43rd International Symposium on Mathematical Foundations of Computer Science

MFCS 2018, August 27-31, 2018, Liverpool, United Kingdom

Edited by
Igor Potapov
Paul Spirakis
James Worrell

LIPIcs – Vol. 117 – MFCS 2018 www.dagstuhl.de/lipics

Editors

Igor Potapov
Department of Computer Science
University of Liverpool
Potapov@liverpool.ac.uk

Paul Spirakis
Department of Computer Science
University of Liverpool
P.Spirakis@liverpool.ac.uk

James Worrell
Department of Computer Science
University of Oxford
jbw@cs.ox.ac.uk

ACM Classification 2012
Theory of computation

ISBN 978-3-95977-086-6

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/978-3-95977-086-6.

Publication date
August, 2018

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

Digital Object Identifier: 10.4230/LIPIcs.MFCS.2018.0

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

http://www.dagstuhl.de/lipics

Contents

Regular Papers

43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018).
Editors: Igor Potapov, Paul Spirakis, and James Worrell
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

The International Symposium on Mathematical Foundations of Computer Science (MFCS conference series) is a well-established venue for presenting research papers in theoretical computer science. The broad scope of the conference encourages interactions between researchers who might not meet at more specialized venues. The first MFCS conference was organized in 1972 in Jablonna (near Warsaw, Poland). Since then, the conference traditionally moved between the Czech Republic, Slovakia, and Poland. A few years ago, the conference started traveling around Europe: in 2013 it was held in Austria, in 2014 in Hungary, in 2015 in Italy, and most recently, in 2016, the conference returned to Poland and then in 2017 was organized in Denmark. This year the conference has been visiting the United Kingdom for the first time and was organized in Liverpool.

Over 210 abstracts were submitted, of which 185 materialized as papers, of which 84 were finally accepted. The authors of the submitted papers represent nearly 40 countries. The authors first registered their papers' abstracts (by the 20th of April, 2018) and only then their content (by the 24th of April, 2018). This division in two stages has helped with the assignment of the papers to the PC members. Each paper was assigned to three PC members, who reviewed and discussed them thoroughly over a period of nearly six weeks. As the co-chairs of the program committee, we would like to express our deep gratitude to all the committee members for their hard, dedicated work. The quality of the submitted papers was very high and many good papers had to be rejected. The conference featured five invited talks, by Christel Baier (TU Dresden, Germany)m Olivier Bournez (LIX, France), Herbert Edelsbrunner (IST, Austria), Leslie Ann Goldberg (University of Oxford, UK) and Christos H. Papadimitriou (Columbia University, USA):

Christel Baier - On energy conditions and stochastic shortest path problems for Markov Decision Processes
Abstract: Markov decision processes (MDP) are widely used to formalize algorithmic problems where the task is to find a policy for traversing a weighted probabilistic graph structure in a somehow optimal way. Examples are the stochastic shortest-path problem where the goal is to minimize the expected accumulated weight until reaching a target or decision problems for MDPs with energy constraints that, e.g., aims to ensure that almost surely a target will be reached while the accumulated weight ("energy") meets a given bound. The talk will discuss solutions for such and related problems for MDPs with integer weights. These rely on a new classification of so-called end components of MDPs according to their limiting behavior with respect to the accumulated weights. This classification will be used to show that the stochastic shortest path problem is solvable in polynomial time for arbitrary finite-state MDPs, generalizing previous results for sub-classes of MDPs. Furthermore, it will be used to provide algorithms for deciding the existence of a policy ensuring that a weight-bounded (repeated) reachability condition or holds almost surely or with positive probability, and the analogous problems for universal rather than existential quantification over policies.

Olivier Bournez - Descriptive Mathematics and Computer Science with Polynomial Ordinary Differential Equations
Abstract: We will see that many continuous and discrete concepts from mathematics and computer science can be presented using ordinary differential equations. Basically, we will start from the following observation: if you know what 0, 1, -1 are, as well as what an

43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018).
Editors: Igor Potapov, Paul Spirakis, and James Worrell
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

addition and a multiplication are, and if you remember what an ordinary differential equation is, then you can define and program many concepts from Mathematics and Computer Science. In particular we will present/rediscover descriptive complexity, computability and complexity using polynomial ordinary differential equations only. A title for this talk could also be "Programming with Ordinary Differential Equations", as these questions also relate to analog models of computations, and in particular to the 1941 General Purpose Analog Computer of Claude Shannon. In some way, we are rediscovering the forgotten art of their programming, and we are only starting to understand the true power of these very old models.

Herbert Edelsbrunner - Tri-partition of a simplicial complex
Abstract: We prove that for every simplicial complex, K, and every dimension, p, there is a partition of the p-simplices into a maximal p-tree, a maximal p-cotree, and the remaining p-simplices defining the p-th homology of K. Given a monotonic order of the simplices, this tri-partition is unique and can be computed by matrix reduction. Collecting the sets over all monotonic orders, we get matroids over the set of p-simplices (Joint work with Katharina Oelsboeck).

Leslie Ann Goldberg - Computational Complexity and the Independence Polynomial
Abstract: The independence polynomial is one of the most well-studied graph polynomials, arising in combinatorics and computer science. It is also known in statistical physics as the "partition function of the hard-core model". After describing the polynomial, I will tell you something about the complexity of approximating this polynomial, including the now-classical breakthrough results of Weitz and Sly, incursions into the complex plane by Harvey, Srivastava, and Vondrák and by Patel and Regts and finally more recent work using tools from complex analysis by Peters and Regts and also in joint work with Bezakova, Galanis, and Stefankovic.

Christos H. Papadimitriou - A computer scientist thinks about the Brain
Abstract: How does the Brain give rise to the Mind? How do neurons and synapses, molecules and genes, evolution and development, give rise to behavior and cognition, language and intelligence? Despite lightning progress in recording and molecular technology and a deluge of experimental data, we do not seem to get closer to an answer. This is a talk about admiring and appreciating the problem, and proposing a new approach based on a recognized but little studied intermediate level of Brain computation carried out by the synchronous firing of large and highly interconnected sets of neurons called assemblies. We show that assemblies give rise to a novel computational system, and we speculate that they may instrument higher cognitive functions, such as language and math.

We would like to thank them deeply for their contributions and their time. This is the third time that the MFCS proceedings are published in the Dagstuhl/LIPIcs series. We would like to particularly thank Marc Herbstritt and the LIPIcs team for all the help and support. We believe that the cooperation between MFCS and Dagstuhl/LIPIcs in the future will continue to be as seamless and fruitful as ours.

Program Committee

Eric Allender	Rutgers University
Andris Ambainis	University of Latvia
Valerie Berthe	CNRS IRIF
Patricia Bouyer	LSV, CNRS & ENS Cachan, Université Paris Saclay
Jean Cardinal	Universite Libre de Bruxelles (ULB)
Stephanie Delaune	IRISA
Xiaotie Deng	Peking University
Volker Diekert	University of Stuttgart
Nathanaël Fijalkow	CNRS, LaBRI, University of Bordeaux
Dimitris Fotakis	Yahoo Research NY and National Technical University of Athens
Leszek Gasieniec	University of Liverpool
Gregory Gutin	Royal Holloway, University of London
Christoph Haase	University of Oxford
Mika Hirvensalo	University of Turku
Juraj Hromkovic	ETH Zurich
Jarkko Kari	University of Turku
Bakhadyr Khoussainov	The University of Auckland
Lefteris Kirousis	National and Kapodistrian University of Athens
Bartek Klin	University of Warsaw
Adrian Kosowski	IRIF (LIAFA) / Inria Paris
Steve Kremer	INRIA
Antonin Kucera	Masaryk University
Orna Kupferman	Hebrew University
Vitaliy Kurlin	University of Liverpool
Ranko Lazic	The University of Warwick
Vadim Lozin	The University of Warwick
Conrado Martínez	Dept. Computer Science, Univ. Politècnica de Catalunya
Richard Mayr	The University of Edinburgh
Pierre Mckenzie	University of Montreal
George Mertzios	Durham University
Sotiris Nikoletseas	University of Patras
Vangelis Paschos	LAMSADE, University Paris-Dauphine
Giuseppe Persiano	Università degli Studi di Salerno
Igor Potapov	University of Liverpool
Pierre-Alain Reynier	Aix-Marseille Université
Jose Rolim	University of Geneva
Christian Scheideler	University of Paderborn
Maria Serna	Universitat Politècnica de Catalunya
Alexandra Silva	University College London
Paul Spirakis	Research Academic Computer Technology Institute and University of Patras
Philippas Tsigas	School of Computer Science and Engineering, Chalmers University
Prudence W.H. Wong	University of Liverpool
James Worrell	University of Oxford
Moti Yung	Columbia University
Stanislav Živný	University of Oxford

43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018).
Editors: Igor Potapov, Paul Spirakis, and James Worrell
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Additional Reviewers

