus \text{Clique}_{3,n}) \geq 2^{\Omega(n)}$.*

4.1 On a Variant of the Element Distinctness Function

We say that a binary string w is *even* if w has an even number of ones. Analogously, we say that w is *odd* if w has an odd number of ones. For each $r \in \mathbb{N}$, we let $even(r)$ denote the set of all strings of even parity in the set $\{0,1\}^r$. Let $\Sigma(m) = \{1, \ldots, m\}$, and $P(m) \subseteq \Sigma(m)^m$ be the set of all length-m strings over $\Sigma(m) = \{1, \ldots, m\}$ whose entries are pairwise distinct. Let $n = (\lceil \log m \rceil + 1) \cdot m$ and let $b : \{1, \ldots, m\} \to even(\lceil \log m \rceil + 1)$ be an injection that maps each number $j \in \{1, \ldots, m\}$ to an even binary string $b(j)$ of length $\lceil \log m \rceil + 1$. We let

$$B(n) = \{b(w_1)b(w_2)\ldots b(w_m) \in \{0,1\}^n \mid w_1 w_2 \ldots w_m \in P(m)\}$$

be the binary language that is obtained from $P(m)$ by mapping each string in $P(m)$ to its binary representation. We define the *even element distinctness function* $\tau_n : \{0,1\}^n \to \{0,1\}$ as the function that returns 1 on an input $w \in \{0,1\}^n$ if and only if $w \in B(n)$. Note that by definition, the symmetric-NSC of τ_n is the symmetric-NSC of $B(n)$.

▶ **Theorem 14.** *The function $\tau_n : \{0,1\}^n \to \{0,1\}$ has* symmetric-*NSC $2^{\Omega(n/\log n)}$.*

The proof of Theorem 14 will use the following result.

▶ **Theorem 15** (Glaister-Shallit [13])**.** *Let $\mathcal{L} \subseteq \Sigma^n$ be a set of length-n strings over Σ, and suppose that there exists a set $F = \{(x_i, w_i) \mid 1 \leq i \leq k\}$ of pairs of strings such that*
1. *$x_i \cdot w_i \in \mathcal{L}$ for $1 \leq i \leq k$*
2. *$x_i \cdot w_j \notin \mathcal{L}$ for $1 \leq i, j \leq k$ and $i \neq j$*

Then any non-deterministic finite automaton accepting $\mathcal{L}$ has at least k states.

The set F in Theorem 15 is called a fooling set for $\mathcal{L}$. We will prove Theorem 14 by constructing, for each permutation $\pi : [n] \to [n]$, a fooling set F_π of size $2^{\Omega(n/\log n)}$ for the language $\pi(B(n))$. To construct F_π it will be convenient to view strings as Boolean functions over sets of positions. In other words, if S is a set of positive integers, then a string over S is simply a Boolean function $w : S \to \{0,1\}$. We note that we allow S to be any set of positive integers and not necessarily an interval of the form $[n] = \{1, \ldots, n\}$. The parity of w is defined as the parity of the number of positions in which w evaluates to 1: $par(w) = |\{i \in S \mid w(i) = 1\}| \mod 2$. The restriction of w to a subset $T \subseteq S$ is the string $w|_T : T \to \{0,1\}$ which is defined by setting $w|_T(i) = w(i)$ for every $i \in T$. If $S \subseteq [n]$, $w : S \to \{0,1\}$ is a string, and $\pi : [n] \to [n]$ is a permutation, then we let $\pi(w)$ be the string $w' : \pi(S) \to \{0,1\}$ that is defined by setting $w'(\pi(i)) = w(i)$ for each $i \in S$. We let $L = \{1, \ldots, \lfloor n/2 \rfloor\}$ and $R = \{\lfloor n/2 \rfloor + 1, \ldots, n\}$ be respectively the first and the second halves of the set $[n] = \{1, \ldots, n\}$. We say that a permutation π splits a subset $S \subseteq [n]$ if $\pi(S) \cap L \neq \emptyset$ and $\pi(S) \cap R \neq \emptyset$. In other words, π splits S if some elements of S are mapped by π to the first half of $[n]$ and some elements of S are mapped by π to the second half of $[n]$.

If S and S' are subsets of $[n]$ such that $S \cap S' = \emptyset$, and $w : S \to \{0,1\}$ and $w' : S' \to \{0,1\}$ are strings with domain S and S' respectively, then the concatenation of w with w' is simply the function $w \cdot w' : S \cup S' \to \{0,1\}$ which is equal to w when restricted to S and equal to w' when restricted to S'. Let $S = \{j_1, j_2, \ldots, j_k\} \subseteq [n]$ where $j_1 < j_2 < \ldots < j_k$. We let $ord_S(i) = j_i$ denote the i-th element of S. Let S and S' be subsets of $[n]$ of same size. We say that strings $w : S \to \{0,1\}$ and $w' : S' \to \{0,1\}$ are equivalent, which we denote by $w \equiv w'$, if for each $i \in \{1, \ldots, |S|\}$, $w(ord_S(i)) = w(ord_{S'}(i))$. Note that if $S = S'$ then $w \equiv w'$ if and only if $w = w'$. Let $n = (1 + \lceil \log m \rceil) \cdot m$. Let $I_1, \ldots, I_m$ be the sequence of subsets of $[n]$ such that for each $i \in [m]$, $I_i = \{(i-1) \cdot (1 + \lceil \log m \rceil), \ldots, i \cdot (1 + \lceil \log m \rceil)\}$. In other words, $I_1, \ldots, I_m$ is a partition of the set $[n]$ into m consecutive intervals of equal size. We say that $I_1, \ldots, I_m$ is the *uniform interval partition* of $[n]$. Let $I_{i_1}, I_{i_2}, \ldots, I_{i_k}$ be intervals which are split by π. Let $(I^L_{j_1}, I^R_{j_1}), \ldots, (I^L_{j_l}, I^R_{j_l})$ be pairs of intervals such that for each $r \in \{1, \ldots, l\}$, $\pi(I^L_{j_r}) \subseteq L$ and $\pi(I^R_{j_r}) \subseteq R$. Let $a_1, \ldots, a_k$, $b_1, \ldots, b_k$, $c_1, \ldots, c_l$ and $d_1, \ldots, d_l$ be $2k + 2l$ distinct binary strings with domain $\{1, \ldots, \lceil \log m \rceil + 1\}$ such that the following conditions are satisfied for each $r \in \{1, \ldots, k\}$.
1. $a_r|_{\pi(I_{i_r}) \cap L}$ is even and $a_r|_{\pi(I_{i_r}) \cap R}$ is even.
2. $b_r|_{\pi(I_{i_r}) \cap L}$ is odd and $b_r|_{\pi(I_{i_r}) \cap R}$ is odd.

For each $(k+l)$-tuple of bits $x_1, \ldots, x_k, y_1, \ldots, y_l$, select a string $w[x_1, \ldots, x_k, y_1, \ldots, y_l]$ in $\{0,1\}^n$ satisfying the following properties for each $r \in \{1, \ldots, k\}$ and $s \in \{1, \ldots, l\}$.
1. If $x_r = 0$ then $w[x_1, \ldots, x_k, y_1, \ldots, y_l]|_{I_{i_r}} \equiv a_r$

2. If $x_r = 1$ then $w[x_1, \ldots, x_k, y_1, \ldots, y_l]|_{I_{i_r}} \equiv b_r$
3. If $y_s = 0$ then $w[x_1 \ldots, x_k, y_1, \ldots, y_l]|_{I^L_{j_s}} \equiv c_s$ and $w[x_1, \ldots, x_k, y_1, \ldots, y_l]|_{I^R_{j_s}} \equiv d_s$
4. If $y_s = 1$ then $w[x_1 \ldots, x_k, y_1, \ldots, y_l]|_{I^L_{j_s}} \equiv d_s$ and $w[x_1, \ldots, x_k, y_1, \ldots, y_l]|_{I^R_{j_s}} \equiv c_s$

We let $F^{k,l}_\pi$ be the set obtained by splitting each string $w[x_1, \ldots, x_k, y_1, \ldots, y_l]$ into a left part and a right part.

$$F^{k,l}_\pi = \{ \; (\; w[x_1, \ldots, x_k, y_1, \ldots, y_l]|_L, \; w[x_1, \ldots, x_k, y_1, \ldots, y_l]|_R \;) \; \mid \; x_r, y_s \in \{0,1\}\} \quad (8)$$

▶ **Theorem 16.** *The set $F^{k,l}_\pi$ defined in Equation 8 is a fooling set for $\pi(B(n))$ of size 2^{k+l}.*

We note that for each permutation π, there exists an $\alpha \leq m/4$ such that the fooling set $F_\pi = F^{\alpha, m/4-\alpha}_\pi$ is well defined. Therefore, by Theorem 16, we have that

$$|F_\pi| \geq 2^{m/4} = 2^{\Omega(n/\log n)}.$$

5 Non-Linear Lower Bounds for Read-Once Circuits Excluding a Minor

In this section we show that exponential lower bounds for the symmetric non-deterministic complexity of a function $f_n : \{0,1\}^n \to \{0,1\}$ imply super-linear lower bounds for the size of read-once circuits excluding a fixed graph H as a minor. We start by establishing a connection between the symmetric-NSC of a circuit, and its pathwidth. More precisely, we will show that any function that can be computed by a read-once circuit of pathwidth k has symmetric-NSC at most $2^k \cdot |\mathcal{C}|$.

▶ **Definition 17** (Path Decomposition)**.** A path decomposition of a circuit $\mathcal{C} = (V, E, \mathbf{g})$ is a sequence $\mathcal{P} = (B_1, B_2, \ldots, B_m)$ of subsets of vertices of G satisfying the following properties.

(i) $V = \bigcup_{i=1}^n B_i$.
(ii) For each $i, j, k \in \mathbb{N}$ with $i < j < k$, $B_i \cap B_k \subset B_j$.
(iii) For each edge $(u, v) \in E$ there is an j such that $\{u, v\} \subseteq B_j$.

The sets B_i are the bags of the decomposition. The path-width of $\mathcal{P}$ is defined as the size of its largest bag minus one. In other words $\boldsymbol{pw}(G, \mathcal{P}) = \max_i\{|B_i|\} - 1$. The pathwidth of a graph G is defined as $\boldsymbol{pw}(G) = \min_{\mathcal{P}} \boldsymbol{pw}(G, \mathcal{P})$ where $\mathcal{P}$ ranges over all path decompositions of G. Let $\mathcal{C}$ be a read-once circuit and $\mathcal{P} = (B_1, B_2, \ldots, B_m)$ be a path decomposition of $\mathcal{C}$. If v is a vertex of $\mathcal{C}$ then we let $\mathit{first}(v, \mathcal{P})$ denote the smallest i such that $v \in B_i$.

▶ **Theorem 18.** *Let $\mathcal{C}$ be a read-once circuit and $\mathcal{P} = (B_1, B_2, \ldots, B_m)$ be a path decomposition of $\mathcal{C}$ of width w. Let $x_1 x_2 \ldots x_n$ be an ordering of the variables of $\mathcal{C}$ such that $\mathit{first}(x_i, \mathcal{P}) < \mathit{first}(x_{i+1}, \mathcal{P})$ for each $i \in \{1, \ldots, n-1\}$. Then for each $b \in \Sigma$, one can construct a NFA on $|\Sigma|^{O(w)} \cdot m$ states accepting the following language.*

$$\mathcal{L}(\mathcal{C}, b) = \{a_1 a_2 \ldots a_n \in \Sigma^n \mid C(a_1 a_2 \ldots a_n) = b\}.$$

We note that any read-once circuit $\mathcal{C}$ of pathwidth k has a decomposition of width k with $O(|\mathcal{C}|)$ bags[3]. Therefore, as a corollary of Theorem 18 and Theorem 14 we have a trade-off between the size of a circuit and its pathwidth.

[3] Any graph G of pathwidth w has a path decomposition of width w with $|G|$ bags.

▶ **Theorem 19.** *Let $f_n : \{0,1\}^n \to \{0,1\}$ be a function of symmetric-NSC $snsc(f_n)$. Then for any read-once circuit computing f_n, the following inequality is satisfied.*

$$\boldsymbol{pw}(\mathcal{C}) + \log|\mathcal{C}| \geq \log snsc(f_n). \tag{9}$$

It is well known that the pathwidth of any graph is greater than its treewidth by at most a multiplicative logarithmic factor [5]. In other words, the following relation between the pathwidth and treewidth of a circuit (graph) can be verified: $\boldsymbol{pw}(\mathcal{C}) \leq \boldsymbol{tw}(\mathcal{C}) \cdot O(\log|\mathcal{C}|)$. Therefore, stated in terms of treewidth, Equation 9 can be rewritten as follows.

$$\boldsymbol{tw}(\mathcal{C}) \cdot \log|\mathcal{C}| + \log|\mathcal{C}| \geq \Omega(\log snsc(f_n)). \tag{10}$$

▶ **Theorem 20** ([3],[14])**.** *For any fixed graph H, every H-minor-free graph G with s vertices has treewidth at most $O(\sqrt{s})$.*

Therefore, combining Equation 10 with Theorem 20 we have the following theorem. We say that a circuit $\mathcal{C}$ is H-minor-free if its underlying undirected graph is H-minor-free.

▶ **Theorem 21.** *Let $\mathcal{C}$ be an H-minor-free, read-once circuit computing a Boolean function $f : \{0,1\}^n \to \{0,1\}$. Then $|\mathcal{C}| \geq \Omega\left(\left(\frac{\log snsc(f_n)}{\log n}\right)^2\right)$.*

Finally, as a corollary of Theorem 21, Theorem 13 and Theorem 14 we have that the triangle-freeness function Δ_n, the triangle-parity function $\bigoplus \text{Clique}_{3,n}$ and the even element distinctness function τ_n require H-minor-free read-once circuits of near quadratic size.

▶ **Corollary 22.** *Let $\mathcal{C}$, $\mathcal{C}'$ and $\mathcal{C}''$ be H-minor-free, read-once circuits computing the triangle-freeness function Δ_n, the triangle-parity function $\bigoplus \text{Clique}_{3,n}$ and the even element distinctness function τ_n respectively. Then $|\mathcal{C}| \geq \Omega(\frac{n^2}{\log^2 n})$, $|\mathcal{C}'| \geq \Omega(\frac{n^2}{\log^2 n})$, and $|\mathcal{C}''| \geq \Omega(\frac{n^2}{\log^4 n})$.*

Acknowledgements. The author would like to thank Pavel Pudlák for interesting discussions on circuit lower-bounds, Pavel Hrubeš for pointing me out to reference [20], Michal Koucký and Bruno Loff for useful feedback during a seminar presentation of this work, and anonymous referees for valuable comments.

References

1 Michael Alekhnovich and Alexander A Razborov. Satisfiability, branch-width and Tseitin tautologies. In *Proc. of the 43rd Symposium on Foundations of Computer Science*, pages 593–603, 2002.

2 Eric Allender, Shiteng Chen, Tiancheng Lou, Periklis A. Papakonstantinou, and Bangsheng Tang. Width-parametrized SAT: Time–space tradeoffs. *Theory of Computing*, 10(12):297–339, 2014. doi:10.4086/toc.2014.v010a012.

3 Noga Alon, Paul Seymour, and Robin Thomas. A separator theorem for graphs with an excluded minor and its applications. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 293–299. ACM, 1990.

4 Norbert Blum. A boolean function requiring 3n network size. *Theoretical Computer Science*, 28(3):337–345, 1983.

5 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998.

6 Elizabeth Broering and Satyanarayana V Lokam. Width-based algorithms for SAT and CIRCUIT-SAT. In *Theory and Applications of Satisfiability Testing*, pages 162–171. Springer, 2004.

7 Chris Calabro. A lower bound on the size of series-parallel graphs dense in long paths. *Electronic Colloquium on Computational Complexity (ECCC)*, 15(110), 2008.

8 Evgeny Demenkov and Alexander S Kulikov. An elementary proof of a $3n - o(n)$ lower bound on the circuit complexity of affine dispersers. In *Mathematical Foundations of Computer Science 2011*, pages 256–265. Springer, 2011.

9 Pavol Duris, Juraj Hromkovic, Stasys Jukna, Martin Sauerhoff, and Georg Schnitger. On multi-partition communication complexity. *Information and Computation*, 194(1):49–75, 2004.

10 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.

11 Anna Gál and Jing-Tang Jang. A generalization of spira's theorem and circuits with small segregators or separators. In *Theory and Practice of Computer Science (SOFSEM 2012)*, pages 264–276. Springer, 2012.

12 Konstantinos Georgiou and Periklis A Papakonstantinou. Complexity and algorithms for well-structured k-sat instances. In *Proc. of the 11th International Conference on Theory and Applications of Satisfiability Testing*, pages 105–118. Springer, 2008.