Aaronson, Scott
Abramsky, Samson
Alecu, Bogdan
Amanatidis, Giorgos
Arad, Itai
Arroyuelo, Diego
Bampis, Evripidis
Barbero, Florian
Basset, Nicolas
Baste, Julien
Belazzougui, Djamal
Berlinkov, Mikhail
Bezakova, Ivona
Bhrushundi, Abhishek
Bilò, Davide
Bodwin, Greg
Boeckenhauer, Hans-Joachim
Boiret, Adrien
Bollig, Benedikt
Bonacina, Ilario
Bouland, Adam
Brazdil, Tomas
Bredariol Grilo, Alex
Brlek, Srecko
Bulatov, Andrei
Burjons Pujol, Elisabet
Busatto-Gaston, Damien
Buss, Sam
Buwaya, Julia
Béal, Marie-Pierre
Cadilhac, Michaël
Camby, Eglantine
Capelli, Florent
Caragiannis, Ioannis
Carayol, Arnaud
Carbonnel, Clément
Carton, Olivier
Chakraborty, Diptarka
Chen, Lin
Chen, Yijia
Chen, Yu-Fang
Chistikov, Dmitry
Chlebikova, Janka
Choffrut, Christian
Clemente, Lorenzo
Clementi, Andrea
Colcombet, Thomas
Cote, Hugo
Courcelle, Bruno
Czerwiński, Wojciech
Dabrowski, Konrad Kazimierz
Das, Bireswar
Daviaud, Laure
Dawar, Anuj
De Marco, Gianluca
Delaplace, Claire
Deligkas, Argyrios
Dereniowski, Dariusz
Diaz, Josep
Didier, Gilles
Dinneen, Michael
Downey, Rod
Doyen, Laurent
Droste, Manfred
Duan, Ran
Dublois, Louis
Dudek, Bartlomiej
Dzamonja, Mirna
Elbassioni, Khaled
Epstein, Leah
Erlebach, Thomas
Escoffier, Bruno
Fagerberg, Rolf
Faran, Rachel
Feldmann, Andreas Emil
Feldmann, Michael
Fellows, Michael
Fernique, Thomas
Ferraioli, Diodato
Fischer, Johannes
Fleischer, Lukas
Fluschnik, Till
Fox, Kyle
Frei, Fabian
Frid, Anna
Fuchs, Michael
Fulla, Peter
Gai, Ling
Galesi, Nicola
Galliani, Pietro
Ganardi, Moses
Ganian, Robert
Garg, Mohit
Gavryushkin, Alex
Gawrychowski, Pawel
Gańczorz, Michał
Genitrini, Antoine
Gharibian, Sevag
Giannakos, Aristotelis
Giannopoulou, Archontia
Gittenberger, Bernhard
Godin, Thibault
Golovach, Petr
Goto, Keisuke
Gouveia, João
Graham-Lengrand, Stéphane
Graça, Daniel
Grellois, Charles
Grohe, Martin
Grossman, Ofer
Groß, Martin
Guillon, Bruno
Guo, Heng
Gurski, Frank
Götte, Thorsten
Göös, Mika
Hadjicostas, Petros
Hague, Matthew
Halava, Vesa
Hamoudi, Yassine
Han, Xin
Harju, Tero
Harutyunyan, Ararat

43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018).
Editors: Igor Potapov, Paul Spirakis, and James Worrell
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Hatzel, Meike
Hell, Pavol
Helouet, Loic
Hertrampf, Ulrich
Hinnenthal, Kristian
Hlineny, Petr
Hofman, Piotrek
Holmes, Randall
Iacono, John
Ikenmeyer, Christian
Ito, Takehiro
Jain, Sanjay
Jansson, Jesper
Jerrum, Mark
Jeż, Artur
Ji, Zhengfeng
Johnson, Matthew
Jonsson, Peter
Jugé, Vincent
Jurdzinski, Marcin
Kaaser, Dominik
Kabanets, Valentine
Kacem, Imed
Kalnishkan, Yuri
Kamali, Shahin
Karhumaki, Juhani
Katsikarelis, Ioannis
Király, Csaba
Klaska, David
Kling, Peter
Klíma, Ondřej
Koiran, Pascal
Kolb, Christina
Kolokolova, Antonina
Komm, Dennis
Kontonis, Vasilis
Kontopoulou, Eugenia
Korpelainen, Nicholas
Korwar, Arpita
Koucky, Michal
Kretinsky, Jan
Kufleitner, Manfred
Kuinke, Philipp
Kulikov, Alexander
Kumar, Mrinal
Kunc, Michal
Kuperberg, Greg
Kwisthout, Johan
Kwon, O-Joung
Kyropoulou, Maria
Labbé, Sébastien
Lampis, Michael
Lapinskas, John
Lauria, Massimo
Le Gall, Francois
Lenzen, Christoph
Levy, Caleb
Lhote, Nathan
Lianeas, Thanasis
Limouzy, Vincent
Lin, Anthony Widjaja
Lingas, Andrzej
Livieratos, John
Loff, Bruno
Lohrey, Markus
Lombardy, Sylvain
Lubiw, Anna
Lucarelli, Giorgio
Lumbroso, Jérémie
Malod, Guillaume
Malyshev, Dmitriy
Manea, Florin
Maneth, Sebastian
Markey, Nicolas
Marsault, Victor
Martin, Barnaby
Mary, Arnaud
Maubert, Bastien
Mayordomo, Elvira
Mazowiecki, Filip
Medina, Moti
Mengel, Stefan
Michail, Othon
Milanič, Martin
Molter, Hendrik
Monmege, Benjamin
Monteil, Thierry
Morris, Rob
Moser, Philippe
Mukhopadhyay, Sagnik
Nagaj, Daniel
Nichterlein, André
Niemi, Valtteri
Nikolopoulos, Stavros
Nishimura, Naomi
Niwinski, Damian
Nowotka, Dirk
Obdrzalek, Jan
Ochremiak, Joanna
Ohlmann, Pierre
Ordyniak, Sebastian
Ossona de Mendez, Patrice
Otachi, Yota
Oveis Gharan, Shayan
Paesani, Giacomo
Papadigenopoulos, Vasileios-Orestis
Pasquale, Francesco
Paulusma, Daniel
Paz, Ami
Perifel, Sylvain
Petersen, Holger
Pietracaprina, Andrea
Pilipczuk, Marcin
Pissis, Solon
Place, Thomas
Portugal, Renato
Potechin, Aaron
Pouly, Amaury
Prūsis, Krišjānis
Puppis, Gabriele
Pérez-Lantero, Pablo
Radzik, Tomasz
Ralaivaosaona, Dimbinaina
Rao, Anup
Rao, Michael
Raptopoulos, Christoforos
Rehak, Vojtech
Reimann, Jan
Reiter, Fabian
Remila, Eric
Rescigno, Adele
Reutenauer, Christophe
Rossmanith, Peter
Rubin, Sasha
S, Krishna
Saarela, Aleksi
Saivasan, Prakash
Salamanca, Julian

Salo, Ville
Santhanam, Rahul
Sassolas, Mathieu
Saule, Erik
Saurabh, Saket
Schnoebelen, Philippe
Schramm, Tselil
Schwartz, Oded
Schwoon, Stefan
Serre, Olivier
Setzer, Alexander
Sherstov, Sasha
Shioura, Akiyoshi
Shitov, Yaroslav
Siebertz, Sebastian
Sklinos, Rizos
Skoulakis, Stratis
Skrzypczak, Michał
Slivovsky, Friedrich
Spoerhase, Joachim
Stamoulis, Georgios
Starikovskaya, Tatiana
Stojakovic, Milos
Straubing, Howard
Strozecki, Yann
Taati, Siamak
Tabareau, Nicolas
Takahashi, Yasuhiro
Talbot, Jean-Marc
Tamaki, Suguru
Tamir, Tami
Thilikos, Dimitrios
Todinca, Ioan
Totzke, Patrick
Toulouse, Sophie
Turowski, Krzysztof
Vaananen, Jouko
Valeriote, Matt
Van Den Bogaard, Marie
Vardi, Gal
Vaux, Lionel
Venturini, Rossano
Villagra, Marcos
Villevalois, Didier
Viola, Emanuele
Volkov, Mikhail
Wahlström, Magnus
Wang, Kai
Wasa, Kunihiro
Watrous, John
Wehner, David
Weimann, Oren
Weiss, Armin
Wimmer, Karl
Xiao, Mingy
Xu, Rupei
Yamamoto, Masaki
Yamanaka, Katsuhisa
Zamaraev, Viktor
Zamaraeva, Elena
Zetzsche, Georg
Zheng, Shenggen
Zhou, Zixin
Zimmermann, Martin
Zolotykh, Nikolai
Zou, Mengchuan

Steering Committee

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

43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018).
Editors: Igor Potapov, Paul Spirakis, and James Worrell
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Consensus Strings with Small Maximum Distance and Small Distance Sum

Laurent Bulteau
Université Paris-Est, LIGM (UMR 8049), CNRS, ENPC, ESIEE Paris, UPEM, F-77454, Marne-la-Vallée, France
laurent.bulteau@u-pem.fr

Markus L. Schmid
Fachbereich 4 – Abteilung Informatikwissenschaften, Universität Trier, 54286 Trier, Germany
mlschmid@mlschmid.de

Abstract

The parameterised complexity of consensus string problems (Closest String, Closest Substring, Closest String with Outliers) is investigated in a more general setting, i. e., with a bound on the maximum Hamming distance *and* a bound on the sum of Hamming distances between solution and input strings. We completely settle the parameterised complexity of these generalised variants of Closest String and Closest Substring, and partly for Closest String with Outliers; in addition, we answer some open questions from the literature regarding the classical problem variants with only one distance bound. Finally, we investigate the question of polynomial kernels and respective lower bounds.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness, Theory of computation → Fixed parameter tractability, Theory of computation → W hierarchy

Keywords and phrases Consensus String Problems, Closest String, Closest Substring, Parameterised Complexity, Kernelisation

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.1

1 Introduction

Consensus string problems have the following general form: given input strings $S = \{s_1, \ldots, s_k\}$ and a distance bound d, find a string s with distance at most d from the input strings. With the Hamming distance as the central distance measure for strings, there are two obvious types of distance between a single string and a set S of strings: the maximum distance between s and any string from S (called *radius*) and the sum of all distances between s and strings from S (called *distance sum*). The most basic consensus string problem is Closest String, where we get a set S of k length-ℓ strings and a bound d, and ask whether there exists a length-ℓ *solution string* s with radius at most d. This problem is NP-complete (see [16]), but fixed-parameter tractable for many variants (see [17]), including the parameterisation by d, which in biological applications can often be assumed to be small (see [12, 18]). A classical extension is Closest Substring, where the strings of S have length *at most* ℓ, the solution string must have a given length m and the radius bound d is w. r. t. some length-m substrings of the input strings. A parameterised complexity analysis (see [13, 14, 20]) has shown Closest Substring to be harder than Closest String. If we bound the distance sum instead of the radius, then Closest String collapses to a trivial problem, while Closest Substring, which is then called Consensus Patterns, remains

NP-complete. Closest String with Outliers is a recent extension, which is defined like Closest String, but with the possibility to ignore a given number of t input strings.