13 Ian Glaister and Jeffrey Shallit. A lower bound technique for the size of nondeterministic finite automata. *Information Processing Letters*, 59(2):75–77, 1996.

14 Martin Grohe. Local tree-width, excluded minors, and approximation algorithms. *Combinatorica*, 23(4):613–632, 2003.

15 Johan Håstad. The shrinkage exponent of de morgan formulas is 2. *SIAM Journal on Computing*, 27(1):48–64, 1998.

16 Jing He, Hongyu Liang, and Jayalal MN Sarma. Limiting negations in bounded treewidth and upward planar circuits. In *Mathematical Foundations of Computer Science 2010*, pages 417–428. Springer, 2010.

17 Juraj Hromkovič. Communication complexity and lower bounds on multilective computations. *RAIRO-Theoretical Informatics and Applications*, 33(02):193–212, 1999.

18 Kazuo Iwama and Hiroki Morizumi. An explicit lower bound of 5n- o (n) for boolean circuits. In *Mathematical foundations of computer science 2002*, pages 353–364. Springer, 2002.

19 Maurice Jansen and Jayalal Sarma. Balancing bounded treewidth circuits. In *Computer Science – Theory and Applications*, pages 228–239. Springer, 2010.

20 Stasys Jukna. *Boolean function complexity: advances and frontiers*, volume 27. Springer Science & Business Media, 2012.

21 Oded Lachish and Ran Raz. Explicit lower bound of 4.5 n-o (n) for boolena circuits. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 399–408. ACM, 2001.

22 Richard J Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM journal on computing*, 9(3):615–627, 1980.

23 Igor L Markov and Yaoyun Shi. Constant-degree graph expansions that preserve treewidth. *Algorithmica*, 59(4):461–470, 2011.

24 Nečiporuk. On a Boolean function. *Soviet Math. Dokl.*, 7(4):999–1000, 1966.

25 Ramamohan Paturi and Pavel Pudlák. Circuit lower bounds and linear codes. *Journal of Mathematical Sciences*, 134(5):2425–2434, 2006.

26 Pavel Pudlák. The hierarchy of boolean circuits. *Computers and artificial intelligence*, 6(5):449–468, 1987.

27 Neil Robertson and Paul D Seymour. Graph minors. xiii. the disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995.

28 Rahul Santhanam and Srikanth Srinivasan. On the limits of sparsification. In *Automata, Languages, and Programming*, pages 774–785. Springer, 2012.

29 John E. Savage. Planar circuit complexity and the performance of VLSI algorithms. In *INRIA Report* 77 (1981). *Also in VLSI Systems and Computations*, pages 61–67. Computer Science Press Rockwille MD, 1981.

30 György Turán. On the complexity of planar boolean circuits. *Computational Complexity*, 5(1):24–42, 1995.

31 Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In *6th Symposium on Mathematical Foundations of Computer Science*, pages 162–176, 1977.

On Space Efficiency of Algorithms Working on Structural Decompositions of Graphs*

Michał Pilipczuk[1] and Marcin Wrochna[2]

1 **Institute of Informatics, University of Warsaw, Warsaw, Poland**
michal.pilipczuk@mimuw.edu.pl
2 **Institute of Informatics, University of Warsaw, Warsaw, Poland**
m.wrochna@mimuw.edu.pl

Abstract

Dynamic programming on path and tree decompositions of graphs is a technique that is ubiquitous in the field of parameterized and exponential-time algorithms. However, one of its drawbacks is that the space usage is exponential in the decomposition's width. Following the work of Allender et al. [Theory of Computing, '14], we investigate whether this space complexity explosion is unavoidable. Using the idea of reparameterization of Cai and Juedes [J. Comput. Syst. Sci., '03], we prove that the question is closely related to a conjecture that the Longest Common Subsequence problem parameterized by the number of input strings does not admit an algorithm that simultaneously uses **XP** time and **FPT** space. Moreover, we complete the complexity landscape sketched for pathwidth and treewidth by Allender et al. by considering the parameter *tree-depth*. We prove that computations on tree-depth decompositions correspond to a model of non-deterministic machines that work in polynomial time and logarithmic space, with access to an auxiliary stack of maximum height equal to the decomposition's depth. Together with the results of Allender et al., this describes a hierarchy of complexity classes for polynomial-time non-deterministic machines with different restrictions on the access to working space, which mirrors the classic relations between treewidth, pathwidth, and tree-depth.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.2.3 Tradeoffs between Complexity Measures, G.2.2 Graph Theory

Keywords and phrases tree decomposition, LCS, tree-depth, NAuxSA, Savitch's theorem

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.57

1 Introduction

Treewidth is a parameter that measures how easily a graph can be decomposed into a tree-like structure, called a *tree decomposition*. While initially introduced by Robertson and Seymour in their Graph Minors project [41], treewidth has found numerous applications in the field of algorithms, because many problems that are intractable on general graphs, become efficiently solvable on graphs of small treewidth. Theorems of Courcelle [14] and of Arnborg et al. [5] explain that every problem expressible in Monadic Second Order logic can be solved in time $f(s)\cdot n$ on graphs of treewidth s and size n, for some function f. While f can be non-elementary in general, for many classic problems, like Vertex Cover, 3Coloring, or Dominating

* Research supported by Polish National Science Centre grant DEC-2013/11/D/ST6/03073. During the work on these results, Michał Pilipczuk held a post-doc position at Warsaw Center of Mathematics and Computer Science, and was supported by Foundation for Polish Science (FNP) via START stipend programme.

© Michał Pilipczuk and Marcin Wrochna;
licensed under Creative Commons License CC-BY

SET, the natural dynamic programming approach yields a running time of $\mathcal{O}(c^s \cdot n)$ for a small constant c. Dynamic programming procedures working on tree decompositions are important for applications, as they often serve as critical subroutines in more complex techniques, e.g., subexponential parameterized algorithms derived using *bidimensionality* [16], or approximation schemes obtained via Baker's approach [7]. Algorithms working on tree decompositions are usually analyzed in the paradigm of *parameterized complexity*, where treewidth is the considered parameter. We refer to textbooks [15, 17, 22] for a broad introduction, and to a recent survey of Langer et al. [30] for more specific results.

A certain limitation of dynamic programming on a tree decomposition is that it uses space exponential in its width, which is often a prohibitive factor in practical applications. Therefore, recently there is much focus on reducing the space complexity of exponential-time algorithms to polynomial, even at the cost of slightly worsening the time complexity [6, 9, 23, 24, 34, 37]. Here, the usage of algebraic tools proved to be an extremely useful approach. Unfortunately, algorithms working on treewidth remain a family where virtually no progress has been achieved in this matter. Therefore, a natural question arises: Can we reduce the space complexity of algorithms working on tree decompositions while keeping (or moderately worsening) their time complexity? This was first asked explicitly by Lokshtanov et al. [33], who sketched how a simple tradeoff achieves polynomial space complexity while increasing the time complexity to $2^{\mathcal{O}(s^2)} + \mathcal{O}(n^2)$. The question was reiterated later by Langer et al. [30].

Following early completeness results of Monien and Sudborough [36] on bandwidth-constrained problems and of Gottlob et al. [26] on conjunctive queries of bounded treewidth, Allender et al. [4] recently initiated the systematic study of satisfaction complexity in variable path- and treewidth. Essentially, they observe that CSP-like problems—say, 3COLORING for concreteness[1]—when limited to instances of small treewidth or pathwidth, are complete for certain complexity classes under logspace reductions. More precisely, when the input graph is equipped with a path decomposition of width at most $s(n) \geq \log n$, for some fixed function s of the input size, then 3COLORING (denoted in this case as pw-3COLORING$[s]$) is complete for the class $\mathbf{N}[\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})}]$: problems that admit non-deterministic algorithms working simultaneously in polynomial time and space $\mathcal{O}(s(\text{poly}(n))$. Similarly tw-3COLORING$[s]$, where $s(n)$ bounds the width of a given tree decomposition, is complete for the class $\mathbf{NAuxPDA}[\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})}]$; the difference with $\mathbf{N}[\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})}]$ is that the algorithm can use an auxiliary push-down of unlimited size, to which read/write access is only from the top. Allender et al. also describe the class in terms of *semi-unbounded fan-in* (SAC) circuits. They assume $s(n) = \log^k n$, but the proof works in the more general setting given below.

▶ **Theorem 1** ([4]). *Let $s(n) \geq \log n$ be a nice function[2].*
Then pw-3COLORING$[s]$ *is complete for* $\mathbf{N}[\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})}]$ *under logspace reductions, whereas* tw-3COLORING$[s]$ *is complete for* $\mathbf{NAuxPDA}[\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})}]$ *under logspace reductions.*

Thus, the feasibility of various space-time tradeoffs when working on tree/path decompositions is equivalent to inclusions of corresponding complexity classes. For instance (assuming for conciseness $\forall_c\ s(n^c) = \mathcal{O}(s(n))$, e.g., $s(n) = \log^k n$ for $k \geq 1$), pw-3COLORING$[s]$ is solvable:

[1] Allender et al. use SAT parameterized by treewidth/pathwidth of its primal graph as an exemplary problem, but SAT and 3COLORING can be easily seen to be equivalent under logspace reductions; see Lemma 10. In this paper, we prefer to use 3COLORING as an exemplary hard CSP-like problem.
[2] By a *nice* function we mean a function s that is constructible and such that $s(n)/\lg n$ is non-decreasing.

- in time $2^{o(s(n)\log n)}$ and space $2^{o(s(n))}$ if and only if $\mathbf{N}[\underset{time}{\text{poly}}, \underset{space}{s}] \subseteq \mathbf{D}[\underset{time}{2^{o(s\cdot\log)}}, \underset{space}{2^{o(s)}}]$;
- in time $2^{\mathcal{O}(s(n))}$ and space $\text{poly}(n)$ if and only if $\mathbf{N}[\underset{time}{\text{poly}}, \underset{space}{s}] \subseteq \mathbf{D}[\underset{time}{2^{\mathcal{O}(s)}}, \underset{space}{\text{poly}}]$.

Similar statements can be inferred for treewidth. In contrast, the best known determinization for $\mathbf{N}[\underset{time}{\text{poly}}, \underset{space}{s}]$ come from a brute-force approach or Savitch's theorem [43], yielding respectively (for $s(n) \geq \lg n$) $\mathbf{D}[\underset{time}{2^{\mathcal{O}(s)}}] = \mathbf{D}[\underset{time}{2^{\mathcal{O}(s)}}, \underset{space}{2^{\mathcal{O}(s)}}]$ and $\mathbf{D}[\underset{space}{s\cdot\log}] = \mathbf{D}[\underset{time}{2^{\mathcal{O}(s\cdot\log n)}}, \underset{space}{s\cdot\log}]$.

In this manner, Allender et al. conclude that, intuitively speaking, achieving better time-space tradeoffs for algorithms working on path and tree decompositions of small width would require developing a general technique of improving upon the tradeoff of Savitch. As Lipton phrased it, "one of the biggest embarrassments of complexity theory is the fact that Savitch's theorem has not been improved [...]. Nor has anyone proved that it is tight" [31].

Allender et al. argue that such an improvement would contradict certain rescaled variants of known conjectures about the containment of time- and space-constrained classes, in particular the assumption that $\mathbf{NL} \not\subseteq \mathbf{SC}$; we refer to [4] for details. We consider the study of Allender et al. not as a definite answer in the topic, but rather as an invitation to a further investigation of the introduced conjectures.

Our Contribution. In the Longest Common Subsequence problem (LCS), we are given an alphabet Σ and k strings over Σ, and ask for the longest sequence of symbols that appears as a subsequence in each input string. The applicability of the LCS problem in, e.g., computational biology, motivated many to search for faster, more space-efficient algorithms, as the classical dynamic programming solution, running in time and space $\mathcal{O}(n^k)$ (where n is the length of each string) is often far from practical. From the point of view of parameterized complexity, LCS parameterized by k is W[t]-hard for every level t [11], remains W[1]-hard for a fixed-sized alphabet [39], and is W[1]-complete when parameterized jointly by k and ℓ, the target length of the subsequence [27]. In a recent breakthrough, Abboud et al. [1] proved that the existence of an algorithm with running time $\mathcal{O}(n^{k-\varepsilon})$, for any $\varepsilon > 0$, would contradict the Strong Exponential Time Hypothesis. As far as the space complexity is concerned, only modest progress has been achieved: The best known result, by Barsky et al. [8], improves the space complexity to $\mathcal{O}(n^{k-1})$. This motivates us to formulate the following conjecture.

▶ **Conjecture 2.** *There is no algorithm for LCS that works in time $n^{f(k)}$ and space $f(k)\text{poly}(n)$ for a computable function f, where k is the number of input strings and n their total length.*

Quite surprisingly, we show that Conjecture 2 is closely related to the question of time-space tradeoffs for algorithms working on small pathwidth, as detailed in Theorem 15. There, the conjecture is sandwiched between a weaker statement that it is impossible to achieve subexponential space while keeping single exponential time complexity, and a stronger statement that this holds even if we allow the time complexity exponent to increase by an arbitrarily slowly growing function of the width. To prove this, we use a completeness result of Elberfeld et al. [21] for LCS, which allows to formulate Conjecture 2 as an equivalent statement in parameterized complexity about the impossibility of determinization results improving upon Savitch's theorem. Using the ideas of Cai and Juedes [12] connecting subexponential complexity to fixed-parameter tractability, we consider a reparameterized version of pw-3Coloring. This allows us to compare questions concerning time-space tradeoffs for pw-3Coloring and determinization of $\mathbf{N}[\underset{time}{t}, \underset{space}{s}]$ classes to those concerning parameterized classes and the complexity of LCS. In particular, we show that Conjecture 2

implies $\mathbf{NL} \not\subseteq \mathbf{D}[\underset{time}{\text{poly}}, \underset{space}{\text{poly}\log}]$ (the latter class being usually called **SC**) and is implied by a rescaled version of the following stronger variant: $\mathbf{NL} \not\subseteq \mathbf{D}[\underset{time}{2^{o(\log^2 n)}}, \underset{space}{n^{o(1)}}]$.

In the second part of this work, we complement the findings of Allender et al. [4] by considering the graph parameter *tree-depth*. Tree-depth of a graph is lower bounded by its pathwidth and upper bounded by its treewidth times $\lg n$. Our motivation for considering this parameter is two-fold. First, recent advances have uncovered a wide range of topics where tree-depth appears naturally. For instance, it plays an important role in the theory of sparse graphs of Nešetřil and Ossona de Mendez [38], it is the key factor in classifiyng homomorphism problems that can be solved in logspace [13], and characterizes classes of graphs where the expressive power of First-Order and Monadic Second-Order logic coincides [19]. It was rediscovered several times under different names: *minimum elimination tree height* [40], *ordered chromatic number* [28], *vertex ranking* [10], or the maximum number of introduce nodes on a root-to-leaf path of a tree decomposition [24].

Second, algorithms working on tree-depth decompositions model generic exponential-time Divide&Conquer algorithms. In this approach, after finding a small, balanced separator S in the graph, the algorithm tries all possible ways a solution can interact with S, and solves connected components of $G - S$ recursively. This naturally gives rise to a tree-depth decomposition of the graph, where S is placed on top, and decompositions of the components of $G - S$ are attached below it as subtrees. The maximum total number of separator vertices handled at any moment in the recursion corresponds to the depth of the decomposition. Thus, many classic Divide&Conquer algorithms, including the ones derived for planar graphs using the Lipton-Tarjan separator theorem [32], can be reinterpreted as first building a tree-depth decomposition of the graph using a separator theorem, and then running the algorithm on it.

Most importantly for us, recursive algorithms working on tree-depth decompositions run in polynomial space. For instance, such an algorithm for 3Coloring on a tree-depth decomposition of depth s runs in time $3^s \cdot \text{poly}(n)$ and space $\mathcal{O}(s + \log n)$ (see Lemma 20), which places td-3Coloring$[s]$ in $\mathbf{D}[\underset{time}{2^{\mathcal{O}(s)}\text{poly}}, \underset{space}{s + \log}] = \mathbf{D}[\underset{space}{s + \log n}]$. This is immediate for CSP-like problems like 3Coloring, but recently Fürer and Yu [24] showed that algebraic transforms can be used to reduce the space usage to polynomial in n also for other problems, like counting perfect matchings or dominating sets. We describe how this approach gives an $3^s \cdot \text{poly}(n)$-time $\text{poly}(n)$-space algorithm for Dominating Set in more detail in the full version of the article. This means that the reduction of space complexity that is conjectured to be impossible for treewidth and pathwidth, actually is possible for tree-depth. Therefore, we believe that it is useful to study the computation model standing behind low tree-depth decompositions, in order to understand how it differs from the models for treewidth and pathwidth.

Consequently, mirroring Theorem 1, we prove that computations on tree-depth decompositions exactly correspond to the class $\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{\log}, \underset{height}{s}]$: problems that can be decided by a non-deterministic Turing Machine that uses polynomial time and logarithmic space, but also has access to an auxiliary stack of maximum height s. The stack can be freely read by the machine, just as the input tape, but write access is only via `push`/`pop` operations.

▶ **Theorem 3.** *Let $s(n) \geq \log^2 n$ be a nice function. Then* td-3Coloring$[s]$ *is complete for* $\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{\log}, \underset{height}{s(\text{poly})}]$ *under logspace reductions.*

Thus, computations on tree-depth and path decompositions differ by the access restrictions to $\mathcal{O}(s)$ space used by the machine. While for pathwidth this space can be accessed freely, for tree-depth all except an $\mathcal{O}(\log n)$ working buffer has to be organized in a stack.

The proof of Theorem 3 largely follows the approach of Akatov and Gottlob [3], who proved a different completeness result for the class $\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{\log}, \underset{height}{\log^2}]$, which they call $\mathbf{DC}^1$. The main idea is to regularize the run of the machine so that the push-pop tree has the rigid shape of a full binary tree. Then we can use this concrete structure to "wrap around" gadgets encoding an accepting run of a regularized NAuxSA machine. However, the motivation in the work of Akatov and Gottlob was answering conjunctive queries in a hypergraph by exploiting a kind of balanced decomposition, and hence the problem proven to be complete for $\mathbf{DC}^1$ is a quite general and expressive problem originating in database motivations; see [2, 3] for details. In our setting, in order to get a reduction to 3Coloring, we need to work more to encode an accepting run. In particular, to encode each part of the computation where no push or pop is performed, instead of producing a single atom in a conjunctive query, we use computation gadgets that originate in Cook's proof of the NP-completeness of SAT. The assumption that the computation has a polynomial number of steps is essential here for bounding the tree-depth of each such gadget. This way, Theorem 3 presents a more natural complete problem for $\mathbf{DC}^1$.

Another difference is that Theorem 3 works for any well-behaved function $s(n) \geq \log^2 n$, as opposed to the bound $s(n) = \log^2 n$ inherent to the problem considered by Akatov and Gottlob. For this, the crucial new idea is to increase the working space of the machine to $s(n)/\log n$ in order to be able to perform regularization – a move that looks dangerous at first glance, but turns out not to increase the expressive power of the computation model. This proves the following interesting by-product of our work.

▶ **Theorem 4.** *Let $s(n) \geq \log^2 n$ be a nice function. Then*

$$\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{\log}, \underset{height}{s(\text{poly})}] = \mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})/\log}, \underset{height}{s(\text{poly})}].$$

The following determinization follows from the $\mathcal{O}(s + \log n)$ algorithm for td-3Coloring.

▶ **Theorem 5.** *Let $s(n) \geq \log^2(n)$ be a nice function. Then*

$$\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{\log}, \underset{height}{s(\text{poly})}] \subseteq \mathbf{D}[\underset{space}{s(\text{poly})}].$$

Theorem 5 for $s(n) = \log^2 n$ also follows from the work of Akatov and Gottlob [3]. Observe that now the justification for the assumption $s(n) \geq \log^2 n$ becomes apparent: for, say, $s(n) = \log n$, the theorem would state that $\mathbf{L} = \mathbf{NL}$, a highly unexpected outcome.

We find Theorem 5 interesting, because a naive simulation of the whole configuration space for NAuxSA would require space exponential in s. It appears, however, that the exponential blow-up of the space complexity can be avoided. We do not see any significantly simpler way to prove this result other than going through the td-3Coloring$[s]$ problem, and hence it seems that the tree-depth view gives a valuable insight into the computation model of NAuxSA. The classic relations between treewidth, pathwidth and tree-depth are, through completeness results, mirrored in a hierarchy between NAuxPDA, N, and NAuxSA classes, as detailed in the concluding section. In particular, this answers a question of Akatov and Gottlob [2, 3] about the relation of $\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{\log}, \underset{height}{\text{poly}\log}]$ to other classes in **NP**.

Finally, using Theorem 3 we also give an alternative view on NAuxSA computations using alternating Turing machines in Theorem 21, answering another question of Akatov and Gottlob. From this point of view, Theorem 5 is immediate.

2 Preliminaries

Reductions and complexity classes. For two languages P, Q, we write $P \leq_L Q$ when P is logspace reducible to Q. Most of the complexity classes we consider are closed under logspace reductions. Because we handle various measures of complexity and compare a wide array of classes that bound two measures simultaneously, we introduce the following notation. A complexity class is first described by the machine model: **D**, **N**, **A** denote deterministic, non-deterministic, and alternating (see [42]) Turing machines, respectively. Then bounds on complexity measures are described (up to constant factors) as a list of functions with the measure's name underneath. All functions except the symbol f (which we reserve for classes in parameterized complexity) are functions of the input size n. For example, $\mathbf{N}[\underset{time}{t}, \underset{space}{s}]$ is the class often known as $\mathrm{NTiSp}(t(n), s(n))$. We write $\mathrm{poly}(n)$ for $n^{\mathcal{O}(1)}$, e.g., $\mathbf{D}[\underset{time}{\mathrm{poly}}] = \mathbf{P}$. An auxiliary push-down or stack is denoted as AuxPDA or AuxSA, respectively: the difference is that a push-down can only be read at the top, while a stack can be read just as a tape (both can be written to only by pushing and popping symbols at the top), see e.g. [44]. The measure named *height* is the maximum height of the push-down or stack.

We write lg for the logarithm with base 2 and log when the base is irrelevant. We say a function $s : \mathbb{N} \to \mathbb{N}$ is *constructible* if there is a Turing machine which given a number n in unary outputs $s(n)$ in unary using logarithmic space; in particular, this implies $s(n) \leq \mathrm{poly}(n)$. A function s is *nice* if it is constructible and $\frac{s(n)}{\lg n}$ is non-decreasing. For simplicity, we will assume all functions $s : \mathbb{N} \to \mathbb{N}$ describing complexity bounds to be nice.

Note that logspace reductions can blow-up instance sizes polynomially, hence the closure of $\mathbf{N}[\underset{time}{\mathrm{poly}}, \underset{space}{s}]$ under such reductions is $\mathbf{N}[\underset{time}{\mathrm{poly}}, \underset{space}{s(\mathrm{poly})}]$, for example. These are equal for functions $s(n)$ such that $s(\mathrm{poly}(n)) \leq \mathcal{O}(s(n))$ (that is, if for every $c > 0$ there is a $d > 0$ such that $s(n^c) \leq d \cdot s(n)$). This includes $\lg^k(n)$ for any $k \geq 1$ and $\lg n \lg\lg n$, for example.

Structural parameters. We recall the definition of tree-depth. See e.g. [15, 41] for definitions of treewidth and pathwidth. For technical reasons, we assume that in all given tree and path decompositions $\mathcal{T}$, $|\mathcal{T}| \leq 2|V(G)|^2$; standard methods allow to prune any decomposition to this size in logspace, see e.g. [29, Lemma 13.1.2]. For conciseness, we will refer to the certifying structures as *decompositions* for all three parameters.

▶ **Definition 6** (tree-depth)**.** A *tree-depth decomposition* of an undirected graph G is a rooted forest $\mathcal{T}$ (disjoint union of rooted trees) together with a bijection μ from the vertices of G to the nodes of $\mathcal{T}$, such that for every edge uv of G, $\mu(u)$ is an ancestor of $\mu(v)$ or vice-versa in $\mathcal{T}$. The *depth* of $\mathcal{T}$ is the largest number of nodes on a path between a root and a leaf. The *tree-depth* of G is the minimum depth over all possible tree-depth decompositions of G.

The following lemma describes well-known inequalities between the three parameters.

▶ **Lemma 7** (♠)**.** [3] *There is a constant $c \in \mathbb{N}$ such that for any graph G, $\mathtt{td}(G) \geq \mathtt{pw}(G) \geq \mathtt{tw}(G) \geq \mathtt{td}(G)/(c \cdot \log |V(G)|)$. Furthermore, each inequality is certified by an algorithm that transforms the respective graph decompositions in logspace.*

For a graph problem, such as 3Coloring, a structural parameter $\pi \in \{\mathtt{td}, \mathtt{pw}, \mathtt{tw}\}$, and a nice function $s : \mathbb{N} \to \mathbb{N}$, we define π-3Coloring$[s]$ to be the decision problem where given

[3] Proofs of statements marked with ♠ are deferred to the full version of the article (in the appendix).

an instance G of 3Coloring and a π-decomposition of G, we ask whether the decomposition has width at most $s(|V(G)|)$ and G is a yes-instance of 3Coloring. The assumption that a decomposition is given on input is to factor away the complexity of finding it, which is a problem not directly relevant to our work. Note that the validity and width/depth of a decomposition given in any natural encoding can easily be checked in logarithmic space.

Observe also that for any $c > 0$, π-3Coloring$[s(n)]$ is equivalent to π-3Coloring$[s(n^c)]$ under logspace reductions. A reduction to π-3Coloring$[s(n^c)]$ is trivial, while the reverse reduction follows easily by padding: adding isolated vertices up to size n^c that do not change the answer nor the value of π. Also, as we assume s to be nice, we have $\frac{s(n)}{\lg n} \leq \frac{s(n^c)}{\lg n^c}$, hence $c \cdot s(n) \leq s(n^c)$ for any $c \geq 1$. This implies that π-3Coloring$[c \cdot s(n)]$ is equivalent to π-3Coloring$[s(n)]$. Thus, the hierarchy of Lemma 7 takes the following form.

▶ **Corollary 8.** *Let $s : \mathbb{N} \to \mathbb{N}$ be a nice function. Then*

$$\text{td-3Coloring}[s] \leq_L \text{pw-3Coloring}[s] \leq_L \text{tw-3Coloring}[s] \leq_L \text{td-3Coloring}[s \cdot \log].$$

Equivalence of problems. A reduction between two graph problems *preserves structural parameters* if for each parameter $\mathtt{tw}, \mathtt{pw}, \mathtt{td}$ and any instance with graph G, a decomposition of G of width/depth at most s can be transformed in logspace into a decomposition of the graph H produced by the reduction of width/depth at most $\mathcal{O}(s)$. Many NP-hardness reductions have this property, in particular those that replace each vertex or edge with a constant-size gadget (see the 'local replacement' and 'component design' methods in Garey and Johnson [25]). For example, 3Coloring and variants of SAT are equivalent in all our theorems, while Vertex Cover or Dominating Set (defined in [25]) are at least as hard.

▶ **Definition 9.** Let ϕ be a CNF formula. The *primal (Gaifman) graph of ϕ* is the graph with a vertex for each variable of ϕ and an edge between every pair of variables that appear together in some clause. The *incidence graph of ϕ* is the bipartite graph with a vertex for each clause and each variable of ϕ, where every clause is adjacent to variables contained in it.

▶ **Lemma 10 (♠).** *The following problems are equivalent under logspace reductions preserving structural parameters:* 3Coloring*, CNF-SAT (with the primal graph), k-SAT (with either the primal or incidence graph) for each $k \geq 3$. Furthermore,* Vertex Cover*,* Independent Set *and* Dominating Set *each admit such a reduction from the above problems.*

Cook's theorem with bounded space. A common element in our reductions is the description of Turing machine computations using CNF formulas, as in Cook's theorem. Already Monien and Sudborough [36] observed that Cook's reduction applied to machines with bounded space yields formulas of bounded width. The difference is that machine's worktape space bound can be much smaller than the input word—access to the read-only input tape has to be implemented differently. To later handle stack machines, we also need to consider a second input tape separately. We informally state the version of Cook's construction we need.

▶ **Lemma 11** (Computation gadget, ♠). *Let M be an NTM over alphabet Σ with two read-only input tapes and one work tape. Given an input word α of length n and integers s, t, h in unary such that $\lg n, \lg h \leq \mathcal{O}(s)$, one can in logspace output a CNF formula such that:*

- *The formula has $\text{poly}(n, t, s, h)$ variables, including named variables $w_1, \ldots, w_{h \cdot \lceil |\Sigma| \rceil}$, $u_1, \ldots, u_{\Theta(s)}$, $v_1, \ldots, v_{\Theta(s)}$, describing: a word $\bar{w}$ and two configurations $\mathbf{u}$, $\mathbf{v}$ of M (up to s symbols of the working tape, heads' positions encoded in binary, and the state).*
- *Any assignment to the named variables can be extended to a satisfying assignment iff M on inputs α and $\bar{w}$ has a run from $\mathbf{u}$ to $\mathbf{v}$, using at most t steps and s space.*
- *The formula's primal graph has pathwidth $\mathcal{O}(s+h)$ and tree-depth $\mathcal{O}(s \cdot \log(n+s+t+h)+h)$.*

3 Connections with Tradeoffs for LCS

In this section we relate Conjecture 2 to statements of varying strength concerning different time-space tradeoffs. The results are summarized in Figure 1.

A *pl-reduction* between parameterized problems is an algorithm that transforms an instance of one problem with parameter k into an equivalent instance of another problem with parameter $k' \leq f(k)$, working in space $f(k)+\mathcal{O}(\log n)$, for some computable f. Following Elberfeld et al. [21] we define[4] $\mathbf{N}[f\mathrm{poly}, f\log]$ as the class of parameterized problems that can be solved in non-deterministic time $f(k)\mathrm{poly}(n)$ and space $f(k)\log(n)$ for some computable function f, where k is the parameter. Deterministic classes $\mathbf{D}[t, s]$ are defined analogously for various expressions t, s. All those mentioned in the article are closed under pl-reductions. We do not use the better known fpt-reductions because $\mathbf{N}[f\mathrm{poly}, f\log]$ is not expected to be closed under them; its closure under fpt-reductions has been called WNL by Guillemot [27].

We use $o_{\mathrm{eff}}(h(n))$ as an effective variant of $o(h(n))$: for $f, h : \mathbb{N} \to \mathbb{N}$ we write $f = o_{\mathrm{eff}}(h)$ if there is a non-decreasing, unbounded, computable function $g(n)$ such that $f = \mathcal{O}(\frac{h}{g})$. The *inverse* of a function f is the function $f^{-1}(n) := \max\{i \mid f(i) \leq n\}$; observe that $f(f^{-1}(n)) \leq n \leq f^{-1}(f(n))$. Conjecture 2 concerns the following parameterized problem.

LCS **Parameter:** k
Input: A finite alphabet Σ, k strings $s_1, s_2, \ldots, s_k$ over Σ, and an integer ℓ.
Question: Is there a common subsequence of $s_1, s_2, \ldots, s_k$ of length at least ℓ?

Elberfeld et al. [21], drawing on the work of Guillemot [27], pinpointed the complexity of LCS, allowing Conjecture 2 to be phrased as a general statement in parameterized complexity.

▶ **Theorem 12** ([21]). *LCS is complete for* $\mathbf{N}[f\mathrm{poly}, f\log]$ *under pl-reductions.*

▶ **Corollary 13.** *Conjecture 2 holds if and only if* $\mathbf{N}[f\mathrm{poly}, f\log] \not\subseteq \mathbf{D}[n^f, f\mathrm{poly}]$.

Similarly as described in the introduction, the best known determinization results can only place $\mathbf{N}[f\mathrm{poly}, f\log]$ in $\mathbf{D}[n^f, n^f]$ (commonly known as **XP**) and $\mathbf{D}[n^{f(k)\cdot\log n}, f(k)\cdot\log^2 n]$.

Following Cai and Juedes [12], to relate parameterized tractability bounds to subexponential bounds, we define a reparameterized version of pw-3Coloring.

pw-3Coloring$^{\log n}$ **Parameter:** $s/\lg n$
Input: A graph G and a path decomposition of G of width s
Question: Is G 3-colorable?

Similarly as in Theorem 1, pathwidth-constrained problems turn out to be complete for non-deterministic computation with simultaneous time and space bounds.

▶ **Theorem 14 (♠).** pw-3Coloring$^{\log n}$ *is complete for* $\mathbf{N}[f\mathrm{poly}, f\log]$ *under pl-reductions.*

Containement follows from a poly-time, $\mathcal{O}(s + \log n)$-space non-deterministic algorithm that proceeds on consecutive bags of the decomposition, guessing each vertex color and remembering only those in the current bag. Completeness is proved with a direct application of Lemma 11 to a problem with parameter k solved in space $\mathcal{O}(f(k)\log n)$, yielding through Lemma 10 a pw-3Coloring instance of width $s = \mathcal{O}(f(k)\log n)$.