The main motivation for consensus string problems comes from the important task of finding similar regions in DNA or other protein sequences, which arises in many different contexts of computational biology, e. g., universal PCR primer design [9, 18, 19, 23], genetic probe design [18], antisense drug design [8, 18], finding transcription factor binding sites in genomic data [25], determining an unbiased consensus of a protein family [3], and motif-recognition [18,21,22]. The consensus string problems are a formalisation of this computational task and most variants of them are NP-hard. However, due to their high practical relevance, it is necessary to solve them despite their intractability, which has motivated the study of their approximability, on the one hand, but also their fixed-parameter tractability, on the other (see the survey [6] for an overview of the parameterised complexity of consensus string problems). This work is a contribution to the latter branch of research.

Problem Definition. Let Σ be a finite alphabet, Σ^* be the set of all strings over Σ, including the empty string ε and $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. For $w \in \Sigma^*$, $|w|$ is the length of w and, for every i, $1 \leq i \leq |w|$, by $w[i]$, we refer to the symbol at position i of w. For every $n \in \mathbb{N} \cup \{0\}$, let $\Sigma^n = \{w \in \Sigma^* \mid |w| = n\}$ and $\Sigma^{\leq n} = \bigcup_{i=0}^{n} \Sigma^i$. By $\preceq$, we denote the *substring relation* over the set of strings, i. e., for $u, v \in \Sigma^*$, $u \preceq v$ if $v = xuy$, for some $x, y \in \Sigma^*$. We use the concatenation of sets of strings as usually defined, i. e., for $A, B \subseteq \Sigma^*$, $A \cdot B = \{uv \mid u \in A, v \in B\}$.

For strings $u, v \in \Sigma^*$ with $|u| = |v|$, $\mathsf{d_H}(u, v)$ is the *Hamming distance* between u and v. For a multi-set $S = \{u_i \mid 1 \leq i \leq n\} \subseteq \Sigma^\ell$ and a string $v \in \Sigma^\ell$, for some $\ell \in \mathbb{N}$, the *radius of S (w. r. t. v)* is defined by $\mathsf{r_H}(v, S) = \max\{\mathsf{d_H}(v, u) \mid u \in S\}$ and the *distance sum of S (w. r. t. v)* is defined by $\mathsf{s_H}(v, S) = \sum_{u \in S} \mathsf{d_H}(v, u)$.[1] Next, we state the problem (r, s)-Closest String in full detail, from which we then derive the other considered problems:

(r, s)-Closest String
Instance: Multi-set $S = \{s_i \mid 1 \leq i \leq k\} \subseteq \Sigma^\ell$, $\ell \in \mathbb{N}$, and integers $d_r, d_s \in \mathbb{N}$.
Question: Is there an $s \in \Sigma^\ell$ with $\mathsf{r_H}(s, S) \leq d_r$ and $\mathsf{s_H}(s, S) \leq d_s$?

For (r, s)-Closest Substring, we have $S \subseteq \Sigma^{\leq \ell}$ and an additional input integer $m \in \mathbb{N}$, and we ask whether there is a multi-set $S' = \{s'_i \mid s'_i \preceq s_i, 1 \leq i \leq k\} \subseteq \Sigma^m$ with $\mathsf{r_H}(s, S') \leq d_r$ and $\mathsf{s_H}(s, S') \leq d_s$. For (r, s)-Closest String with Outliers (or (r, s)-Closest String-wo for short) we have an additional input integer $t \in \mathbb{N}$, and we ask whether there is a multi-set $S' \subseteq S$ with $|S'| = k - t$ such that $\mathsf{r_H}(s, S') \leq d_r$ and $\mathsf{s_H}(s, S') \leq d_s$. We also call (r, s)-Closest String the *general variant* of Closest String, while (r)-Closest String and (s)-Closest String denote the variants, where the only distance bound is d_r or d_s, respectively; we shall also call them the (r)- and (s)-*variant* of Closest String. Analogous notation apply to the other consensus string problems. The problem names that are also commonly used in the literature translate into our terminology as follows: Closest String = (r)-Closest String, Closest Substring = (r)-Closest Substring, Consensus Patterns = (s)-Closest Substring and Closest String-wo = (r)-Closest String-wo.

[1] Note that we slightly abuse notation with respect to the subset relation: for a multi-set A and a set B, $A \subseteq B$ means that $A' \subseteq B$, where A' is the set obtained from A by deleting duplicates; for multi-sets A, B, $A \subseteq B$ is defined as usual. Moreover, whenever it is clear from the context that we talk about multi-sets, we also simply use the term *set*.

The motivation for our more general setting with respect to the bounds d_r and d_s is the following. While the distance measures of radius and distance sum are well-motivated, they have, if considered individually, also obvious deficiencies. In the distance sum variant, we may consider strings as close enough that are very close to some, but totally different to the other input strings. In the radius variant, on the other hand, we may consider strings as too different, even though they are very similar to all input strings except one, for which the bound is exceeded by only a small amount. Using an upper bound on the distance per each input string and an upper bound on the total sum of distances caters for these cases.[2]

For any problem K, by $K(p_1, p_2, \ldots)$, we denote the variant of K parameterised by the parameters $p_1, p_2, \ldots$. For unexplained concepts of parameterised complexity, we refer to the textbooks [7, 10, 15].

Known Results. In contrast to graph problems, where interesting parameters are often hidden in the graph structure, string problems typically contain a variety of obvious, but nevertheless interesting parameters that could be exploited in terms of fixed-parameter tractability. For the consensus string problems these are the number of input strings k, their length ℓ, the radius bound d_r, the distance sum bound d_s, the alphabet size $|\Sigma|$, the substring length m (in case of (r,s)-Closest Substring), the number of *outliers* t and *inliers* $k-t$ (in case of (r,s)-Closest String-wo). This leads to a large number of different parameterisations, which justifies the hope for fixed-parameter tractable variants.

The parameterised complexity (w. r. t. the above mentioned parameters) of the radius as well as the distance sum variant of Closest String and Closest Substring has been settled by a sequence of papers (see [13, 14, 16, 17, 20] and, for a survey, [6]), except (s)-Closest Substring with respect to parameter ℓ, which has been neglected in these papers and mentioned as an open problem in [24], in which it is shown that the fixed-parameter tractability results from (r)-Closest String carry over to (r)-Closest Substring, if we additionally parameterise by $(\ell - m)$. The parameterised complexity analysis of the radius variant of Closest String with Outliers has been started more recently in [5] and, to the knowledge of the authors, the distance sum variant has not yet been considered.

The parameterised complexity of the general variants, where we have a bound on both the radius and the distance sum, has not yet been considered in the literature. While there are obvious reductions from the (r)- and (s)-variants to the general variant, these three variants describe, especially in the parameterised setting, rather different problems.

Our Contribution. In this work, we answer some open questions from the literature regarding the (r)- and (s)-variants of the consensus string problems, and we initiate the parameterised complexity analysis of the general variants.

We extend all the FPT-results from (r)-Closest String to the general variant; thus, we completely settle the fixed-parameter tractability of (r,s)-Closest String. While for some parameterisations, this is straightforward, the case of parameter d_r follows from a non-trivial extension of the known branching algorithm for (r)-Closest String(d_r) (see [17]).

For (r,s)-Closest Substring, we classify all parameterised variants as being in FPT or W[1]-hard, which is done by answering the open question whether (s)-Closest Substring(ℓ) is in FPT (see [24]) in the negative (which also settles the parameterised complexity of (s)-Closest Substring) and by slightly adapting the existing FPT-algorithms.

[2] To the knowledge of the authors, optimising both the radius and the distance sum has been considered first in [1], where algorithms for the special case $k = 3$ are considered.

Regarding (r,s)-Closest String-wo, we solve an open question from [5] w. r. t. the radius variant, we show W[1]-hardness for a strong parameterisation of the (s)-variant, we show fixed-parameter tractability for some parameter combinations of the general variant and, as our main result, we present an FPT-algorithm (for the general variant) for parameters d_r and t (which is the same algorithm that shows (r,s)-Closest String$(d_r) \in$ FPT mentioned above). Many other cases are left open for further research.

Finally, we investigate the question whether the fixed-parameter tractable variants of the considered consensus string problems allow polynomial kernels; thus, continuing a line of work initiated by Basavaraju et al. [2], in which kernelisation lower bounds for (r)-Closest String and (r)-Closest Substring are proved. Our respective main result is a cross-composition from (r)-Closest String into (r)-Closest String-wo.

Due to space constraints, proofs for results marked with $(*)$ are omitted.

2 Closest String and Closest String-wo

In this section, we investigate (r,s)-Closest String and (r,s)-Closest String-wo (and their (r)- and (s)-variants) and we shall first give some useful definitions.