Conjecture 2 is thus equivalent to the statement that pw-3Coloring$^{\log n}$ is not in $\mathbf{D}[n^f, f\mathrm{poly}]$, which gives Theorem 15.1. To contrast pathwidth with tree-depth, Lemma 20

[4] In this section, we only use time-space-bounded classes, hence we drop the subscripts for readability.

(introduced later) places td-3Coloring$^{\log n}$ in $\mathbf{D}[n^f, f\log]$, a class known as **XL**. Similarly as in the work of Cai and Juedes [12], we show that also subexponential bounds on the complexity of pw-3Coloring are related to the parameterized complexity of pw-3Coloring$^{\log n}$. This gives the sandwiching of Conjecture 2 between two similar statements in Theorem 15.2, 15.3. An even weaker statement is proved equivalent to $\mathbf{NL} \not\subseteq \mathbf{SC}$ by a simple padding argument in Theorem 15.4. For a somewhat less natural, stronger variant of Conjecture 2, we can show a similar, but exact correspondence in 15.5 (note the quasi-polynomial factor on both sides).

▶ **Theorem 15 (♠).** *Consider deterministic algorithms for* pw-3Coloring *working on instances of size n with a given path decomposition of width s (uniformly for all values of s).*

1. *There is no such algorithm working in time $n^{f(s/\lg n)}$ and space $f(s/\lg n)\mathrm{poly}(n)$ for any computable f if and only if Conjecture 2 holds.*
2. *Assuming Conjecture 2, there is no such algorithm working in time $2^{\mathcal{O}(s)}\mathrm{poly}(n)$ and space $2^{o_{\mathit{eff}}(s)}\mathrm{poly}(n)$.*
3. *If Conjecture 2 fails, then for every unbounded, computable function g, there is such an algorithm working in time $2^{s\cdot g(s)}\mathrm{poly}(n)$ and space $2^{o_{\mathit{eff}}(s)}\mathrm{poly}(n)$.*
4. *There is no such algorithm working in time $2^{\mathcal{O}(s)}\mathrm{poly}(n)$ and space $\mathrm{poly}(s, \log n)$ if and only if $\mathbf{NL} \not\subseteq \mathbf{SC}$.*
5. *There is no such algorithm working in time $2^{o_{\mathit{eff}}(s^2)}n^{\mathcal{O}(\log n)}$ and space $2^{o_{\mathit{eff}}(s)}\mathrm{poly}(n)$ if and only if $\mathbf{N}[f\mathrm{poly}, f\log] \not\subseteq \mathbf{D}[n^{f+\log}, f\mathrm{poly}]$.*

Proof of Theorem 15(2). Suppose to the contrary that pw-3Coloring can be solved in time $2^{\mathcal{O}(s)}\mathrm{poly}(n)$ and space $2^{o_{\mathrm{eff}}(s)}\mathrm{poly}(n)$. We show that $\mathbf{N}[f\mathrm{poly}, f\log] \subseteq \mathbf{D}[n^f, f\mathrm{poly}]$, contradicting Conjecture 2. The assumption implies that pw-3Coloring$^{\log n}$ can be solved in time $2^{\mathcal{O}(k\cdot\lg n)} = n^{\mathcal{O}(k)}$ and space $2^{k\cdot\lg n/g(k\cdot\lg n)}$ for some unbounded and non-decreasing computable function $g(\cdot)$. If $k \leq g(k\cdot\lg n)$, then the bound on space is bounded by n. Otherwise, if $k > g(k\lg n) \geq g(\lg n)$, then $n \leq 2^{g^{-1}(k)}$. In this case the bound on space is bounded by a computable function of k, namely $2^{k\cdot g^{-1}(k)}$. Hence in each case, the same algorithm solves pw-3Coloring$^{\log n}$ in time $n^{\mathcal{O}(k)}$ and space $n + 2^{k\cdot g^{-1}(k)}$. By Theorem 14, this implies $\mathbf{N}[f\mathrm{poly}, f\log] \subseteq \mathbf{D}[n^f, f\mathrm{poly}]$. ◀

We summarize the relationships around Conjecture 2 in Figure 1. The weakest statement there is $\mathbf{NL} \not\subseteq \mathbf{SC}$, a widely explored hypothesis in complexity theory. Since Directed (s,t)-Reachability (asking given a directed graph and two nodes s, t, whether is t reachable from s) is an **NL**-complete problem, this is also equivalent to the question of whether this problem can be decided in polynomial time and polylogarithmic space. However, even this weakest statement is not known to be implied by better known conjectures such as the Exponential Time Hypothesis. It seems that the simultaneous requirement on bounding two complexity measures—time and space—has a nature independent of the usual time complexity considerations. Hence, new assumptions may be needed to explore this paradigm, and we hope that Conjecture 2 may serve as a transparent and robust example of such.

In a certain restricted computation model (allowing operations on graph nodes only, not on individual bits), unconditional tight lower bounds have been proved by Edmonds et al. [18]: it is impossible to decide Directed (s,t)-Reachability in time $2^{o(\log^2 n)}$ and space $\mathcal{O}(n^{1-\varepsilon})$ (for any $\varepsilon > 0$), even if randomization is allowed. Essentially all known techniques for solving Directed (s,t)-Reachability are known to be implementable in this model [35] (including DFS, BFS, theorems of Savitch, of Immerman and Szelepcsényi, as well as Reingold's breakthrough), therefore this strongly suggests that no algorithm running in time $2^{o_{\mathrm{eff}}(\log^2 n)}$ and space $n^{o_{\mathrm{eff}}(1)}$ is possible, that is, $\mathbf{NL} \not\subseteq \mathbf{D}[\underset{\mathit{time}}{2^{o_{\mathrm{eff}}(\log^2 n)}}, \underset{\mathit{space}}{n^{o_{\mathrm{eff}}(1)}}]$.

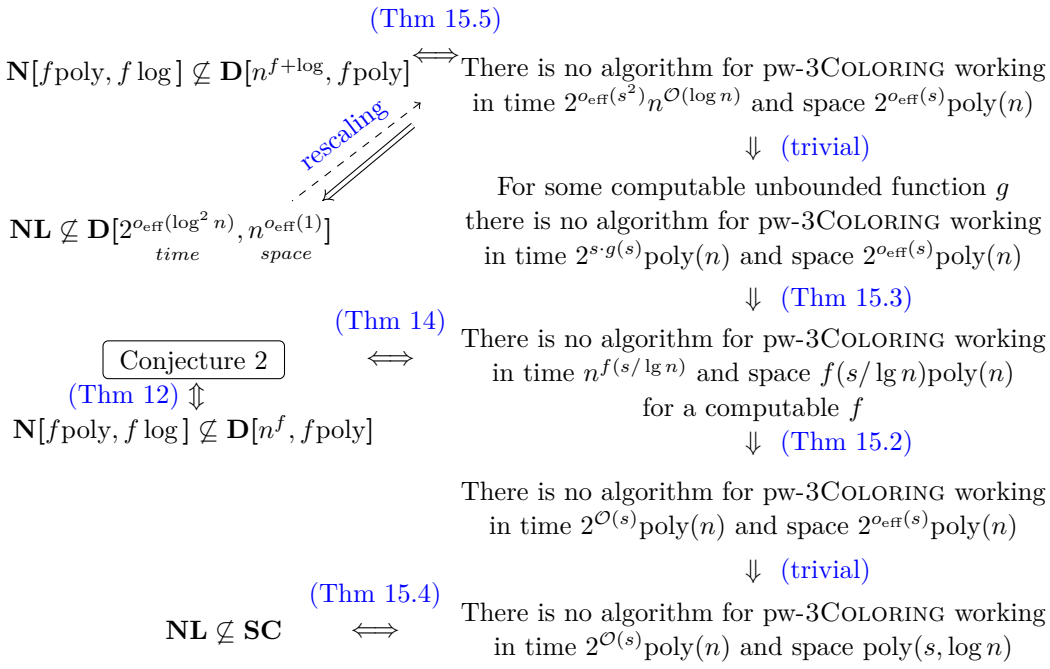


Figure 1 A summary of the relationships between various statements related to Conjecture 2.

By Theorem 1, this is equivalent to saying that pw-3Coloring[log] cannot be solved in these time and space bounds. The strongest statement on Figure 1 is a rescaling of this, that is, it implies $\mathbf{NL} \not\subseteq \mathbf{D}[2^{o_{\mathrm{eff}}(\log^2 n)}_{time}, n^{o_{\mathrm{eff}}(1)}_{space}]$ by a trivial padding argument, but the reverse implication is also probable in the sense that any proof of the latter would likely scale to prove the former. However, it is still possible that an algorithm working in polynomial space refutes the stronger statement even though $\mathbf{NL} \not\subseteq \mathbf{D}[2^{o_{\mathrm{eff}}(\log^2 n)}_{time}, n^{o_{\mathrm{eff}}(1)}_{space}]$.

4 Treedepth

In this section we sketch the proof of Theorem 3. Let $s : \mathbb{N} \to \mathbb{N}$ be a nice function. First, we discuss more precisely the model of machines used to define class $\mathbf{NAuxSA}[\mathrm{poly}, \log, s]_{time\ space\ height}$. The machine has three tapes, each using a fixed, finite alphabet Σ: a read-only input tape, a working tape of length $\mathcal{O}(\log n)$, and a stack tape of length $s(n)$. On each of the tapes there is a head; the transitions of the machine depend on its state and the triple of symbols under the heads. The input tape is read-only. The stack tape can be read but not freely written on; instead, the transitions of the machine may contain instructions `push`σ or `pop`, working naturally. Since s is nice, $s(n) \leq \mathrm{poly}(n)$ so within the working tape the machine can keep track of the current height of the stack and the indices on which the heads are positioned.

We start the proof of Theorem 3 by showing containment, exemplifying how the resources are used. The idea is to perform a depth-first search of the tree-depth decomposition, guessing the color of each entered vertex, pushing it onto the stack and popping it when withdrawing from the vertex. Thus, the stack maintains the guessed colors on the path from the current vertex to the root, allowing correctness to be checked.

▶ **Lemma 16 (♠).** *For any nice $s(n)$,* td-3Coloring$[s]$ *is in* $\mathbf{NAuxSA}[\mathrm{poly}, \log, s]_{time\ space\ height}$.

Clearly if $s(n) \geq \log^2 n$, $\mathbf{NAuxSA}[\mathrm{poly}, \log, s]_{time\ space\ height} \subseteq \mathbf{NAuxSA}[\mathrm{poly}, s/\log, s]_{time\ space\ height}$. The next step is to show how the stack operations of the latter class' machines can be regularized. This idea originates in the approach of Akatov and Gottlob [3]. Following [3], we define a regular stack machine in the following way. For any valid sequence S of push/pop operations that starts and ends with an empty stack, define the corresponding push-pop tree $\tau(S)$ to be the ordered tree (a rooted tree with an order imposed on the children of each node) in which a depth-first search would result in the sequence S, where entering/withdrawing from a vertex corresponds to a push/pop operation. We say that a language is in reg-$\mathbf{NAuxSA}[\mathrm{poly}, s/\log, s]_{time\ space\ height}$ if it is recognized by an NTM M with $s(n)/\log(n)$ working space and an auxiliary stack of height $s(n)$ that has the following properties:

(1) M pushes and pops blocks of $\mathbf{b} = \lceil s(n)/\lg(n) \rceil$ symbols at a time, say, simultaneously from/to the first $\mathbf{b}$ positions of the worktape.

(2) Whenever M decides to push or pop, it can only change its state. Moreover, the decision about using a push or pop transition depends only on the machine's state.

(3) If M accepts input α, then there is a run on α where the push-pop tree (where pushing/popping a block is considered atomic) is the full binary tree of depth exactly $c\lceil \lg n \rceil$, for some fixed integer c. In particular, at the moment of accepting the stack is empty.

Restriction (2) is a technical adjustment. Restriction (1) is easily achieved by simulating the top symbols from the stack in a length $\mathbf{b}$ buffer on the working tape, pushing and popping a full buffer when needed. The most important restriction is (3): the push-pop tree has a fixed shape of a full binary tree. For this, we use the following observation of Akatov and Gottlob [3, 2], used also by Elberfeld et al. [20]. The *traversal ordering* of the nodes of an ordered tree is the linear ordering which places a parent before its children and, for children a, b of a node, a occurring before b, places all descendants of a before all descendants of b.

▶ **Lemma 17** (Lemma 3.3 of [2]; Theorem 3.14 of [20])**.** *Given an ordered tree T with n nodes and depth at most $\lg n$, one can in logarithmic space compute an embedding (an injection that preserves the ancestor relation and traversal ordering) into a full binary tree of depth $4 \lg n$.*

As in [3], this allows us to regularize our machines, as dummy pushes/pops can be non-deterministically guessed so that the push-pop tree of at least one run is a full binary tree.

▶ **Lemma 18 (♠).** $\mathbf{NAuxSA}[\mathrm{poly}, s/\log, s]_{time\ space\ height} \subseteq$ *reg-*$\mathbf{NAuxSA}[\mathrm{poly}, s/\log, s]_{time\ space\ height}$.

Knowing that computations for NAuxSA can be conveniently regularized, we can describe the existence of such a computation by a CNF formula "wrapped around" the rigid shape of the full binary tree that encodes the push-pop tree of the run. We think of the computation as starting at the root node, moving down an edge whenever a push is made and moving up an edge whenever a pop is made. This was also the idea of Akatov and Gottlob [3], but our reduction needs to introduce many more elements, in particular copies of the gadget of Lemma 11 for every fragment between two push/pop operations. Each part of the computation depends only on symbols pushed onto the stack on the path to the root. This, together with the $\mathcal{O}(\frac{s(n)}{\log n} \cdot \log n) = \mathcal{O}(s(n))$ bound on the tree-depth of the computation gadget, will give rise to a tree-depth decomposition of depth $\mathcal{O}(s(n))$ of the obtained formula's primal graph.

▶ **Lemma 19 (♠).** *If* $L \in$ *reg-*$\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{s/\log}, \underset{height}{s}]$*, then* $L \leq_L$ *td-CNF-SAT*$[s]$.

Lemmas 18 and 19 show that td-CNF-SAT$[s]$ is hard for $\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{s/\log}, \underset{height}{s}]$, and by Lemma 10 so is td-3Coloring$[s]$. Since the closure of $\mathbf{NAuxSA}[\text{poly}, \log, s]$ under logspace reductions is $\mathbf{NAuxSA}[\text{poly}, \log, s(\text{poly})]$, Lemmas 16, 18, 19 give the following chain of containments (here $[A]^L$ denotes the class of problems reducible to A in logspace):

$$\begin{aligned}[\text{td-3Coloring}[s(\text{poly})]]^L \subseteq [\text{td-3Coloring}[s]]^L &\subseteq \mathbf{NAuxSA}[\text{poly}, \log, s(\text{poly})] \\ &\subseteq \mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{s(\text{poly})/\log}, \underset{height}{s(\text{poly})}] \\ &\subseteq [\text{td-3Coloring}[s(\text{poly})]]^L\end{aligned}$$

Therefore, all containments must be equalities, which concludes the proof of Theorems 3 and 4. Now, to prove the determinization of Theorem 5, we only need an algorithm for td-3Coloring$[s]$. The following lemma implies td-3Coloring$[s] \in \mathbf{D}[\underset{space}{s}]$ (for nice s), hence Theorem 3, and the fact that $\mathbf{D}[\underset{space}{s(\text{poly})}]$ is closed under logspace, yield Theorem 5.

▶ **Lemma 20 (♠).** td-3Coloring$[s]$ *can be solved in time* $3^s \cdot \text{poly}(n)$ *and space* $\mathcal{O}(s + \log n)$.

Characterization via alternating machines. In the full version of this paper, we use Theorem 3 to give another characterization in terms of alternating Turing machines with polynomial size of an accepting tree, i.e. *treesize*. Both the notions of ATMs and that of treesize later introduced by Ruzzo [42] gave a new unified view on various complexity classes, simplifying a few containment proofs. Ruzzo showed that $\mathbf{NAuxPDA}[\underset{time}{\text{poly}}, \underset{space}{s}] = \mathbf{A}[\underset{space}{s}, \underset{treesize}{\text{poly}}]$. We show that bounding the time (as opposed to space) of a polynomial treesize ATM, leads to the classes corresponding to small tree-depth, as opposed to small treewidth.