It will be convenient to treat a set $S = \{s_i \mid 1 \leq i \leq k\} \subseteq \Sigma^\ell$ as a $k \times \ell$ matrix with entries from Σ. By the term *column of* S, we refer to the transpose of a column of the matrix S, which is an element from Σ^k; thus, the introduced string notations apply, e. g., if c is the i^{th} column of S, then $c[j]$ corresponds to $s_j[i]$. A string $s \in \Sigma^\ell$ is a *majority string* (*for a set* $S \subseteq \Sigma^\ell$) if, for every i, $1 \leq i \leq \ell$, $s[i]$ is a symbol with majority in the i^{th} column of S. Obviously, $\mathsf{s_H}(s,S) = \min\{\mathsf{s_H}(s',S) \mid s' \in \Sigma^\ell\}$ if and only if s is a majority string for S. We call a string $s \in \Sigma^\ell$ *radius optimal* or *distance sum optimal* (*with respect to a set* $S \subseteq \Sigma^\ell$) if $\mathsf{r_H}(s,S) = \min\{\mathsf{r_H}(s',S) \mid s' \in \Sigma^\ell\}$ or $\mathsf{s_H}(s,S) = \min\{\mathsf{s_H}(s',S) \mid s' \in \Sigma^\ell\}$, respectively.

It is a well-known fact that (r)-Closest String allows FPT-algorithms for any of the single parameters k, d_r or ℓ, and it is still NP-hard for $|\Sigma| = 2$ (see [17]). While the latter hardness result trivially carries over to (r,s)-Closest String (by setting $d_s = k\,d_r$), we have to modify the FPT-algorithms for extending the fixed-parameter tractability results to (r,s)-Closest String. We start with parameter k, for which we can extend the ILP-approach that is used in [17] to show (r)-Closest String$(k) \in$ FPT.

▶ **Theorem 1 (*).** (r,s)-Closest String$(k) \in$ FPT.

Next, we consider the parameter d_r. For the (r)-variant of (r,s)-Closest String, the fixed-parameter tractability with respect to d_r is shown in [17] by a branching algorithm, which proved itself as rather versatile: it has successfully been extended in [5] to (r)-Closest String-wo(d_r,t) and in [24] to (r)-Closest Substring$(d_r,(\ell-m))$.

We propose an extension of the same branching algorithm, that allows for a bound d_s on the distance sum; thus, it works for (r,s)-Closest String(d_r). In fact, we prove in Theorem 7 an even stronger result, where we also extend the algorithm to exclude up to t outlier strings from the input set S, i. e., we extend it to the problem (r,s)-Closest String-wo(d_r,t). Since Theorem 3 can therefore be seen as a corollary of this result by taking $t = 0$, we only give an informal description of a direct approach that solves (r,s)-Closest String(d_r) (and refer to Theorem 7 for a formal proof).

The core idea is to apply the branching algorithm starting with the majority string for the input set S, instead of any random string from S. Then, as in [17], the algorithm would replace some characters of the current string with characters of the solution string. This way, it can be shown that the distance sum of the current string is always a lower bound of the

Table 1 Results for (r,s)-Closest String.

k	d_r	d_s	$\lvert\Sigma\rvert$	ℓ	Result	Note/Ref.
p	–	–	–	–	FPT	Thm. 1
–	**p**	–	–	–	FPT	Thm. 3
–	–	**p**	–	–	FPT	Cor. 4
–	–	–	2	–	NP-hard	from (r)-variant [16]
–	–	–	–	**p**	FPT	Cor. 4

distance sum of the optimal string, which allows to cut any branch where the distance sum goes beyond the threshold d_s. We prove the following lemma, which allows to bound the depth of the search tree (and shall also be used in the proof of Theorem 7 later on):

▶ **Lemma 2 (*).** *Let $S \subseteq \Sigma^\ell$, $s \in \Sigma^\ell$ such that $\mathsf{r}_\mathsf{H}(s,S) \leq d_r$, and let s_m be a majority string for S. Then $\mathsf{d}_\mathsf{H}(s_m,s) \leq 2d_r$.*

▶ **Theorem 3.** (r,s)-Closest String$(d_r) \in$ FPT.

Obviously, we can assume $d_r \leq \ell$ and we can further assume that every column of S contains at least two different symbols (all columns without this property could be removed), which implies $\mathsf{s}_\mathsf{H}(s_i,S) \geq \ell$ for every $s \in \Sigma^\ell$; thus, we can assume $\ell \leq d_s$. Consequently, we obtain the following corollary:

▶ **Corollary 4.** (r,s)-Closest String$(\ell) \in$ FPT, (r,s)-Closest String$(d_s) \in$ FPT.

This completely settles the parameterised complexity of (r,s)-Closest String with respect to parameters k, d_r, d_s, $|\Sigma|$ and ℓ; recall that the (r)-variant is already settled, while the (s)-variant is trivial.

2.1 (r, s)-Closest String-wo

We now turn to the problem (r,s)-Closest String-wo and we first prove several fixed-parameter tractability results for the general variant; in Sec. 2.2, we consider the (r)- and (s)-variants separately.

First, we note that solving an instance of (r,s)-Closest String-wo(k) can be reduced to solving $f(k)$ many (r,s)-Closest String(k)-instances, which, due to the fixed-parameter tractability of the latter problem, yields the fixed-parameter tractability of the former.

▶ **Theorem 5 (*).** (r,s)-Closest String-wo$(k) \in$ FPT.

If the number $k-t$ of inliers exceeds d_s, then an (r,s)-Closest String-wo-instance becomes easily solvable; thus, $k-t$ can be bounded by d_s, which yields the following result:

▶ **Theorem 6 (*).** (r,s)-Closest String-wo$(d_s,t) \in$ FPT.

The algorithm introduced in [17] to prove (r)-Closest String$(d_r) \in$ FPT has been extended in [5] with an additional branching that guesses whether a string s_j should be considered an outlier or not; thus, yielding fixed-parameter tractability of (r)-Closest String-wo(d_r,t). We present a non-trivial extension of this algorithm, with a carefully selected starting string, to obtain the fixed-parameter tractability of (r,s)-Closest String-wo(d_r,t) (and, as explained in Section 2, also of (r,s)-Closest String(d_r)):

▶ **Theorem 7.** (r,s)-Closest String-wo$(d_r,t) \in$ FPT.

Input:		$s_1 =$	d b a d d c b c d b b d b b				
$d_r = 5$		$s_2 =$	d a a a a c b c d c c d b d				
$d_s = 14$		$s_3 =$	d a a d d a b c a c c d b d				
$t = 2$		$s_4 =$	a a c d a c c d c c c a b d				
		$s_5 =$	a a c d a a b d a c c a d d				
$D = 10$		$s_6 =$	a c a a a a b c d d b a d d				
Step	S'	t	s'		d	$\mathsf{r_H}(s', S')$	action
1	$\{s_1, s_2, \ldots, s_6\}$	2	◇ a ◇ ◇ ◇ ◇ b ◇ ◇ c ◇ ◇ ◇ ◇ d		20	13	$s[3] \leftarrow s_1[3]$
2	$\{s_1, s_2, \ldots, s_6\}$	2	◇ a a ◇ ◇ ◇ b ◇ ◇ c ◇ ◇ ◇ ◇ d		19	12	$s[12] \leftarrow s_1[12]$
3	$\{s_1, s_2, \ldots, s_6\}$	2	◇ a a ◇ ◇ ◇ b ◇ ◇ c ◇ d ◇ d		18	11	remove s_6
4	$\{s_1, s_2, \ldots, s_5\}$	1	◇ a a ◇ ◇ ◇ b ◇ ◇ c ◇ d ◇ d		18	11	$s[6] \leftarrow s_1[6]$
5	$\{s_1, s_2, \ldots, s_5\}$	1	◇ a a ◇ ◇ c b ◇ ◇ c ◇ d ◇ d		17	10	remove s_5
6	$\{s_1, \ldots, s_4\}$	0	◇ a a ◇ ◇ c b ◇ ◇ c ◇ d ◇ d		17	10	
			$s'' =$	d a a d a c b c d c c d b d			$s[7] \leftarrow s_4[7]$
7	$\{s_1, \ldots, s_4\}$	0	◇ a a ◇ ◇ c c ◇ ◇ c ◇ d ◇ d		16	10	
			$s'' =$	d a a d a c c c d c c d b d			return S', s''

Figure 1 Example for Algorithm 1 on an instance of (r, s)-Closest String-wo. The shown steps correspond to one branch that yields a correct solution. The algorithm starts with the majority string where disputed characters are replaced by ◇. At each step, the algorithm either inserts a character from an input string at maximal distance from s' (note that even non-disputed characters may be replaced), or removes one such string. When $t = 0$, it is checked whether the completion s'' of s' is a correct solution. At step 7, we return a solution with $\mathsf{r_H}(s'', S') = 5$ and $\mathsf{s_H}(s'', S') = 14$.

Proof. Let (S, d_s, d_r, t) be a positive instance of (r, s)-Closest String-wo(d_r, t) with $k \geq 5t$ (otherwise k can be considered as a parameter). A character x is *frequent in column i* if it has at least as many occurrences as the majority character minus t (thus, for any $S' \subseteq S$, $|S'| \geq |S| - t$, all majority characters for S' are frequent characters). A column i is *disputed* if it contains at least two frequent characters. Let D be the number of disputed columns.