▶ **Theorem 21 (♠).** *Let* $s(n) \geq \log^2(n)$ *be a nice function. Then*

$$\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{\log}, \underset{height}{s(\text{poly})}] = \mathbf{A}[\underset{time}{s(\text{poly})}, \underset{treesize}{\text{poly}}].$$

5 Conclusions

Let $s(n) \geq \log^2 n$ be a nice function such that $\forall_c\ s(n^c) = \mathcal{O}(s(n))$ (e.g. $s(n) = \lg^k n$, $k \geq 2$). The hierarchy of graph parameters of Corollary 8 together with Theorems 1, 3, and 21 implies the following hierarchy of complexity classes between **NL** and **NP**.

$$\begin{array}{ccccccc}
\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{\log}, \underset{height}{s}] & = & [\text{td-3Coloring}[s]]^L & = & \mathbf{A}[\underset{time}{s}, \underset{treesize}{\text{poly}}] & \subseteq & \mathbf{D}[\underset{space}{s}] \\
 & & \cap\vert & & & & \\
\mathbf{N}[\underset{time}{\text{poly}}, \underset{space}{s}] & = & [\text{pw-3Coloring}[s]]^L & = & \mathbf{N}[\underset{time}{\text{poly}}, \underset{space}{s}] & & \\
 & & \cap\vert & & & & \\
\mathbf{NAuxPDA}[\underset{time}{\text{poly}}, \underset{space}{s}] & = & [\text{tw-3Coloring}[s]]^L & = & \mathbf{A}[\underset{space}{s}, \underset{treesize}{\text{poly}}] & \subseteq & \mathbf{D}[2^{\mathcal{O}(s)}]_{time} \\
 & & \cap\vert & & & & \\
\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{\log}, \underset{height}{s \cdot \log}] & = & [\text{td-3Coloring}[s \cdot \log]]^L & = & \mathbf{A}[\underset{time}{s \cdot \log}, \underset{treesize}{\text{poly}}] & \subseteq & \mathbf{D}[\underset{space}{s \cdot \log}]
\end{array}$$

For $s(n) = \log^k(n)$, the classes have been considered under different names:

- $\mathbf{NAuxSA}[\underset{time}{\text{poly}}, \underset{space}{\log}, \underset{height}{\log^k}]$ was named $\mathbf{DC}^{k-1}$ (for *divide and conquer*) in [3, 2],
- $\mathbf{N}[\underset{time}{\text{poly}}, \underset{space}{\log^k}]$ are known as $\mathbf{NSC}^k$ (the non-deterministic variant of *Steve's Class*),
- $\mathbf{NAuxPDA}[\underset{time}{\text{poly}}, \underset{space}{\log^k}]$ is shown equal to a class named $\mathbf{SAC}^k_{quasi}$ in [4].

This yields the following hierarchy:

$$\mathbf{L} \subseteq \begin{array}{c}\mathbf{NL}\\ \| \\ \mathbf{NSC}^1\end{array} \subseteq \begin{array}{c}\mathbf{SAC}^1\\ \| \\ \mathbf{SAC}^1_{quasi}\end{array} \subseteq \mathbf{DC}^1 \subseteq \cdots \subseteq \mathbf{DC}^{k-1} \subseteq \mathbf{NSC}^k \subseteq \mathbf{SAC}^k_{quasi} \subseteq \mathbf{DC}^k \subseteq \cdots \subseteq \mathbf{NP}$$

References

1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78. IEEE Computer Society, 2015.
2. Dmitri Akatov. *Exploiting parallelism in decomposition methods for constraint satisfaction.* PhD thesis, University of Oxford, 2010.
3. Dmitri Akatov and Georg Gottlob. Balanced queries: Divide and conquer. In *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6281 of *Lecture Notes in Computer Science*, pages 42–54. Springer, 2010. doi:10.1007/978-3-642-15155-2_6.
4. Eric Allender, Shiteng Chen, Tiancheng Lou, Periklis A. Papakonstantinou, and Bangsheng Tang. Width-parametrized SAT: time–space tradeoffs. *Theory of Computing*, 10:297–339, 2014. doi:10.4086/toc.2014.v010a012.
5. Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.
6. Per Austrin, Petteri Kaski, Mikko Koivisto, and Jussi Määttä. Space-time tradeoffs for subset sum: An improved worst case algorithm. In *Automata, Languages, and Programming – 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 45–56. Springer, 2013.
7. Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994. doi:10.1145/174644.174650.
8. Marina Barsky, Ulrike Stege, Alex Thomo, and Chris Upton. Shortest path approaches for the longest common subsequence of a set of strings. In *Proceedings of the 7th IEEE International Conference on Bioinformatics and Bioengineering, BIBE 2007, October 14-17, 2007, Harvard Medical School, Boston, MA, USA*, pages 327–333. IEEE Computer Society, 2007. doi:10.1109/BIBE.2007.4375584.
9. Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *CoRR*, abs/1007.1161, 2010. URL: http://arxiv.org/abs/1007.1161.
10. Hans L. Bodlaender, Jitender S. Deogun, Klaus Jansen, Ton Kloks, Dieter Kratsch, Haiko Müller, and Zsolt Tuza. Rankings of graphs. *SIAM J. Discrete Math.*, 11(1):168–181, 1998. doi:10.1137/S0895480195282550.
11. Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Harold T. Wareham. The parameterized complexity of sequence alignment and consensus. *Theor. Comput. Sci.*, 147(1&2):31–54, 1995. doi:10.1016/0304-3975(94)00251-D.
12. Liming Cai and David W. Juedes. On the existence of subexponential parameterized algorithms. *J. Comput. Syst. Sci.*, 67(4):789–807, 2003. doi:10.1016/S0022-0000(03)00074-6.

13 Hubie Chen and Moritz Müller. One hierarchy spawns another: graph deconstructions and the complexity classification of conjunctive queries. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS'14, Vienna, Austria, July 14-18, 2014*, pages 32:1–32:10. ACM, 2014. doi:10.1145/2603088.2603107.

14 Bruno Courcelle. The Monadic Second-Order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

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

16 Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and h-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

17 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

18 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.

19 Michael Elberfeld, Martin Grohe, and Till Tantau. Where first-order and monadic second-order logic coincide. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 265–274. IEEE Computer Society, 2012.

20 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of bodlaender and courcelle. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:62, 2010. Extended abstract included in the proceedings of FOCS 2010.

21 Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3):661–701, 2015. doi:10.1007/s00453-014-9944-y.

22 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1 edition, 2006.

23 Fedor V. Fomin, Petteri Kaski, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Parameterized single-exponential time polynomial space algorithm for Steiner Tree. In *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 494–505. Springer, 2015.

24 Martin Fürer and Huiwen Yu. Space saving by dynamic algebraization. In *Computer Science – Theory and Applications – 9th International Computer Science Symposium in Russia, CSR 2014, Moscow, Russia, June 7-11, 2014. Proceedings*, volume 8476 of *Lecture Notes in Computer Science*, pages 375–388. Springer, 2014.

25 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

26 Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, 2001. doi:10.1145/382780.382783.

27 Sylvain Guillemot. Parameterized complexity and approximability of the longest compatible sequence problem. *Discrete Optimization*, 8(1):50–60, 2011. doi:10.1016/j.disopt.2010.08.003.

28 Meir Katchalski, William McCuaig, and Suzanne M. Seager. Ordered colourings. *Discrete Mathematics*, 142(1-3):141–154, 1995.

29 Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994. doi:10.1007/BFb0045375.

30 Alexander Langer, Felix Reidl, Peter Rossmanith, and Somnath Sikdar. Practical algorithms for MSO model-checking on tree-decomposable graphs. *Computer Science Review*, 13-14:39–74, 2014. doi:10.1016/j.cosrev.2014.08.001.

31 Richard J Lipton. Savitch's theorem. In *The P=NP Question and Gödel's Lost Letter*, pages 135–138. Springer, 2010.

32 Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

33 Daniel Lokshtanov, Matthias Mnich, and Saket Saurabh. Planar k-path in subexponential time and polynomial space. In *Graph-Theoretic Concepts in Computer Science – 37th International Workshop, WG 2011, Teplá Monastery, Czech Republic, June 21-24, 2011. Revised Papers*, volume 6986 of *Lecture Notes in Computer Science*, pages 262–270. Springer, 2011. doi:10.1007/978-3-642-25870-1_24.

34 Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 321–330. ACM, 2010. doi:10.1145/1806689.1806735.

35 Pinyan Lu, Jialin Zhang, Chung Keung Poon, and Jin-yi Cai. Simulating undirected *st*-connectivity algorithms on uniform jags and nnjags. In *Algorithms and Computation, 16th International Symposium, ISAAC 2005, Sanya, Hainan, China, December 19-21, 2005, Proceedings*, volume 3827 of *Lecture Notes in Computer Science*, pages 767–776. Springer, 2005. doi:10.1007/11602613_77.

36 Burkhard Monien and Ivan Hal Sudborough. Bandwidth constrained NP-complete problems. *Theor. Comput. Sci.*, 41:141–167, 1985. doi:10.1016/0304-3975(85)90068-4.

37 Jesper Nederlof. Fast polynomial-space algorithms using inclusion-exclusion. *Algorithmica*, 65(4):868–884, 2013. doi:10.1007/s00453-012-9630-x.

38 J. Nešetřil and P. Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.

39 Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet Shortest Common Supersequence and Longest Common Subsequence problems. *J. Comput. Syst. Sci.*, 67(4):757–771, 2003.

40 Alex Pothen. The complexity of optimal elimination trees, 1988. Technical Report CS 88-16, Pennsylvania State University.

41 Neil Robertson and Paul D. Seymour. Graph Minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

42 Walter L. Ruzzo. Tree-size bounded alternation. *J. Comput. Syst. Sci.*, 21(2):218–235, 1980. doi:10.1016/0022-0000(80)90036-7.

43 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

44 V. Vinay and V. Chandru. The expressibility of nondeterministic auxiliary stack automata and its relation to treesize bounded alternating auxiliary pushdown automata. In *Foundations of Software Technology and Theoretical Computer Science, Tenth Conference, Bangalore, India, December 17-19, 1990, Proceedings*, volume 472 of *Lecture Notes in Computer Science*, pages 104–114. Springer, 1990. doi:10.1007/3-540-53487-3_38.

Improved Approximation Algorithms for Balanced Partitioning Problems

Harald Räcke[1] **and Richard Stotz**[2]

1 **Technische Universität München, Garching, Germany**
raecke@in.tum.de
2 **Technische Universität München, Garching, Germany**
stotz@in.tum.de

Abstract

We present approximation algorithms for balanced partitioning problems. These problems are notoriously hard and we present new bicriteria approximation algorithms, that approximate the optimal cost and relax the balance constraint.

In the first scenario, we consider Min-Max k-Partitioning, the problem of dividing a graph into k equal-sized parts while minimizing the maximum cost of edges cut by a single part. Our approximation algorithm relaxes the size of the parts by $(1+\varepsilon)$ and approximates the optimal cost by $\mathcal{O}(\log^{1.5} n \log\log n)$, for every $0 < \varepsilon < 1$. This is the first nontrivial algorithm for this problem that relaxes the balance constraint by less than 2.

In the second scenario, we consider strategies to find a minimum-cost mapping of a graph of processes to a hierarchical network with identical processors at the leaves. This Hierarchical Graph Partitioning problem has been studied recently by Hajiaghayi et al. who presented an $(\mathcal{O}(\log n), (1+\varepsilon)(h+1))$ approximation algorithm for constant network heights h. We use spreading metrics to give an improved $(\mathcal{O}(\log n), (1+\varepsilon)h)$ approximation algorithm that runs in polynomial time for arbitrary network heights.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases graph partitioning, dynamic programming, scheduling

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.58

1 Introduction

The problem of scheduling the processes of a parallel application onto the nodes of a parallel system can in its full generality be viewed as a graph mapping problem. Given a graph $G = (V_G, E_G)$ that represents the *workload*, and a graph $H = (V_H, E_H)$ that represents the *physical computing resources*, we want to map G onto H such that on the one hand the processing load is well balanced, and on the other hand the communication load is small.

More concretely, nodes of G represent processes and are weighted by a weight function $w : V_G \to \mathbb{R}_+$. Edges of G represent communication requirements between processes, and are weighted by $c_G : E_G \to \mathbb{R}_+$. Nodes and edges of H represent processors and communication links, respectively, and may also be weighted with $w_H : V_H \to \mathbb{R}_+$ representing the processing capacity of a physical node, and $c_H : E_H \to \mathbb{R}_+$ representing the bandwidth capacity of a communication link. The goal is to map nodes of G to nodes of H, and to map edges of G to paths in H between the corresponding end-points such that a) every node in H obtains approximately the same weight and b) the communication load is small.

This general problem is very complex, and research has therefore focused on special cases. Many results can be interpreted as mapping to a star graph H with k leaves where the center


© Harald Räcke and Richard Stotz;
licensed under Creative Commons License CC-BY
33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 58; pp. 58:1–58:14

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


node x has $w_H(x) = 0$, i.e., processes may not be mapped to the center ($c_H(e)$ is assumed to be uniform). When the goal is to minimize *total communication load* (sum over all edge loads in H) this problem is known as Min-Sum k-partitioning. You want to partition G into k disjoint pieces $V_1, \ldots, V_k$ such that the weight of any piece is at most $w(V_i) \leq w(V)/k$ but the total capacity of edges between partitions is minimized.

For this problem it is not possible to obtain meaningful approximation guarantees without allowing a slight violation of the balance constraints ($w(V_i) \leq w(V)/k$). Even et al. [5] obtain a bicriteria approximation guarantee of $(\mathcal{O}(\log n), 2)$, which means they obtain a solution that may violate balance constraints by a factor of 2, and that has a cost that is at most an $\mathcal{O}(\log n)$-factor larger than the cost of the optimum solution that does *not* violate balance constraints. This has been improved by Krauthgamer et al. [9] to $(\mathcal{O}(\sqrt{\log n \log k}), 2)$. If one aims to violate the balance constraints by less than 2, completely different algorithms seem to be required that are based on Dynamic Programming. Andreev and Räcke [1] obtain an $(\mathcal{O}(\log^{1.5} n), 1+\varepsilon)$-approximation which has been improved by Feldmann and Foschini to $(\mathcal{O}(\log n), 1+\varepsilon)$. For the case that the weight function w_H is non-uniform Krauthgamer et al. [10] obtain an $(\mathcal{O}(\log n), \mathcal{O}(1))$-approximation, which has been improved by Makarychev and Makarychev [11] to $(\mathcal{O}(\sqrt{\log n \log k}), 5+\varepsilon)$.

When the goal is to minimize the *congestion* (maximum load of a link) in the star network H, instead of the total communication load, the problem turns into the so-called Min-Max k-Partitioning problem. You want to partition a given graph G into k-parts $V_1, \ldots, V_k$ of equal size such that $\max_i c(V_i)$ is minimized, where $c(V_i)$ denotes the total weight of edges leaving V_i in G. For this problem Bansal et al. [2] obtain an $(\mathcal{O}(\sqrt{\log n \log k}), 2+\varepsilon)$-approximation but no non-trivial approximation that violates the balance constraint by less than 2 is known.

Hajiaghayi et al. [7] consider the mapping problem when H is not just a star but exhibits parallelism on many different levels. For example, a large parallel system may consist of many workstations, each equipped with several CPUs, etc. Such parallel systems are usually highly symmetric. Therefore, Hajiaghayi et al. model them as a tree in which all subtrees routed at a specific level are isomorphic. They develop an approximation algorithm for a Hierarchical Graph Partitioning problem which then gives rise to a scheduling algorithm for such a *regular tree topology* with an approximation guarantee of $(\mathcal{O}(\log n), (1+\varepsilon)(h+1))$. This algorithm runs in polynomial time as long as the height h of the hierarchy is constant.

1.1 Our Contribution

In this paper we first consider the Min-Max k-Partitioning problem and show how to obtain a polylogarithmic approximation on the cost while only violating the balance constraints by a factor of $1+\varepsilon$. For this we use a dynamic programming approach on trees, and then use an approximation of the graph by a *single* tree to obtain our result. Note that results that approximate the graph by a convex combination of trees [12] are not useful for obtaining an approximation for Min-Max k-Partitioning, as the objective function is not linear.

▶ **Theorem 1.** *There is a polynomial-time $(1+\varepsilon, 1+\varepsilon)$-approximation algorithm for Min-Max k-Partitioning on trees. This gives rise to an $(\mathcal{O}(\log^{1.5} n \log\log n), 1+\varepsilon)$-approximation algorithm for general graphs.*

Then we consider the Hierarchical Graph Partitioning problem as introduced by Hajiaghayi et al. [7]. We give a slight improvement on the factor by which the balance constraints are violated, while maintaining the same approximation w.r.t. cost. More crucially, our result also works if the height h is not constant.

▶ **Theorem 2.** *There exists an* $(\mathcal{O}(\log n), (1+\varepsilon)h)$ *approximation algorithm for Hierarchical Graph Partitioning whose running time is polynomial in* h *and* n*.*

Our technique is heavily based on spreading metrics and we show that a slight variation of the techniques in [5] can be applied.