Let (S^*, s^*) be a solution for this instance. In a disputed column i, no character occurs more than $\frac{k+t}{2}$ times, hence, among the $k - t$ strings of S^*, there are at least $(k-t) - \frac{k+t}{2} = \frac{k-3t}{2}$ mismatches at position i. The disputed columns thus introduce at least $D\frac{k-3t}{2}$ mismatches. Since the overall number of mismatches is upper-bounded by $d_r(k-t)$, we have $D \leq \frac{2d_r(k-t)}{k-3t} = 2d_r\left(1 + \frac{2t}{k-3t}\right)$, and, with $k \geq 5t$, the upper-bound $D \leq 4d_r$ follows.

We introduce a new character $\diamond \notin \Sigma$. A string $s' \in (\Sigma \cup \{\diamond\})^\ell$ is a *lower bound* for a solution s^*, if, for every i such that $s'[i] \neq s^*[i]$, either i is a disputed column and $s'[i] = \diamond$, or i is not disputed and $s'[i]$ is the majority character for column i of S^* (which is equal to the majority character for column i of S). Intuitively speaking, whenever a character $s'[i]$ differs from $s^*[i]$, it is the majority character of its column (except for disputed columns in which we use an "undecided" character $\diamond$). Finally, the *completion for S'* of a string $s' \in (\Sigma \cup \{\diamond\})^*$ is the string obtained by replacing each occurrence of $\diamond$ by a majority character of the corresponding column in S'.

We now prove that Algorithm 1 solves (r, s)-Closest String-wo in time at most $O^*((d_r+1)^{6d_r}2^{6d_r+t})$, using the following three claims (see Figure 1 for an example).

Claim 1. Any call to Solve Closest String-wo(S', t, s', d) always returns after a time $O^*((d_r+1)^d 2^{d+t})$.

ALGORITHM 1: Solve Closest String-wo.

Input : $S' \subseteq S$, $t \in \mathbb{N}$, $s' \in (\Sigma \cup \{\diamond\})^\ell$, $d \in \mathbb{N}$
Output : a pair (S^*, s^*) or the symbol ∇

1 **if** $t = 0$ **then**
2 $s'' =$ completion of s' in S';
3 **if** $\mathsf{s_H}(s'', S') \leq d_s$, and $\mathsf{r_H}(s'', S') \leq d_r$ **then** return (S', s'');
4 **if** $d = 0$ **then** return ∇;
5 Let $s_j \in S'$ be such that $\mathsf{d_H}(s', s_j)$ is maximal;
6 **if** $t > 0$ **then**
7 $sol =$ Solve Closest String-wo$(S' \setminus \{s_j\}, t-1, s', d)$;
8 **if** $sol \neq \nabla$ **then** return sol;
9 **if** $d > 0$ **then**
10 Let $I \subseteq \{1, \ldots, \ell\}$ contain $d_r + 1$ indices i s. t. $s'[i] \neq s_j[i]$ (or all indices if $\mathsf{d_H}(s_j, s') \leq d_r$);
11 **for** $i \in I$ **do**
12 $s'' = s'$, $s''[i] = s_j[i]$;
13 $sol =$ Solve Closest String-wo$(S', t, s'', d-1)$;
14 **if** $sol \neq \nabla$ **then** return sol;
15 return ∇;

Proof of Claim 1. We prove this running time by induction: if $d = t = 0$, then the function returns in Line 3 or 4; thus, it returns after polynomial time. Otherwise, it performs at most d_r+1 recursive calls with parameters $(d-1, t)$, and one recursive call with parameters $(d, t-1)$. By induction, the complexity of this step is $O^*((d_r+1)(d_r+1)^{d-1}2^{d+t-1} + (d_r+1)^d 2^{d+t-1}) = O^*((d_r+1)^d 2^{d+t})$. ◀

A tuple (S', t', s', d) is *valid* if $|S'| - t' = |S| - t$, there exists an optimal solution (S^*, s^*) for which $S^* \subseteq S'$, $|S^*| = |S'| - t'$, $\mathsf{d_H}(s', s^*) \leq d$, and s' is a lower bound for s^*. A call of the algorithm is *valid* if its parameters form a valid tuple, its *witness* is the pair (S^*, s^*).

Claim 2. Any valid call to Solve Closest String-wo either directly returns a solution or performs at least one recursive valid call.

Proof of Claim 2. Let $S' \subseteq \Sigma^\ell$, $t' \geq 0$, $s' \in (\Sigma \cup \{\diamond\})^\ell$, and $d \geq 0$. Consider the call to Solve Closest String-wo(S', t', s', d). Assume it is valid, with witness (S^*, s^*).

Case 1. If $d = t' = 0$, then $s^* = s'$ and $S^* = S'$. The completion s'' of s' is exactly s', and since (S', s') is a solution, it satisfies the conditions of Line 3 and is returned on Line 3.

Case 2. If $t' = 0$ and $\forall s \in S' : \mathsf{d_H}(s, s') \leq d_r$. Then $S^* = S'$ and s' is a lower bound for s^*. Let s'' be the completion of s'. We show that $\mathsf{s_H}(s'', S') \leq \mathsf{s_H}(s^*, S') \leq d_s$. Indeed, consider any column i with $s''[i] \neq s^*[i]$. Either $s'[i] = \diamond$, in which case $s''[i]$ is the majority character for column i of S', or $s'[i] \neq \diamond$, in which case by the definition of lower bound, i is not a disputed column and $s'[i] = s''[i]$ contains the only frequent character of this column, which is the majority character for S'. In both cases, $s''[i]$ is a majority character for S' in any column where it differs from s^*; thus, it satisfies the upper-bound on the distance sum. Since it also satisfies the distance radius (by the case hypothesis: $\mathsf{d_H}(s, s'') \leq \mathsf{d_H}(s, s') \leq d_r$ for all $s \in S'$), it satisfies the conditions of Line 3; thus, solution (S', s'') is returned on Line 3.

In the following cases, we can thus assume that the algorithm reaches Line 5. Indeed, if it returns on Line 3 then it returns a solution, and if it returns on Line 4 then we have $d = t' = 0$, which is dealt in Case 1 above (the algorithm may not return on this line when it has a valid input). We can thus define s_j to be the string selected in Line 5.

Case 3. $s_j \in S' \setminus S^*$. Then in particular $t' > 0$; and since $S^* \subseteq S' \setminus \{s_j\}$, the recursive call in Line 7 is valid, with the same witness (S^*, s^*).

Case 4. $s_j \in S^*$, $d = 0$ and $t' > 0$. Then $s' = s^*$, let s'_j be any string of $S' \setminus S^*$, and $S^+ = S^* \setminus \{s'_j\} \cup \{s_j\}$. Then the pair (S^+, s^*) is a solution, since $\mathsf{d_H}(s^*, s'_j) \leq \mathsf{d_H}(s^*, s_j)$ by definition of s_j. Thus the recursive call on Line 7 is valid, with witness (S^+, s^*).

Case 5. $s_j \in S^*$, $d > 0$ and $\mathsf{d_H}(s_j, s') > d_r$. Consider the set I defined in Line 10. I has size $d_r + 1$, hence there exists $i_0 \in I$ such that $s_j[i_0] = s^*[i_0]$. Then the recursive call with parameters $(S', t, s'', d-1)$ in Line 13 with $i = i_0$ is valid with the same witness (S^*, s^*). Indeed, s'' is obtained from s' by setting $s''[i_0] = s^*[i_0] \neq s'[i_0]$, hence, all mismatches between s'' and s^* already exist between s' and s^*, which implies that s'' is still a lower bound for s^*. Moreover, $\mathsf{d_H}(s'', s^*) = \mathsf{d_H}(s', s^*) - 1 \leq d - 1$.

From now on, we can assume that $d > 0$ and $t' > 0$. Indeed, $d = 0$ is dealt with in cases 1, 3 and 4, and $t' = 0, d > 0$ is dealt with in cases 2 and 5. Moreover, with cases 3 and 5, we can assume that $s_j \in S^*$ and $\mathsf{d_H}(s_j, s') \leq d_r$ (i.e. $\mathsf{d_H}(s, s') \leq d_r$ for all $s \in S^*$).

Case 6. There exists i_0 such that $s_j[i_0] = s^*[i_0] \neq s'[i_0]$. Then again consider the set I defined in Line 10. Since $\mathsf{d_H}(s_j, s') \leq d_r$, we have $i_0 \in I$, and, with the same argument as in Case 5, there is a valid recursive call in Line 13 when $i = i_0$.

Case 7. For all i, $s_j[i] \neq s'[i] \Rightarrow s_j[i] \neq s^*[i]$. In this case no character from s_j can be used to improve our current solution, so the character switching procedure Line 13 will not improve the solution, but still s_j is part of our witness set S^*, so it is not clear a priori that we can remove s_j from our current solution, i.e. that the recursive call on Line 7 is valid.

We handle this situation as follows. Let s^+ be obtained from s' by filling the $\diamond$-positions of s' with the corresponding symbols of s^*. We now show that (S^*, s^+) is a solution. To this end, let $s \in S^*$. For every i, $1 \leq i \leq \ell$, if $s[i] \neq s^+[i]$, then either $s'[i] = \diamond$ or $s'[i] \in \Sigma$ with $s'[i] = s^+[i]$. In both cases, we have $s[i] \neq s'[i]$, which implies $\mathsf{d_H}(s, s^+) \leq \mathsf{d_H}(s, s') \leq d_r$, i. e., the radius is satisfied. Regarding the distance sum, we note that if $s^+[i] \neq s^*[i]$, then, since occurrences of $\diamond$ of s' have been replaced by the corresponding symbol from s^*, $s'[i] \neq \diamond$, which, by the definition of lower bound, implies that $s^+[i] = s'[i]$ is the majority character for column i of S^*. Consequently, $\sum_{s \in S^*} \mathsf{d_H}(s^+[i], s[i]) \leq \sum_{s \in S^*} \mathsf{d_H}(s^*[i], s[i])$, which implies $\mathsf{s_H}(s^+, S^*) \leq \mathsf{s_H}(s^*, S^*) \leq d_s$.

Having defined a new solution string s^+ (with respect to S^*), we now prove that s^+ is also a solution string with respect to $S^+ = (S^* \setminus \{s_j\}) \cup \{s'_j\}$, where s'_j is any string of $S' \setminus S^*$. To this end, we prove that $\mathsf{d_H}(s'_j, s^+) \leq \mathsf{d_H}(s_j, s^+)$; together with the fact that $\mathsf{d_H}(s'_j, s') \leq d_r$, this implies that (S^+, s^+) is a solution. For two strings $s_1, s_2 \in \Sigma^\ell$, let $d_\diamond(s_1, s_2)$ be the number of mismatches between s_1 and s_2 at positions i such that $s'[i] = \diamond$, and $d_\Sigma(s_1, s_2)$ be the number of mismatches at other positions. Clearly $\mathsf{d_H}(s_1, s_2) = d_\diamond(s_1, s_2) + d_\Sigma(s_1, s_2)$. Comparing strings s_j and s'_j to s', we have $d_\diamond(s_j, s') = d_\diamond(s'_j, s')$ (both distances are equal to the number of occurrences of $\diamond$ in s'). Since $\mathsf{d_H}(s_j, s')$ is maximal, we have $d_\Sigma(s'_j, s') \leq d_\Sigma(s_j, s')$. Consider now s^+. Since s^+ is equal to s' in every non-$\diamond$ characters, we have $d_\Sigma(s'_j, s^+) \leq d_\Sigma(s_j, s^+)$. Finally, for any i such that $s'[i] = \diamond$, by hypothesis of this case we have $s_j[i] \neq s^*[i] = s^+[i]$, hence $d_\diamond(s_j, s^+)$ is equal to the number of occurrences of $\diamond$ in s', which is an upper bound for $d_\diamond(s'_j, s^+)$. Overall, $d(s'_j, s^+) \leq d(s_j, s^+)$, and (S^+, s^+) is a solution.

Thus, (S^+, s^+) is a solution such that $S^+ \subseteq S' \setminus \{s_j\}$, s' is a lower bound for s^+, and $\mathsf{d_H}(s', s^+) \leq d$, hence the recursive call in Line 7 is valid. ◀

It follows from the claim above that any valid call to Solve Closest String-wo returns a solution. Indeed, if it does not directly return a solution, then it receives a solution of a more constrained instance from a valid recursive call, which is returned on Line 8 or 14.

Claim 3. Let s' be the majority string for S where for every disputed column i, $s'[i] = \diamond$. Then Solve Closest String-wo$(S, t, s', 2d_r + D)$ is a valid call.

Proof of Claim 3. Consider a solution (S^*, s^*). We need to check whether $\mathsf{d_H}(s^*, s') \leq 2d_r + D$, and whether s' is a lower bound of s^*. The fact that s' is a lower bound follows from the definition, since $\diamond$ is selected in every disputed column, and the majority character is selected in the other columns. String s^* can be seen as a solution of (r, s)-Closest String over S^*, d_r, d_s, thus, we can use Lemma 2: the distance between s^* and the majority string of S^* is at most $2d_r$. Hence there are at most $2d_r$ mismatches between s' and s^* in non-disputed columns (since in those columns, the majority characters are identical in S and S^*). Adding the D mismatches from disputed columns, we get the $2d_r + D$ upper bound. ◀

◀

2.2 The (r)- and (s)-Variants of Closest String-wo

In [5], the fixed-parameter tractability of (r)-Closest String-wo w. r. t. parameter k and w. r. t. parameters $(|\Sigma|, d_r, k-t)$ are reported as open problems. Since Theorem 5 also applies to (r)-Closest String-wo, the only open cases left for the (r)-variant are the following:

▶ **Open Problem 8.** *What is the fixed-parameter tractability of* (r)-Closest String-wo *with respect to* $(|\Sigma|, k-t)$, $(|\Sigma|, d_r)$ *and* $(|\Sigma|, d_r, k-t)$*?*

Next, we consider the (s)-variant of Closest String-wo. We recall that replacing the radius bound by a bound on the distance sum turns (r)-Closest String into a trivial problem, while (s)-Closest Substring remains hard. The next result shows that Closest String-wo behaves like Closest Substring in this regard. For the proof, we use Multi-Coloured Clique (which is W[1]-hard, see [11]), which is identical to the standard parameterisation of Clique, but the input graph $G = (V, E)$ has a partition $V = V_1 \cup \ldots \cup V_{\mathrm{k_c}}$, such that every V_i, $1 \leq i \leq \mathrm{k_c}$, is an independent set (we denote the parameter by $\mathrm{k_c}$ to avoid confusion with the number of input strings k).

▶ **Theorem 9.** (s)-Closest String-wo$(d_s, \ell, k-t)$ *is* W[1]*-hard.*

Proof. Let $G = (V_1 \cup \ldots \cup V_{\mathrm{k_c}}, E)$ be a Multi-Coloured Clique-instance. We assume that, for some $q \in \mathbb{N}$, $V_i = \{v_{i,1}, v_{i,2}, \ldots v_{i,q}\}$, $1 \leq i \leq \mathrm{k_c}$, i. e., each vertex has an index depending on its colour-class and its rank within its colour-class. Let $\Sigma = V \cup \Gamma$, where Γ is some alphabet with $|\Gamma| = |E|(\mathrm{k_c} - 2)$. For every $e = (v_{i,j}, v_{i',j'}) \in E$, let $s_e \in \Sigma^{\mathrm{k_c}}$ with $s_e[i] = v_{i,j}$, $s_e[i'] = v_{i',j'}$ and all other non-defined positions are filled with symbols from Γ such that each $x \in \Gamma$ has exactly one occurrence in the strings s_e, $e \in E$. We set $S = \{s_e \mid e \in E\}$, $t = |E| - \binom{\mathrm{k_c}}{2}$ (i. e., the number of inliers is $\binom{\mathrm{k_c}}{2}$) and $d_s = \binom{\mathrm{k_c}}{2}(\mathrm{k_c} - 2)$.

Let K be a clique of G of size $\mathrm{k_c}$, let $s \in \Sigma^{\mathrm{k_c}}$ be defined by $\{s[i]\} = K \cap V_i$, $1 \leq i \leq \mathrm{k_c}$, and let $S' = \{s_e \mid e \subseteq K\}$. Since $\mathsf{d_H}(s, s') = \mathrm{k_c} - 2$, for every $s' \in S'$, $\mathsf{s_H}(s, S') = d_s$. Consequently, S' and s is a solution for the (s)-Closest String-wo-instance S, t, d_s.

Now let $s \in \Sigma^{\mathrm{k_c}}$ and $S' \subseteq S$ with $|S'| = \binom{\mathrm{k_c}}{2}$ be a solution for the (s)-Closest String-wo-instance S, t, d_s. If, for some $s'_1 \in S'$, $\mathsf{d_H}(s'_1, s) \geq \mathrm{k_c} - 1$, then there is an $s'_2 \in S'$ with $\mathsf{d_H}(s'_2, s) \leq \mathrm{k_c} - 3$. Thus, for some i, $1 \leq i \leq \mathrm{k_c}$, $s[i] = s'_2[i]$ and $s'_2[i] \in \Gamma$, which implies that replacing $s[i]$ by $s'_1[i]$ does not increase $\mathsf{s_H}(s, S')$. Moreover, after this modification, $\mathsf{d_H}(s'_1, s)$ has decreased by 1, while $\mathsf{d_H}(s'_2, s) \leq \mathrm{k_c} - 2$. By repeating such operations, we can transform s such that $\mathsf{d_H}(s', s) \leq \mathrm{k_c} - 2$, $s' \in S'$. Next, assume that, for some i, $1 \leq i \leq \mathrm{k_c}$, there is an $S'' \subseteq S'$ with $|S''| = \mathrm{k_c}$ and, for every $s' \in S''$, $s[i] = s'[i]$. Since $\mathsf{d_H}(s', s) \leq \mathrm{k_c} - 2$ for every

Table 2 Results for (r, s)-Closest String-wo, including (r)- and (s)-variants.

k	t	$\|\Sigma\|$	ℓ	d_r	d_s	$k-t$	Result	Note/Ref.
p	–	–	–	–	–	–	FPT	Thm. 5, Open Prob. in [5]
–	0	2	–	–	–	–	NP-hard	even for d_r-var., but P for d_s-var.
–	**p**	–	**p**	–	–	–	FPT	$d_r \leq \ell$
–	**p**	–	–	**p**	–	–	FPT	Thm. 7, and [5] for d_r-var.
–	**p**	–	–	–	**p**	–	FPT	Thm. 6
–	**p**	–	–	–	–	**p**	FPT	$k = t + (k-t)$
–	–	**p**	**p**	–	–	–	FPT	trivial
–	–	**p**	–	$\star$	$\star$	$\star$	Open	param. $\|\Sigma\|$ and some of $d_r, d_s, k-t$
–	–	–	**p**	**p**	**p**	**p**	W[1]-hard	even for d_r-var. [5] and d_s-var. (Thm. 9)