At first glance the result of Theorem 2 does not look very good as the factor by which the balance constraints are violated seems to be very high as it depends on h. However, we give some indication that a large dependency on h might be necessary. We show that an approximation guarantee of $\alpha \leq h/2$ implies an algorithm for the following approximate version of parallel machine scheduling (PMS). Given a PMS instance I, with a set T of tasks, a length w_t, $t \in T$ for every task, and a deadline d, we define the *g-copied instance of I* ($g \in \mathbb{N}$) as the instance where each task is copied g times and the deadline is multiplied by g. The approximate problem is the following: Given I, either certify that I is not feasible or give a solution to the g-copied instance of I for some $g \leq \alpha$. It seems unlikely that this problem can be solved efficiently for constant α, albeit we do not have a formal hardness proof. These results have been deferred to the full version.

1.2 Basic Notation

Throughout this paper, $G = (V, E)$ denotes the input graph with $n = |V|$ vertices. Its edges are given a cost function $c : E \to \mathbb{N}$ and its vertices are weighted by $w : V \to \mathbb{N}$. For a subset of vertices $S \subseteq V$, $w(S)$ denotes the total weight of vertices in S and $c(S)$ denotes the total cost of edges leaving S, i.e., $c(S) = \sum_{e=\{x,y\}\in E,\ |\{x,y\}\cap S|=1} c(e)$. We will sometimes refer to $w(S)$ as the *size* of S and to $c(S)$ as its *boundary cost*. In order to simplify the presentation, we assume that all edge costs and vertex weights are polynomially bounded in the number of vertices n. This can always be achieved with standard rounding techniques at the loss of a factor $(1+\varepsilon)$ in the approximation guarantee.

A *partition* $\mathcal{P}$ of the vertex set V is a collection $\mathcal{P} = \{P_i\}_i$ of pairwise disjoint subsets of V with $\bigcup_i P_i = V$. We define two different cost functions for partitions, namely $\text{cost}^{\text{sum}}(\mathcal{P}) = \sum_i c(P_i)$ and $\text{cost}^{\max}(\mathcal{P}) = \max_i c(P_i)$. We drop the superscript whenever the type is clear from the context. A (k,ν)-*balanced* partition is a partition into at most k parts, such that the inequality $w(P_i) \leq \nu \cdot w(V)/k$ holds for all parts. The costs of the $(k,1)$-balanced partition with minimum cost w.r.t. cost^{sum} and $\text{cost}^{\max}$ are denoted with $\mathsf{OPT}^{\text{sum}}(k,G)$ and $\mathsf{OPT}^{\max}(k,G)$, respectively.

A *hierarchical partition* $\mathcal{H} = (\mathcal{P}_1, \ldots, \mathcal{P}_h)$ of height h is a sequence of h partitions, where $\mathcal{P}_{\ell+1}$ is a *refinement* of $\mathcal{P}_\ell$, i.e., for every $S \in \mathcal{P}_{\ell+1}$, there is a set $S' \in \mathcal{P}_\ell$ with $S \subseteq S'$. We call $\mathcal{P}_\ell$ the level-ℓ-partition of $\mathcal{H}$. For any cost vector $\vec{\mu} \in \mathbb{R}^h$, the cost of $\mathcal{H}$ is given by $\text{cost}_{\vec{\mu}}(\mathcal{H}) = \sum_{\ell=1}^{h} \mu_\ell \,\text{cost}^{\text{sum}}(\mathcal{P}_\ell)$. A $(\vec{k},\nu)$-*balanced* hierarchical partition is a hierarchical partition where $\mathcal{P}_\ell$ is (k_ℓ,ν)-balanced. The minimum cost of a $(\vec{k},1)$-balanced hierarchical partition w.r.t. $\vec{\mu}$ is denoted with $\mathsf{OPT}_{\vec{\mu}}(\vec{k},G)$.

An (α,β)-*approximate* hierarchical partition with respect to $\vec{\mu}$ and $\vec{k}$ is a $(\vec{k},\beta)$-balanced hierarchical partition whose cost is at most $\alpha \cdot \mathsf{OPT}_{\vec{\mu}}(\vec{k},G)$. An (α,β)-*approximation algorithm* for Hierarchical Graph Partitioning finds an (α,β)-approximate hierarchical partition for any given input graph and cost vector. This also gives an (α,β)-approximation for scheduling on regular tree topologies (see [7]).

2 Min-Max K-Partitioning

In this section, we present an approximation algorithm for Min-Max k-Partitioning. Recall that this problem asks to compute a $(k,1)$-balanced partition $\mathcal{P}$ of a given graph that

minimizes $\text{cost}^{\max}(\mathcal{P})$. We first consider instances, where the input graph is a tree $T = (V, E)$. A suitable approximation of the graph by a tree (see [3], [8], [13]) allows to extend the results to arbitrary graphs with a small loss in the approximation factor. Our algorithm is a decision procedure, i.e., it constructs for a given bound b a solution whose cost is at most $(1+\varepsilon)b$ or proves that $b < \mathsf{OPT}^{\max}(k, G)$. A $(1+\varepsilon, 1+\varepsilon)$-approximation algorithm follows by using binary search. In order to simplify the presentation, we assume throughout this chapter that all vertex weights are 1 and that k divides n. We further assume that the approximation constant ε is at most 1.

The algorithm heavily relies on the construction of a *decomposition of* T, which is defined as follows. A decomposition of T w.r.t. k and b is a partition $\mathcal{D} = \{D_i\}_i$, whose parts D_i are connected components, have size $w(D_i) \leq n/k$ and boundary cost $c(D_i) \leq b$. With each decomposition $\mathcal{D}$ we associate a corresponding *vector set* $\mathcal{I}_\mathcal{D}$. This set $\mathcal{I}_\mathcal{D}$ contains a vector $(c(D_i)/b, w(D_i)k/n)$ that encodes the size and boundary cost of D_i in a normalized way. By this $\mathcal{I}_\mathcal{D}$ contains (most of) the information about $\mathcal{D}$ and our algorithm can just work with $\mathcal{I}_\mathcal{D}$ instead of the full decomposition $\mathcal{D}$. Note that every vector in $\mathcal{I}_\mathcal{D}$ is bounded by $(1, 1)$.

We call a partition of $\mathcal{I}_\mathcal{D}$ into k subsets $\{I_i\}_i$, s.t. the vectors in each subset sum to at most α in both dimensions, an *α-packing* of $\mathcal{I}_\mathcal{D}$. The decomposition $\mathcal{D}$ is called *α-feasible* if an α-packing of $\mathcal{I}_\mathcal{D}$ exists. Note that this implies $\alpha \geq 1$. Also note that this definition of feasibility may be nonstandard, as feasibility incorporates the decomposition cost, i.e., a 1-packing has cost b (the bound that we guessed for the optimal solution).

Determining whether there is an α-packing of a given vector set is an instance of the NP-hard Vector Scheduling problem. Chekuri and Khanna [4] gave a polynomial-time approximation scheme (PTAS) for the Vector Scheduling problem which leads to the following result.

▶ **Lemma 3.** *Let $\mathcal{D}$ be an α-feasible decomposition of T w.r.t. k and b, then there is a polynomial-time algorithm to construct a $(k, (1+\varepsilon)\alpha)$-balanced partition of T of cost at most $(1+\varepsilon)\alpha \cdot b$, for any $\varepsilon > 0$.*

Proof. The proof has been deferred to the full version. ◀

An optimal (but slow) algorithm for Min-Max k-Partitioning could iterate over all possible decompositions of the graph and check if any of these decompositions is 1-feasible. If a 1-feasible decomposition is found, the corresponding 1-packing is computed, otherwise the bound b is rejected.

We modify this approach to obtain a polynomial-time approximation algorithm. As the number of decompositions is exponential, we partition them into a polynomial number of *classes*. This classification is such that decompositions of the same class are similar w.r.t. feasibility. More precisely, we show that if a decomposition is α-feasible, then all decompositions of the same class are $(1+\varepsilon)\alpha$-feasible.

We present a dynamic program that, for every class, checks whether a decomposition in that class exists and that computes some representative decomposition that is in this class. Then, we check the feasibility of every representative. As an exact check is NP-hard, we use the approximate Vector Scheduling as described above. If a 1-feasible decomposition exists, then a representative of the same class has been computed and furthermore, this representative is $(1+\varepsilon)$-feasible. Consequently, we obtain a $(1+\varepsilon)^2$-packing of the representative, which induces a $(k, (1+\varepsilon)^2)$-balanced partition of T with cost at most $(1+\varepsilon)^2 b$.

2.1 Classification of Decompositions

We now describe the classification of decompositions using the vector set $\mathcal{I}_\mathcal{D}$. A vector $\vec{p} \in \mathcal{I}_\mathcal{D}$ is *small*, if both its coordinates are at most ε^3 , otherwise it is *large*. Small and large vectors are henceforth treated separately.

We define a *type of a large vector* $\vec{p} = (p_1, p_2)$ as follows. Informally, the type of a large vector classifies the value of both coordinates. Let $i = 0$ if $p_1 < \varepsilon^4$, otherwise let i be the integer such that p_1 lies in the interval $[(1+\varepsilon)^{i-1}\varepsilon^4, (1+\varepsilon)^i\varepsilon^4)$. Let $j = 0$ if $p_2 < \varepsilon^4$, otherwise let j be the integer such that p_2 lies in $[(1+\varepsilon)^{j-1}\varepsilon^4, (1+\varepsilon)^j\varepsilon^4)$. Note that i and j can each take $t = \lceil \log_{1+\varepsilon}(1/\varepsilon^4) \rceil + 1$ different values because $\vec{p}$ is upper-bounded by $(1,1)$. This is a consequence of normalizing the vectors in $\mathcal{I}_\mathcal{D}$ as mentioned earlier. The pair (i,j) is the *type of* $\vec{p}$. We number types from $1, \ldots, t^2$ arbitrarily.

We next define the *type of a small vector* $\vec{q} = (q_1, q_2)$. Informally, the type of a small vector approximates the *ratio* q_1/q_2 of its coordinates. Let $s = \lceil \log_{1+\varepsilon}(1/\varepsilon) \rceil$ and subdivide $[\varepsilon, 1/\varepsilon]$ into $2s$ intervals of the form $[(1+\varepsilon)^i, (1+\varepsilon)^{i+1})$ for $i = -s, \ldots, s-1$. Crop the first and the last interval at ε and $1/\varepsilon$ respectively. Add two outer intervals $[0, \varepsilon)$ and $[1/\varepsilon, \infty)$ to obtain a discretization of the positive reals into $2s+2$ types $-(s+1), \ldots, s$. The vector $\vec{q} = (q_1, q_2)$ has type i, if its ratio q_1/q_2 falls into the i-th interval.

Using these terms, we define the size signature and ratio signature of a vector set $\mathcal{I}_\mathcal{D}$. The size signature stores the number of large vectors of every type while the ratio signature stores the *sum* of small vectors of every type. Let $\mathcal{I}_\mathcal{D}^{\text{large}}$ denote the subset of large vectors and $\mathcal{I}_{\mathcal{D},j}^{\text{small}}$ denote the subset of small vectors of type j.

▶ **Definition 4** (Signature). Let $\mathcal{D}$ be a set of disjoint connected components. The vector $\vec{\ell} = (\ell_1, \ldots, \ell_{t^2})$ is the *size signature* of $\mathcal{I}_\mathcal{D}$, if $\mathcal{I}_\mathcal{D}^{\text{large}}$ contains exactly ℓ_i vectors of type i for $i = 1, \ldots, t^2$. The vector $\vec{h} = (\vec{h}_{-(s+1)}, \ldots, \vec{h}_s)$ is the *ratio signature* of $\mathcal{I}_\mathcal{D}$ if $\vec{h}_j = \sum_{\vec{p} \in \mathcal{I}_{\mathcal{D},j}^{\text{small}}} \vec{p}$ for $j = -(s+1), \ldots, s$. We call $\vec{g} = (\vec{\ell}, \vec{h})$ the signature of $\mathcal{D}$ and $\mathcal{I}_\mathcal{D}$.

We prove in the following that decompositions with the same signature have nearly the same feasibility properties. We start with a technical claim whose proof is omitted here.

▶ **Claim 5.** *Let* $0 < \varepsilon \leq 1$ *and* $s = \lceil \log_{1+\varepsilon}(1/\varepsilon) \rceil$. *Then* $s \cdot \varepsilon^3 \leq 2\varepsilon$.

Proof. The proof has been deferred to the full version. ◀

▶ **Lemma 6.** *Assume that* $\mathcal{D}$ *is an* α*-feasible decomposition of signature* $\vec{g}$ *and that* $\mathcal{D}'$ *has the same signature. Then* $\mathcal{D}'$ *is* $(1+9\varepsilon)\alpha$*-feasible.*

Proof. Given an α-packing $\{I_i\}_i$ of $\mathcal{I}_\mathcal{D}$, we describe a $(1+9\varepsilon)\alpha$-packing $\{I'_i\}_i$ of $\mathcal{I}_{\mathcal{D}'}$. For every vector $\vec{p} \in \mathcal{I}_{\mathcal{D}'}$, we have to select a set I'_i to add this vector to. We only argue about the first coordinate of the resulting packing, the reasoning for the second coordinate is analogous.

We start with the large vectors. We choose an arbitrary bijection $\pi : \mathcal{I}_{\mathcal{D}'}^{\text{large}} \to \mathcal{I}_\mathcal{D}^{\text{large}}$ such that only vectors of the same type are matched. This can be done easily, since $\mathcal{I}_\mathcal{D}$ and $\mathcal{I}_{\mathcal{D}'}$ have the same size signature. Now, we add a vector $\vec{p} \in \mathcal{I}_{\mathcal{D}'}^{\text{large}}$ to set I'_i if and only if $\pi(\vec{p}) \in I_i$.

As vector $\vec{p} = (p_1, p_2)$ and $\pi(\vec{p})$ share the same signature, their sizes are similar in both coordinates. More precisely if the first coordinate of $\vec{p}$ is larger than ε^4, then the first coordinate of $\pi(\vec{p})$ is at most $(1+\varepsilon)p_1$. If p_1 is smaller than ε^4, then p_2 must be larger than ε^3 because $\vec{p}$ is a large vector. Consequently there can be at most α/ε^3 large vectors with such a small first coordinate in the same part of α-packing $\{I_i\}_i$. Their sum is bounded by

$\alpha\varepsilon$ in the first coordinate. Let $(L_{i,\text{large}}, R_{i,\text{large}})$ denote the sum of large vectors in I_i and let $(L'_{i,\text{large}}, R'_{i,\text{large}})$ denote the sum of large vectors in I'_i. We have

$$L'_{i,\text{large}} \leq (1+\varepsilon)L_{i,\text{large}} + \alpha\varepsilon \ . \tag{1}$$

We now consider small vectors; recall that they are classified by their ratio. Let $(L_{i,j}, R_{i,j})$ denote the sum of small vectors of type j in I_i. We call types $j \geq 0$ *left-heavy* as they fulfill $L_{i,j} \geq R_{i,j}$, and accordingly we call types $j < 0$ *right-heavy* as they fulfill $L_{i,j} < R_{i,j}$.

For left-heavy types, add vectors of $\mathcal{I}^{\text{small}}_{\mathcal{D}',j}$ to I'_i until the sum $(L'_{i,j}, R'_{i,j})$ of these vectors exceeds $L_{i,j}$ in the first coordinate. It follows that $L'_{i,j} \leq L_{i,j} + \varepsilon^3$, as the excess is at most one small vector. Summing over all left-heavy types gives

$$\sum_{j=0}^{s} L'_{i,j} \leq (s+1)\varepsilon^3 + \sum_{j=0}^{s} L_{i,j} \leq 3\varepsilon\alpha + \sum_{j=0}^{s} L_{i,j} \ , \tag{2}$$

where we used $\alpha \geq 1$ and Claim 5 in the last step.

For right-heavy types, add vectors of $\mathcal{I}^{\text{small}}_{\mathcal{D}',j}$ to I'_i until the sum of these vectors exceeds $R_{i,j}$ in the second coordinate. It follows that $R'_{i,j} \leq R_{i,j} + \varepsilon^3$. We remark that the above procedure distributes all small vectors of any type j. This follows because on the one hand the ratio signature ensures that the sum of vectors of type j is the same in $\mathcal{I}_{\mathcal{D}}$ and $\mathcal{I}_{\mathcal{D}'}$, and on the other hand our Greedy distribution assigns at least as much mass in the heavier coordinate as in the solution for $\{I_i\}_i$. It remains to show an upper bound on $L'_{i,j}$ for right-heavy types.