$s' \in S''$, pigeon-hole principle implies that there are $s'_1, s'_2 \in S''$ with $s'_1[i'] = s'_2[i'] = s[i']$, for some i', $1 \leq i' \leq \mathrm{k_c}$, and $i' \neq i$, which, by the structure of the strings of S, is a contradiction. Consequently, for every i, $1 \leq i \leq \mathrm{k_c}$, s matches with at most $\mathrm{k_c}-1$ strings from S' at position i. Since there are at least $2\binom{\mathrm{k_c}}{2} = \mathrm{k_c}(\mathrm{k_c}-1)$ matches, we conclude that, for every i, $1 \leq i \leq \mathrm{k_c}$, $s[i]$ matches exactly $k_c - 1$ times with the i^{th} position of a string from S'. Hence, $s[i] \in V_i$, $1 \leq i \leq \mathrm{k_c}$, i. e., $s = v_{1,r_1} v_{2,r_2} \dots v_{\mathrm{k_c},r_{\mathrm{k_c}}}$, for some $r_j \in \{1, 2, \dots, q\}$, $1 \leq j \leq \mathrm{k_c}$. Let $K = \{v_{1,r_1}, v_{2,r_2}, \dots, v_{\mathrm{k_c},r_{\mathrm{k_c}}}\}$. For every $s' \in S'$, by definition of the strings s_e, we have $\mathsf{d_H}(s, s') \geq \mathrm{k_c}-2$, combining with the lower-bound proved ealier, we conclude $\mathsf{d_H}(s, s') = \mathrm{k_c}-2$, for every $s' \in S$. Now let $e = (v_{i,j}, v_{i',j'}) \in E$ be such that $s_e \in S'$. From $\mathsf{d_H}(s, s_e) = \mathrm{k_c}-2$ its follows that $s[i] = v_{i,j}$ and $s[i'] = v_{i',j'}$, which implies $e \subseteq K$. Since $|S| = \binom{\mathrm{k_c}}{2}$, there are $\binom{\mathrm{k_c}}{2}$ edges connecting vertices from K; thus, K is a clique. ◀

Setting $d_r = \mathrm{k_c}-2$ instead of $d_s = \binom{\mathrm{k_c}}{2}(\mathrm{k_c}-2)$ in the reduction of Theorem 9 leads to a simpler proof for the W[1]-hardness of (r)-Closest String-wo$(d_r, \ell, k-t)$ shown in [5] (on the other hand, the reduction of [5] does not work for (s)-Closest String-wo$(d_s, \ell, k-t)$). The results obtained in this section are summarized in Table 2.

3 Closest Substring

In this section, we consider the problem (r, s)-Closest Substring and, as done in Section 2 for (r, s)-Closest String, we classify all parameterisations of (r, s)-Closest Substring (and its (r)- and (s)-variants) with respect to the parameters ℓ, k, m, d_r, d_s and $|\Sigma|$ into either fixed-parameter tractable or W[1]-hard. Of course, many of those questions are already solved in the literature, but, unlike for (r, s)-Closest String, not all cases of the (r)- and (s)-variants are settled, i. e., the status of (s)-Closest Substring(ℓ) is unknown, which is mentioned as open problem in [24]. We shall first close this gap by defining a reduction from Multi-Coloured Clique to (s)-Closest Substring.

Let $G = (V_1 \cup \dots \cup V_{\mathrm{k_c}}, E)$ be a Multi-Coloured Clique-instance. We assume that, for some $q \in \mathbb{N}$, $V_i = \{v_{i,1}, v_{i,2}, \dots v_{i,q}\}$, $1 \leq i \leq \mathrm{k_c}$, i. e., each vertex has an index depending on its colour-class and its rank within its colour-class. Let $\Sigma = V \cup \{\$, \diamond\}$. For every j, $1 \leq j \leq q$, we list all j^{th} elements of the colour-classes as a string $\mathcal{V}_j = \$v_{1,j} v_{2,j} \dots v_{\mathrm{k_c},j}$. For every edge $e = (v_{i,j}, v_{i',j'})$ with $i < i'$, we define a string $\mathcal{E}_e = \$\diamond^i v_{i,j} \diamond^{i'-i-1} v_{i',j'} \diamond^{\mathrm{k_c}-i'-1}$. Note that $\mathcal{E}_e = \$\diamond\mathcal{E}'_e$, where $|\mathcal{E}'_e| = \mathrm{k_c}$, the positions i and i' of $\mathcal{E}'_e$ are $v_{i,j}$ and $v_{i',j'}$, respectively, and all remaining positions are $\diamond$. The (s)-Closest Substring-instance is now defined as follows. Let S contain $N = |E|(\mathrm{k_c}+2)+1$ occurrences of each $\mathcal{V}_j$, $1 \leq j \leq q$, and one occurrence of each $\mathcal{E}_e$, $e \in E$, and let $m = \mathrm{k_c}+1$. We note that $\ell = \mathrm{k_c}+2$. See Figure 2 for an example.

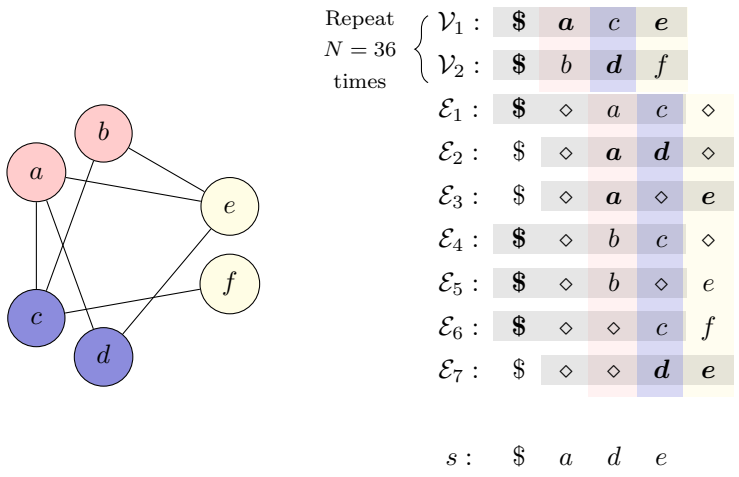


Figure 2 Illustration of the parameterized reduction from a Multi-Coloured Clique-instance to (s)-Closest Substring. The colour-classes of the graph are $V_1 = \{a, b\}$ (red), $V_2 = \{c, d\}$ (blue) and $V_1 = \{e, f\}$ (yellow), the occurrences of symbols from V in the strings $\mathcal{V}_j$ and $\mathcal{E}_i$ are coloured according to their colour-classes. The string $s = \$ade$ is an optimal solution with respect to the substrings emphasised with grey background (positions producing a match are in bold). Note that vertices $\{a, d, e\}$ form a clique in G.

In the following, we extend the notation of *radius optimal* and *distance sum optimal* to sets $S \subseteq \Sigma^{\leq \ell}$ and strings $s \in \Sigma^m$ in the natural way by taking all sets S' of length-m substrings of the string in S into account. The next lemma shows that distance sum optimal strings (with respect to S and m) are basically lists of vertices from each colour-class.

▶ **Lemma 10 (*).** *If $s \in \Sigma^{k+1}$ is distance sum optimal w. r. t. S, then $s \in \{\$\} \cdot V_1 \cdot V_2 \cdot \ldots \cdot V_k$.*

Now let s be distance sum optimal with respect to S and m. From Lemma 10, we can conclude that $s = \$v_{1,r_1} v_{2,r_2} \ldots v_{k_c,r_{k_c}}$, for some $r_j \in \{1, 2, \ldots, q\}$, $1 \leq j \leq k_c$. Let K be the corresponding set of vertices, i. e., $K = \{v_{1,r_1}, v_{2,r_2}, \ldots, v_{k_c,r_{k_c}}\}$.

▶ **Lemma 11 (*).** *Let $e \in E$. The optimal distance between s and a length-$(\mathrm{k_c}+1)$ substring of $\mathcal{E}_e$ is $\mathrm{k_c}-1$ if $e \subseteq K$, and $\mathrm{k_c}$ otherwise.*

Using the lemmas from above, we can now show the correctness of the reduction.

▶ **Theorem 12.** *(s)-Closest Substring(ℓ, m) is W[1]-hard.*

Proof. Let $s \in \Sigma^{\mathrm{k_c}+1}$ be distance sum optimal with respect to S and m, and let K be the corresponding set of vertices. We first note that the total distance from s to the N copies of the strings $\mathcal{V}_j$, $1 \leq j \leq q$, is exactly $Nq\,\mathrm{k_c}$. According to Lemma 11, for every $e \in E$, the optimal distance sum between s and the respective substring of $\mathcal{E}_e$ is $\mathrm{k_c}-1$ if $e \subseteq K$, and $\mathrm{k_c}$ otherwise. Hence, the total distance sum from s to the respective substrings of $\mathcal{E}_e$, $e \in E$, is $|E|\,\mathrm{k_c} - r$, where $r = \{e \in E \mid e \subseteq K\}$, and the total distance sum between s and S is therefore $Nq\,\mathrm{k_c} + |E|\,\mathrm{k_c} - r$. This implies that the distance sum between s and S is $Nq\,\mathrm{k_c} + |E|\,\mathrm{k_c} - \frac{\mathrm{k_c}(\mathrm{k_c}-1)}{2}$ if and only if $r = \frac{\mathrm{k_c}(\mathrm{k_c}-1)}{2}$ if and only if K is a clique of size $\mathrm{k_c}$. Consequently, the above reduction, with the addition of $d_s = Nq\,\mathrm{k_c} + |E|\,\mathrm{k_c} - \frac{\mathrm{k_c}(\mathrm{k_c}-1)}{2}$, is a parameterised reduction from Multi-Coloured Clique to (s)-Closest Substring(ℓ, m). ◀

Table 3 Results for (s)-Closest Substring.