First consider Type $-(s+1)$, which corresponds to interval $[0, \varepsilon)$. Combining the upper bound on $R'_{i,j}$ for right-heavy types and the fact that $L'_{i,-(s+1)}/R'_{i,-(s+1)} < \varepsilon$, it follows that $L'_{i,-(s+1)} \leq \varepsilon(R_{i,-(s+1)} + \varepsilon^3)$. As $\varepsilon^3 \leq \alpha$ and $R_{i,-(s+1)} \leq \alpha$, this implies $L'_{i,-(s+1)} \leq 2\varepsilon\alpha$.

The remaining types correspond to an interval $[(1+\varepsilon)^j, (1+\varepsilon)^{j+1})$ and both $L_{i,j}/R_{i,j}$ and $L'_{i,j}/R'_{i,j}$ must lie in that interval. We derive a bound on $L'_{i,j}$ as follows:

$$\begin{aligned} L'_{i,j} &\leq (1+\varepsilon)^{j+1} R'_{i,j} \leq (1+\varepsilon)^{j+1}(R_{i,j} + \varepsilon^3) \leq (1+\varepsilon)^{j+1}\left(\frac{L_{i,j}}{(1+\varepsilon)^j} + \varepsilon^3\right) \\ &= (1+\varepsilon)L_{i,j} + (1+\varepsilon)^{j+1}\varepsilon^3 \leq (1+\varepsilon)L_{i,j} + \varepsilon^3 \ . \end{aligned}$$

The first step uses the fact that $L'_{i,j}/R'_{i,j} < (1+\varepsilon)^{j+1}$. The second step uses the upper bound on $R'_{i,j}$ for right-heavy types and the third step uses the bound $L_{i,j}/R_{i,j} \geq (1+\varepsilon)^j$. As $j < 0$ for right-heavy types and $\varepsilon > 0$, the last step follows. Summing up the bounds on all right-heavy types and using Claim 5 gives

$$\sum_{j=-(s+1)}^{-1} L'_{i,j} \leq s\varepsilon^3 + 2\varepsilon\alpha + (1+\varepsilon) \sum_{j=-(s+1)}^{-1} L_{i,j} \leq 4\varepsilon\alpha + (1+\varepsilon) \sum_{j=-(s+1)}^{-1} L_{i,j} \ . \tag{3}$$

We now combine all bounds derived for part I'_i. Let (L_i, R_i) be the sum of vectors in I_i and let (L'_i, R'_i) denote the sum of vectors in I'_i. Clearly $L'_i = L'_{i,\text{large}} + \sum_{j=-(s+1)}^{s} L'_{i,j}$ and a respective equality holds for L_i. The first term $L'_{i,\text{large}}$ is upper-bounded in Equation (1). The sum $\sum_{j=0}^{s} L'_{i,j}$ is upper-bounded in Equation (2), and the sum $\sum_{j=-(s+1)}^{-1} L'_{i,j}$ is

upper-bounded in Equation (3). Combining these bounds gives

$$\begin{aligned} L'_i &= L'_{i,\text{large}} + \sum_{j=-(s+1)}^{-1} L'_{i,j} + \sum_{j=0}^{s} L'_{i,j} \\ &\leq (1+\varepsilon) L_{i,\text{large}} + 8\varepsilon\alpha + \sum_{j=0}^{s} L_{i,j} + (1+\varepsilon) \sum_{j=-(s+1)}^{-1} L_{i,j} \\ &= (1+\varepsilon) L_i + 8\varepsilon\alpha \leq (1+9\varepsilon)\alpha \ . \end{aligned}$$

The last step follows from the assumption that $\{I_i\}_i$ is an α-packing. ◀

2.2 Finding Decompositions of the Tree

We now present a dynamic program that computes a set of decompositions $\mathbb{D}$ with the following property: If there exists a decomposition of T with signature $\vec{g}$, then $\mathbb{D}$ contains a decomposition with that signature. The dynamic program adapts a procedure given by Feldmann and Foschini [6]. We first introduce some terminology.

Fix an arbitrary root r of the tree and some left-right ordering among the children of each internal vertex. This defines the *leftmost* and *rightmost* sibling among the children. For each vertex v, let L_v be the set of vertices that are contained in the subtree rooted at v or in a subtree rooted at some left sibling of v. The dynamic program iteratively constructs sets of connected components of a special form, defined as follows.

A *lower frontier* $\mathcal{F}$ is a set of disjoint connected components with nodes, such that vertices that are not covered by $\mathcal{F}$ form a connected component including the root. In addition we require that components of a lower frontier have at most size n/k and at most cost b. We define the *cost* of a lower frontier $\mathcal{F}$ as the total cost of edges with exactly one endpoint covered by $\mathcal{F}$. Note that this may be counter-intuitive, because the cost of a lower frontier does not consider the boundary cost of the individual connected components.

The algorithm determines for every vertex v, signature $\vec{g}$, cost $\kappa \leq b$ and number of vertices $m \leq n$, if there exists a lower frontier of L_v with signature $\vec{g}$ that covers m vertices with cost κ. It stores this information in a table $D_v(\vec{g}, m, \kappa)$ and computes the entries recursively using a dynamic program. Along with each positive entry, the table stores the corresponding lower frontier. In the following paragraphs, we describe the dynamic program in more detail.

First, consider the case where v is a leaf and the leftmost among its siblings. Let v_p be the parent of v. As $L_v = \{v\}$, the lower frontier of L_v is either empty or a single connected component $\{v\}$. Therefore $D_v((\vec{0}, \vec{0}), 0, 0) = 1$ and $D_v(\vec{g}(1, c(v, v_p)), 1, c(v, v_p)) = 1$. Here, we use $\vec{g}(|S|, c(S))$ to denote the signature of a set that just contains connected component S (recall that $c(S)$ denotes the cost of S). For all other values, $D_v(\vec{g}, m, \kappa) = 0$.

Second, we consider the case where v is neither a leaf, nor the leftmost among its siblings. The case when v is a leaf but not leftmost sibling and the case when v is an inner vertex that is leftmost sibling and follow from an easy adaption. Let w be the left sibling of v and u its rightmost child. Observe that L_v is the disjoint union of L_u, L_w and $\{v\}$.

If the edge from v to its parent is not cut by a lower frontier $\mathcal{F}$ of L_v, then v is not covered by $\mathcal{F}$. Consequently $\mathcal{F}$ splits into two lower frontiers $\mathcal{F}_u$ and $\mathcal{F}_w$ of L_u and L_w respectively. Denote their signatures with $\vec{g}_u$ and $\vec{g}_w$; they must sum up to $\vec{g}$. If $\mathcal{F}_u$ covers m_u vertices, then $\mathcal{F}_w$ needs to cover $m - m_u$ vertices. Similarly, if $\mathcal{F}_u$ has cost κ_u, then $\mathcal{F}_w$

needs cost $\kappa - \kappa_u$. Hence, if the edge from v to its parent is not cut, $D_v(\vec{g}, m, \kappa)$ is equal to

$$\bigvee_{\substack{m_u \leq m,\ \kappa_u \leq \kappa,\\ \vec{g}_u + \vec{g}_w = \vec{g}}} (D_w(\vec{g}_w, m - m_u, \kappa - \kappa_u) \wedge D_u(\vec{g}_u, m_u, \kappa_u)) \ . \tag{4}$$

If the edge from v to its parent v_p is cut by the lower frontier $\mathcal{F}$ of L_v, then $\mathcal{F}$ covers the entire subtree of v. It follows that $\mathcal{F}$ consists of three disjoint parts: a component S that contains v, a lower frontier $\mathcal{F}_u$ of L_u and a lower frontier $\mathcal{F}_w$ of L_w. Denote the cost of $\mathcal{F}_u$ by κ_u and the cost of $\mathcal{F}_w$ by κ_w. Let T_v denote the subtree of v. The cost $c(S)$ equals $\kappa_u + c(v, v_p)$, because S is connected to v_p in the direction of the root and with the highest vertices of $\mathcal{F}_u$ in the direction of the leaves. The cost κ of the lower frontier $\mathcal{F}$ is $\kappa_w + c(v, v_p)$, because all edges leaving $\mathcal{F}_u$ have their endpoints in $\mathcal{F}$. The signatures of the three parts of $\mathcal{F}$ must sum up to $\vec{g}$, therefore $\vec{g} = \vec{g}_u + \vec{g}_w + \vec{g}(|S|, c(S))$. Hence, if the edge from v to its parent is cut, $D_v(\vec{g}, m, \kappa)$ is equal to

$$\bigvee_{\substack{|S| \leq n/k,\ c(S) \leq b,\\ \vec{g}_u + \vec{g}_w + \vec{g}(|S|, c(S)) = \vec{g}}} \Big(D_w(\vec{g}_w, m - |T_v|, \kappa - c(v, v_p)) \wedge D_u(\vec{g}_u, |T_v| - |S|, c(S) - c(v, v_p)) \Big) \ . \tag{5}$$

We conclude that $D_v(\vec{g}, m, \kappa) = 1$, if Term (4) or Term (5) evaluate to 1.

The running time of the dynamic program depends on the number of signatures that need to be considered at each vertex. The following lemma bounds their number, its proof is omitted due to space constraints.

▶ **Lemma 7.** *At vertex v, the dynamic program considers $|L_v|^{t+4s+4}(2b)^{2s+2}\gamma$ signatures, where $|L_v|$ is the size of L_v and $\gamma = (k/\varepsilon^2)^{t^2}$.*

Proof. The proof has been deferred to the full version. ◀

As the number of signatures is polynomial in n and k, it follows that the above dynamic program only needs a polynomial number of steps.

▶ **Observation 8.** *Let $\mathbb{D}$ be the set of decompositions corresponding to positive entries of $D_r(\vec{g}, n, 0)$ as computed by the dynamic program. Then for any decomposition of T with signature $\vec{g}$, there exists a decomposition in $\mathbb{D}$ with the same signature.*

Combining the dynamic program with the properties of signatures, we obtain an approximation algorithm for Min-Max k-Partitioning on trees.

▶ **Theorem 9.** *There is a polynomial-time $(1+\varepsilon, 1+\varepsilon)$-approximation algorithm for Min-Max k-Partitioning on trees.*

Proof. Follows from Lemmas 6, 7 and Observation 8. The running time is polynomial as both the dynamic program and the Vector Scheduling subroutine run in polynomial time. ◀

We extend our results to arbitrary undirected graphs $G = (V, E, c)$ with the help of a decomposition tree that acts as a cut sparsifier (see Räcke and Shah [13]). Theorem 1 follows by standard arguments that are deferred to the full version.

Algorithm 1 merge$(\mathcal{H}' = (\mathcal{P}'_1, \dots, \mathcal{P}'_h), \vec{k})$

$\mathcal{P}_1 \leftarrow$ Merge two sets of $\mathcal{P}'_1$ with minimum combined weight until $|\mathcal{P}'_1| = k_1$.
for $\ell \leftarrow 2, \dots, h$ **do**
  $d \leftarrow k_\ell / k_{\ell-1}$.
  for all $P_{\ell-1,i} \in \mathcal{P}_{\ell-1}$ **do**
    $Q \leftarrow \{P' \in \mathcal{P}'_\ell \mid P' \cap P_{\ell-1,i} \neq \emptyset\}$. // Subclusters of $P_{\ell-1,i}$
    $P_{\ell,1} \leftarrow \emptyset, \dots, P_{\ell,d} \leftarrow \emptyset$.
    while $Q \neq \emptyset$ **do**
      Assign Q's next element to $P_{\ell,j}$ with smallest weight.
    end while
    $\mathcal{P}_\ell \leftarrow \mathcal{P}_\ell \cup \bigcup_j P_{\ell,j}$.
  end for
end for
Return $\mathcal{H} = (\mathcal{P}_1, \dots, \mathcal{P}_h)$.

3 Hierarchical Partitioning

In this section, we present a bicriteria approximation algorithm for Hierarchical Graph Partitioning. In the Hierarchical Graph Partitioning problem, we are given a graph $G = (V, E)$ and parameters $\vec{k}$ and $\vec{\mu}$. We are asked to compute a $(\vec{k}, 1)$-balanced hierarchical partition $\mathcal{P}$ of G that minimizes $\text{cost}_{\vec{\mu}}(\mathcal{P})$ among all such partitions. We assume furthermore that a $(\vec{k}, 1)$-balanced partition of G exists and therefore $k_{\ell+1}/k_\ell$ is integral for all levels. This assumption is fulfilled in particular when $\vec{k}$ is derived from a regular hierarchical network. Note that this is an extension of Min-Sum k-Partitioning and that $\text{cost}_{\vec{\mu}}$ minimizes the weighted *sum* of all edges cut by all subpartitions. This contrasts with the objective function $\text{cost}^{\max}$ considered in the previous section.

Our algorithm relies on the construction of *graph separators*. Graph separators are partitions in which the maximum weight of a part is bounded, but the number of parts is arbitrary. More precisely, a *σ-separator* of G is a partition $\mathcal{P}$ of G, such that $w(P_i) \leq w(V)/\sigma$ for every i.

For any positive vector $\vec{\sigma} \in \mathbb{R}^h$, a *$\vec{\sigma}$-hierarchical separator* of G is a hierarchical partition of G, where on every level $\mathcal{P}_\ell$ is a σ_ℓ-separator. Note that any (k, ν)-balanced partition is a (k/ν)-separator and any $(\vec{k}, \nu)$-balanced hierarchical partition is a $(\vec{k}/\nu)$-hierarchical separator. An *α-approximate* $\vec{\sigma}$-hierarchical separator w.r.t. $\vec{k}$ and $\vec{\mu}$ is a σ-hierarchical separator whose cost is at most $\alpha \cdot \mathsf{OPT}_{\vec{\mu}}(\vec{k}, G)$.

Approximate hierarchical separators can be transformed into approximate hierarchical partitions using the merging procedure described in Algorithm 1. It is essentially a greedy strategy for bin packing on every level that makes sure that partitions remain refinements of each other. The next lemma states that the approximation error of the merging procedure is linear in the height h.

▶ **Lemma 10.** *Let $\mathcal{H}' = (\mathcal{P}'_1, \dots, \mathcal{P}'_h)$ be an α-approximate $\vec{k}/(1+\varepsilon)$-hierarchical separator w.r.t. $\vec{k}$ and $\vec{\mu}$. Then Algorithm 1 constructs an $(\alpha, (1+\varepsilon)(h+1))$-approximate hierarchical partition $\mathcal{H}$ w.r.t. $\vec{\mu}$ and $\vec{k}$.*

Moreover, if $\mathcal{P}'_1$ contains at most k_1 parts, then $\mathcal{H}$ is an $(\alpha, (1+\varepsilon)h)$-approximate hierarchical partition w.r.t. $\vec{\mu}$ and $\vec{k}$.

Proof. We use induction over h. For $h = 1$, assume without loss of generality that $P_{1,1} \in \mathcal{P}_1$ has maximum weight. If $w(P_{1,1}) > 2(1+\varepsilon)w(V)/k_1$, then some merging must have occurred

in the first line of the algorithm. It follows that all distinct parts $P_{1,i}, P_{1,j} \in \mathcal{P}_1$ have combined weight at least $2(1+\varepsilon)w(V)/k_1$. This is a contradiction to the fact that the total weight of all parts is $w(V)$.

Assume that the claim holds for some $h \geq 1$, i.e., the weight of all parts of $\mathcal{P}_h$ is upper-bounded by $(1+\varepsilon)(h+1)w(V)/k_h$. Use $d = k_{h+1}/k_h$. Assume without loss of generality that $P_{h+1,1}$ receives the last element S from Q when considering $P_{h,i}$ in the fourth line of the algorithm; this implies that the weight of $P_{h+1,1}$ is smallest before receiving S. To derive a contradiction, assume that $w(P_{h+1,1}) > (1+\varepsilon)(h+2)w(V)/k_{h+1}$. It follows that

$$\begin{aligned} w(P_{h,i}) &\geq w(S) + \sum_{j=1}^{d} w(P_{h+1,1} \setminus S) > \sum_{j=1}^{d} (w(P_{h+1,1}) - w(S)) \\ &> d \cdot \Big((1+\varepsilon)(h+2)\frac{w(V)}{k_{h+1}} - (1+\varepsilon)\frac{w(V)}{k_{h+1}} \Big) \end{aligned}$$

The last line equals $(1+\varepsilon)(h+1)\cdot w(V)/k_h$ which is a contradiction to the induction hypothesis. For the first inequality, we used that part $P_{h+1,1}$ had the smallest weight before receiving S. The last step uses the fact that the weight of all $S \in Q$ is at most $(1+\varepsilon)w(V)/k_{h+1}$.

As merging sets does not increase the cost of a hierarchical partition, the approximation factor α on the cost does not change. This finishes the proof of the first statement.

To see the second statement, observe that if $\mathcal{P}'_1$ contains at most k_1 parts, then there is nothing to do in the first line of the algorithm. As no merging occurs, every set in $\mathcal{P}'_1$ has at most weight $(1+\varepsilon)w(V)/k_1$. With a similar induction argument as above, it follows that the merging procedure returns an $(\alpha, (1+\varepsilon)h)$-approximate hierarchical partition. ◀

In the following, we describe an algorithm that computes an $\mathcal{O}(\log n)$-approximate $\vec{k}/(1+\varepsilon)$-hierarchical separator $\mathcal{H}' = (\mathcal{P}'_1, \ldots, \mathcal{P}'_h)$. We use $\tilde{\mu}_\ell$ as a shorthand for $\sum_{j=\ell}^{h} \mu_j$.

Our algorithm first computes partition $\mathcal{P}'_1$ using the Min-Sum k-Partitioning algorithm by Feldmann and Foschini [6]. It can easily be adapted to handle polynomial vertex weights.

The following theorem states the result for the Min-Sum k-Partitioning algorithm.

▶ **Theorem 11** ([6]). *There is a polynomial-time algorithm that computes a $(k_1, 1+\varepsilon)$-balanced partition of G with cut cost at most $\mathcal{O}(\log n) \cdot \mathsf{OPT}^{\mathrm{sum}}(k_1, G)$.*

Note that $\tilde{\mu}_1 \cdot \mathsf{OPT}^{\mathrm{sum}}(k_1, G)$ is an upper bound on the cost of any $(\vec{k}, 1)$-balanced hierarchical partition.

The algorithm to construct $(\mathcal{P}'_2, \ldots, \mathcal{P}'_h)$ is an adaptation of the algorithm by Even et al. for Min-Sum k-Partitioning. Note that the following description is basically the description in [5]. It relies on a distance assignment $\{d(e)\}_{e \in E}$ to the edges of G, the *spreading metric*. The distances are used in a procedure called cut-proc that permits to find within a subgraph of G a subset of vertices T, whose cut cost is bounded by its volume $\mathrm{vol}(T)$. The volume of a set of vertices is approximately the weighted length of the edges in its cut.

Given the procedure cut-proc, the algorithm constructs all the remaining partitions $(\mathcal{P}'_2, \ldots, \mathcal{P}'_h)$ as follows. First recall that $\mathcal{P}'_1$ is already given by Theorem 11. For $\ell \geq 2$, the algorithm constructs $\mathcal{P}'_\ell$ by examining all parts of $\mathcal{P}'_{\ell-1}$ separately. On a given part $P'_{\ell-1} \in \mathcal{P}'_{\ell-1}$, it uses procedure cut-proc to identify a subgraph H of $P'_{\ell-1}$. This subgraph becomes a part of $\mathcal{P}'_\ell$ and cut-proc is restarted on $P_{\ell-1} \setminus H$. This process is repeated until less than $(1+\varepsilon)w(V)/k_\ell$ vertices are left. The remaining vertices also constitute a part of $\mathcal{P}'_\ell$.

It is important to note during the following discussion that the boundary cost $c(S)$ of a set $S \subseteq V$ depends on the subgraph that S was chosen from. Sometimes, we will choose S from a subgraph H of G; in this case, $c(S)$ only counts the cost of edges leaving S towards H.

The spreading metric used is an optimal solution to the following linear program, called spreading metrics LP.

$$\begin{aligned} \min & \sum_{e\in E} c(e)d(e) \\ \text{s.t.} & \sum_{u\in S} \text{dist}_V(v,u)\cdot w(u) \geq \tilde{\mu}_\ell\left(w(S) - \frac{w(V)}{k_\ell}\right), \qquad 1\leq \ell \leq h,\ S\subseteq V,\ v\in S \\ & 0\leq d(e)\leq \tilde{\mu}_1, \qquad e\in E\ . \end{aligned} \tag{6}$$

Using the distances $d(e)$, $\text{dist}_S(v,u)$ denotes the length of the shortest path between vertices v and u in subgraph S. Let τ denote the optimal value of the spreading metrics LP.

The following two lemmas prove the basic properties of solutions of the spreading metrics LP. Their proofs are omitted due to space constraints.

▶ **Lemma 12.** *Any $(\vec{k},1)$-balanced hierarchical partition $\mathcal{H}=(\mathcal{P}_1,\ldots,\mathcal{P}_h)$ induces a feasible solution to the spreading metrics LP of value $\text{cost}_{\vec{\mu}}(\mathcal{H})$.*

Proof. The proof has been deferred to the full version. ◀

The above lemma implies that τ is a lower bound on $\text{cost}_{\vec{\mu}}(\mathcal{H})$. We define the radius of vertex v in some $S\subseteq V$ as $\text{radius}(v,S) := \max\{\text{dist}_S(v,u)\mid u\in S\}$.

The following lemma proves that any set of sufficient weight has at least constant radius.

▶ **Lemma 13.** *If $S\subseteq V$ satisfies $w(S) > (1+\varepsilon)w(V)/k_\ell$ for some level ℓ, then for every vertex $v\in S$, $\text{radius}(v,S) > \frac{\varepsilon}{1+\varepsilon}\tilde{\mu}_\ell$.*

Proof. The proof has been deferred to the full version. ◀

Even though the spreading metrics LP contains an exponential number of constraints, it can be solved in polynomial time using the ellipsoid algorithm and a polynomial-time separation oracle. By rewriting the Constraints (6), we obtain for all levels ℓ, $S\subseteq V$ and $v\in S$ the constraint $\sum_{u\in S}(\text{dist}_V(u,v)-\tilde{\mu}_\ell)w(u)\geq w(V)/k_\ell$.

For any vertex v, the left-hand side of the constraint is minimized when the subset $S_v=\{u\in V\mid \text{dist}_V(v,u)\leq\tilde{\mu}_\ell\}$ is chosen. Consequently, h single-source shortest path computations from every vertex provide a polynomial-time separation oracle for the spreading metrics LP.

The notation introduced next serves the definition of procedure cut-proc in Algorithm 2. A ball $B(v,r)$ in subgraph $G'=(V',E')$ is the set of vertices in V' whose distance to v is strictly less than r, thus $B(v,r) := \{u\in V'\mid \text{dist}_{V'}(u,v)<r\}$. The set $E(v,r)$ denotes the set of edges leaving $B(v,r)$. The volume of $B(v,r)$, denoted by $\text{vol}(v,r)$, is defined by

$$\text{vol}(v,r) := \eta\cdot\tau + \sum_{\substack{(x,y)\in E(v,r),\\ x\in B(v,r)}} c(x,y)(r-\text{dist}_{B(v,r)}(v,x))\ , \tag{7}$$

where $\eta=\frac{1}{hn}$. Note that the volume only considers the edges in the cut, not the edges within the ball, which is different from [5]. The following lemma is a corollary of Even et al. [5].

▶ **Lemma 14.** *Given a subgraph $G'=(V',E')$ that satisfies $w(V')>(1+\varepsilon)w(V)/k_\ell$, and given a spreading metric $\{d(e)\}_e$, procedure cut-proc finds a subset $T\subseteq V'$ that satisfies $w(T)\leq(1+\varepsilon)w(V)/k_\ell$.*

Algorithm 2 cut-proc$(V', E', \{c(e)\}_{e\in E'}, \{d(e)\}_{e\in E'}, \tilde{\mu}_\ell)$

Choose an arbitrary $v \in V'$
$\tilde{r} \leftarrow \frac{\varepsilon}{1+\varepsilon}\tilde{\mu}_\ell$
$T \leftarrow \{v\}$
$v' \leftarrow$ closest vertex to T in $V' \setminus T$
while $c(T) > \frac{1}{\tilde{r}} \cdot \ln(\frac{\mathrm{vol}(v,\tilde{r})}{\mathrm{vol}(v,0)}) \cdot \mathrm{vol}(v, \mathrm{dist}_{V'}(v, v'))$ **do**
  $T \leftarrow T \cup \{v'\}$
  $v' \leftarrow$ closest vertex to T in $V' \setminus T$
end while
Return T

Proof. The proof has been deferred to the full version. ◀

The approximation guarantee on the cost induced by partitions $\mathcal{P}'_1$ to $\mathcal{P}'_h$ is given by the next theorem.

▶ **Theorem 15.** *The sequence $\mathcal{H}' = (\mathcal{P}'_1, \ldots, \mathcal{P}'_h)$ is a $\vec{k}/(1+\varepsilon)$-hierarchical separator with $|\mathcal{P}'_1| \le k_1$ and $\mathrm{cost}_{\vec{\mu}}(\mathcal{H}) \le \alpha \cdot \mathsf{OPT}_{\vec{\mu}}(\vec{k}, G)$, where $\alpha = \mathcal{O}(\log n)$.*

Proof. The fact that $\mathcal{H}'$ is a $\vec{k}/(1+\varepsilon)$-hierarchical separator follows immediately from the construction and Algorithm 2. In particular, Theorem 11 ensures that $|\mathcal{P}'_1| \le k_1$. The cost of $\mathcal{P}'_2$ to $\mathcal{P}'_h$ w.r.t. $\vec{\mu}$ equals $\sum_{\ell=2}^{h} \mu_\ell \mathcal{P}'_\ell$. By Lemma 14, the algorithm constructs sets T_ℓ for $\mathcal{P}'_\ell$ whose cost $c(T_\ell)$ is at most

$$\frac{1+\varepsilon}{\varepsilon\tilde{\mu}_\ell} \cdot \ln\left(\frac{\mathrm{vol}(v_\ell, \varepsilon\tilde{\mu}_\ell/(1+\varepsilon))}{\mathrm{vol}(v_\ell, 0)}\right) \cdot \mathrm{vol}(v_\ell, \mathrm{dist}_{V'}(v_\ell, v'_\ell)) \ , \tag{8}$$

where v_ℓ and v'_ℓ are some vertices in T_ℓ. Recall from the description of the algorithm that the left-hand side of this inequality ignores all edges that have been cut on a higher level or that have been cut previously on the same level. By using the shorthand $\tilde{\mu}_\ell$, we account for cuts on subsequent levels, thus obtaining $\sum_{\ell=2}^{h} \mu_\ell \mathcal{P}_\ell \le \sum_{\ell=2}^{h} \tilde{\mu}_\ell \sum_{T_\ell} c(T_\ell)$.

Observe that $\mathrm{vol}(v, 0) \ge \eta\tau$ and $\mathrm{vol}(v, \varepsilon\tilde{\mu}_\ell/(1+\varepsilon)) \le (1+\eta)\tau$ for all $v \in V$. We apply this to Equation (8) and obtain

$$c(T_\ell) \le \frac{1+\varepsilon}{\varepsilon\tilde{\mu}_\ell} \cdot \ln\left(\frac{1+\eta}{\eta}\right) \cdot \mathrm{vol}(v_\ell, \mathrm{dist}_{V'}(v_\ell, v'_\ell)) \ . \tag{9}$$

It follows that

$$\begin{aligned}
&\sum_{\ell=2}^{h} \sum_{T_\ell} \tilde{\mu}_\ell \cdot c(T_\ell) \le \frac{1+\varepsilon}{\varepsilon} \ln\left(\frac{1+\eta}{\eta}\right) \sum_{\ell=2}^{h} \sum_{T_\ell} \mathrm{vol}(v_\ell, \mathrm{dist}_{V'}(v_\ell, v'_\ell)) \\
&= \frac{1+\varepsilon}{\varepsilon} \ln\left(\frac{1+\eta}{\eta}\right) \sum_{\ell=2}^{h} \sum_{T_\ell} (\eta\tau + \sum_{e \in E(v_\ell, \mathrm{dist}_{V'}(v_\ell, v'_\ell))} c(e)d(e)) \\
&\le \frac{1+\varepsilon}{\varepsilon} \ln\left(\frac{1+\eta}{\eta}\right) (hn\eta\tau + \sum_{\ell=2}^{h} \sum_{T_\ell} \sum_{e} c(e)d(e)) \ .
\end{aligned}$$

We plug in Equation (9) to obtain the first step and use the definition of the volume in Equation (7) for the second step.

We now claim that every edge $e \in E$ appears at most once in the triple sum above. If e is cut by cut-proc for $\mathcal{P}'_\ell$, it is removed from the subgraph for all further iterations of cut-proc

on the same level ℓ. Moreover, edge e is not considered on all remaining levels $\ell' > \ell$, because procedure cut-proc is initiated on these levels with subgraphs of the parts of $\mathcal{P}_\ell$. It follows that the triple sum is upper-bounded by τ.

Using $\eta = \frac{1}{hn}$ and the fact that ε is constant, we conclude that

$$\sum_{\ell=2}^{h} \sum_{T_\ell} \tilde{\mu}_\ell \cdot c(T_\ell) \leq \frac{1+\varepsilon}{\varepsilon} \ln\left(\frac{1+\eta}{\eta}\right)(hn\eta+1)\tau = \mathcal{O}(\log n)\tau \ .$$

Recall that by Theorem 11, the weighted cost of the first partition $\tilde{\mu}_1 \cdot \text{cost}^{\text{sum}}(\mathcal{P}'_1)$ is at most $\mathcal{O}(\log n)$ times the cost of the optimal hierarchical partition $\mathsf{OPT}_{\vec{\mu}}(\vec{k}, G)$. As furthermore $\tau \leq \mathsf{OPT}_{\vec{\mu}}(\vec{k}, G)$, the theorem follows. ◀

In other words, the above theorem states that $\mathcal{H}'$ is an $\mathcal{O}(\log n)$-approximate $\vec{k}/(1+\varepsilon)$-hierarchical separator. Combining Lemma 10 and Theorem 15 gives the following.

▶ **Theorem 16.** *There exists a $(\mathcal{O}(\log n), (1+\varepsilon)h)$ approximation algorithm for Hierarchical Graph Partitioning whose running time is polynomial in h and n.*

References

1. Konstantin Andreev and Harald Räcke. Balanced graph partitioning. *Theory Comput. Syst.*, 39(6):929–939, 2006. doi:10.1007/s00224-006-1350-7.
2. Nikhil Bansal, Uriel Feige, Robert Krauthgamer, Konstantin Makarychev, Viswanath Nagarajan, Joseph (Seffi) Naor, and Roy Schwartz. Min-max graph partitioning and small set expansion. In *Proc. of the 52nd FOCS*, pages 17–26, 2011. doi:10.1109/FOCS.2011.79.
3. Marcin Bienkowski, Miroslaw Korzeniowski, and Harald Räcke. A practical algorithm for constructing oblivious routing schemes. In *Proc. of the 15th SPAA*, pages 24–33, 2003. doi:10.1145/777412.777418.
4. Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM J. Comput.*, 33(4):837–851, 2004. doi:10.1137/S0097539799356265.
5. Guy Even, Joseph (Seffi) Naor, Satish Rao, and Baruch Schieber. Fast approximate graph partitioning algorithms. *SIAM J. Comput.*, 28(6):2187–2214, 1999. Also in *Proc. 8th SODA*, 1997, pp. 639–648. doi:10.1137/S0097539796308217.
6. Andreas E. Feldmann and Luca Foschini. Balanced partitions of trees and applications. In *Proc. of the 29th STACS*, pages 100–111, 2012. doi:10.4230/LIPIcs.STACS.2012.100.
7. Mohammad Taghi Hajiaghayi, Theodore Johnson, Mohammad Reza Khani, and Barna Saha. Hierarchical graph partitioning. In *Proc. of the 26th SPAA*, pages 51–60, 2014. doi:10.1145/2612669.2612699.
8. Chris Harrelson, Kirsten Hildrum, and Satish Rao. A polynomial-time tree decomposition to minimize congestion. In *Proc. of the 15th SPAA*, pages 34–43, 2003. doi:10.1145/777412.777419.
9. Robert Krauthgamer, Joseph (Seffi) Naor, and Roy Schwartz. Partitioning graphs into balanced components. In *Proc. of the 20th SODA*, pages 942–949, 2009. arXiv:soda:1496770.1496872.
10. Robert Krauthgamer, Joseph (Seffi) Naor, Roy Schwartz, and Kunal Talwar. Non-uniform graph partitioning. In *Proc. of the 25th SODA*, pages 1229–1243, 2014. arXiv:soda:2634165.
11. Konstantin Makarychev and Yuri Makarychev. Nonuniform graph partitioning with unrelated weights. In *Proc. of the 41st ICALP*, pages 812–822, 2014. doi:10.1007/978-3-662-43948-7_67.

12 Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proc. of the 40th STOC*, pages 255–264, 2008.

13 Harald Räcke and Chintan Shah. Improved guarantees for tree cut sparsifiers. In *Proc. of the 22nd ESA*, pages 774–785, 2014. doi:10.1007/978-3-662-44777-2_64.