ℓ	k	m	d_s	$\lvert\Sigma\rvert$	Result	Reference
–	–	**p**	–	**p**	FPT	trivial
p	–	–	–	**p**	FPT	[24]
p	**p**	–	–	–	FPT	[24]
p	–	–	**p**	–	FPT	[24]
–	–	–	**p**	**p**	FPT	[20]
–	**p**	–	–	**2**	W[1]-hard	[14]
–	**p**	**p**	**p**	–	W[1]-hard	[14]
p	–	**p**	–	–	W[1]-hard	Thm. 12

Table 4 Results for (r, s)-Closest Substring.

ℓ	k	m	d_r	d_s	$\lvert\Sigma\rvert$	Result	Reference
–	–	**p**	–	–	**p**	FPT	Thm. 14
p	**p**	–	–	–	–	FPT	Thm. 14
p	–	–	–	**p**	–	FPT	Thm. 14
p	–	–	–	–	**p**	FPT	Thm. 14
p	–	**p**	**p**	–	–	W[1]-hard	Cor. 13, Open Prob. in [24]
–	**p**	–	**p**	**p**	**p**	W[1]-hard	[20]
–	**p**	**p**	**p**	**p**	–	W[1]-hard	[14]

As illustrated by Table 3, Theorem 12 together with known results from the literature completely settle the parameterised complexity of (s)-Closest Substring.

Moving on to the problem (r, s)-Closest Substring, we first observe that reducing (s)-Closest Substring to (r, s)-Closest Substring by setting $d_r = m$ is a parameterised reduction from (s)-Closest Substring(ℓ, m) to (r, s)-Closest Substring(ℓ, m, d_r), which implies the following corollary:

▶ **Corollary 13.** *(r, s)-Closest Substring(ℓ, m, d_r) is W[1]-hard.*

Next, we consider several fixed-parameter tractable variants of (r, s)-Closest Substring.

▶ **Theorem 14 (*).** *(r, s)-Closest Substring$(x) \in$ FPT, for every $x \in \{(m, |\Sigma|), (\ell, k), (\ell, |\Sigma|), (\ell, d_s)\}$.*

It remains to observe that all remaining parameterisations of (r, s)-Closest Substring are W[1]-hard. More precisely, it is known that (r)-Closest Substring is W[1]-hard for parameterisations $(k, d_r, |\Sigma|)$ (see [20]) and (k, m, d_r) (see [14]). Hence, the obvious reduction from (r)-Closest Substring to (r, s)-Closest Substring, i. e., setting $d_s = k\, d_r$, shows that (r, s)-Closest Substring is W[1]-hard for parameterisations $(k, d_r, d_s, |\Sigma|)$ and (k, m, d_r, d_s). As can be checked with the help of Table 4, this now classifies all parameterised variants of (r, s)-Closest Substring.

4 Kernelisation

Neither (r)-Closest String$(d_r, \ell, |\Sigma|)$ nor (r)-Closest Substring(k, m, d_r) admit polynomial kernels unless coNP $\subseteq$ NP/Poly (see [2]), and (r)-Closest String(k, d_r) has a kernel of size $\mathrm{O}(k^2 d_r \log k)$ (see [17]). From these results, we can conclude the following:

▶ **Proposition 15** (*).

- (r, s)-Closest String$(d_r, \ell, |\Sigma|)$ *has no polynomial kernel unless* coNP $\subseteq$ NP/Poly.
- (r, s)-Closest String(k, d_r) *has a kernel of size* $\mathrm{O}(k^2 d_r \log k)$.
- (r, s)-Closest String(d_s) *has a kernel of size* $\mathrm{O}((d_s)^3 \log d_s)$.

This only leaves the case open, where only k (or k and $|\Sigma|$, which, due to the dependency $|\Sigma| \leq k$ (see [17]), is the same question) is a parameter (regarding this case, note that for (r)-Closest String(k) no combinatorial kernel or combinatorial FPT-algorithm is known).

▶ **Proposition 16** (*).

- (r, s)-Closest Substring$(k, m, d_r, d_s, |\Sigma|)$ *has no polynomial kernel unless* coNP $\subseteq$ NP/Poly.
- (r, s)-Closest Substring(ℓ, k) *and* (r, s)-Closest Substring(ℓ, d_s) *have kernels of size* $\mathrm{O}(\ell k)$ *and* $\mathrm{O}(\ell d_s)$*, respectively.*

This almost settles the (r, s)-variant, for which only the parameterisation $(\ell, |\Sigma|)$ is open. For the (r)-variant, the parameterisations ℓ, (ℓ, d_r) and $(\ell, |\Sigma|)$, and for the (s)-variant, the parameterisations $(m, |\Sigma|)$ and $(d_s, |\Sigma|)$ are open.

For (r)-Closest String-wo no kernelisation lower bounds are known so far. However, the following can be concluded from [2]:

▶ **Proposition 17** (*). (r)-Closest String-wo$(d_r, \ell, t, |\Sigma|)$ *has no polynomial kernel unless* coNP $\subseteq$ NP/Poly.

By a cross-composition[3] from (r)-Closest String into (r)-Closest String-wo, we can rule out a polynomial kernel for the parameterisation $(d_r, d_s, \ell, (k-t), |\Sigma|)$.

To this end, we define a polynomial equivalence relation $\sim$ over the (r)-Closest String-instances as follows. For $j \in \{1, 2\}$, let $S_j = \{s_{j,i} \mid 1 \leq i \leq k_j\} \subseteq \Sigma^{\ell_j}$ and $d_{r,j} \in \mathbb{N}$. Then $(S_1, d_{r,1}) \sim (S_2, d_{r,2})$ if $k_1 = k_2$, $\ell_1 = \ell_2$ and $d_{r,1} = d_{r,2}$. Now let $(S_1, d_r), (S_2, d_r), \ldots, (S_q, d_r)$ be $\sim$-equivalent (r)-Closest String-instances, where, for the sake of convenience, $S_i = \{s_{i,1}, s_{i,2}, \ldots, s_{i,k}\} \subseteq \Sigma^{\ell}$, $1 \leq i \leq q$. For every i, $1 \leq i \leq q$, let B_i denote the binary representation of i with exactly $\lceil \log(q) \rceil$ bits, and let $C_i = (B_i)^{2d_r+1}$. Moreover, for every i, $1 \leq i \leq q$, let $S'_i = \{s'_{i,1}, s'_{i,2}, \ldots, s'_{i,k}\}$, where, for every j, $1 \leq j \leq k$, $s'_{i,j} = s_{i,j} C_i$. Finally, let the (r, s)-Closest String-wo-instance be (S', d'_r, d'_s, t) with $S' = \bigcup_{i=1}^{q} S'_i$, $d'_r = d_r$, $d'_s = k d_r$ and $t = (q-1)k$.

▶ **Theorem 18** (*). (r, s)-Closest String-wo$(d_r, d_s, \ell, (k-t), |\Sigma|)$ *does not admit a polynomial kernel unless* coNP $\subseteq$ NP/Poly.

5 Conclusions

The parameterised complexity of the (r)-, (s)- and general variant of Closest String and Closest Substring with respect to $\ell, k, m, d_r, d_s, |\Sigma|$ is now completely settled. For (r, s)-Closest Substring, where positive results are less abundant, it might be worthwhile to identify other parameters that yield fixed-parameter tractability. For (r, s)-Closest String, it should be pointed out that the FPT-algorithms with respect to k are based on ILP and are most likely practically not relevant; direct combinatorial FPT-algorithms are still unknown. For the outlier variant of (r, s)-Closest String, many cases are left open, most prominently, the ones with $|\Sigma|$ as parameter, and we expect those to be challenging. Moreover, for several FPT-variants, the existence of polynomial kernels is not yet answered.

[3] For the technique of cross-composition, see Bodlaender et al. [4].

References

1. A. Amir, G. M. Landau, J. C. Na, H. Park, K. Park, and J. S. Sim. Efficient algorithms for consensus string problems minimizing both distance sum and radius. *Theoretical Computer Science*, 412:5239–5246, 2011.
2. M. Basavaraju, F. Panolan, A. Rai, M. S. Ramanujan, and S. Saurabh. On the kernelization complexity of string problems. In *Proc. 20th International Conference on Computing and Combinatorics, COCOON 2014*, volume 8591 of *LNCS*, pages 141–153, 2014.
3. A. Ben-Dor, G. Lancia, R. Ravi, and J. Perone. Banishing bias from consensus sequences. In *Proc. 8th Annual Symposium on Combinatorial Pattern Matching, CPM 1997*, volume 1264 of *LNCS*, pages 247–261, 1997.
4. H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal of Discrete Mathematics*, 28(1):277–305, 2014.
5. C. Boucher and B. Ma. Closest string with outliers. *BMC Bioinformatics*, 12:S55, 2011.
6. L. Bulteau, F. Hüffner, C. Komusiewicz, and R. Niedermeier. Multivariate algorithmics for NP-hard string problems. *Bulletin of the EATCS*, 114:31–73, 2014.
7. M. Cygan, F. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.
8. X. Deng, G. Li, Z. Li, B. Ma, and L. Wang. Genetic design of drugs without side-effects. *SIAM Journal of Computing*, 32(4):1073–1090, 2003.
9. J. Dopazo, A. Rodríguez, J. Sáiz, and F. Sobrino. Design of primers for PCR amplification of highly variable genomes. *Computer Applications in the Biosciences*, 9(2):123–125, 1993.
10. R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
11. Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.
12. P. A. Evans, A. Smith, and H. T. Wareham. The parameterized complexity of p-center approximate substring problems. Technical Report TR01-149, Faculty of Computer Science, University of New Brunswick, Canada, 2001.
13. P. A. Evans, A. D. Smith, and H. T. Wareham. On the complexity of finding common approximate substrings. *Theoretical Computer Science*, 306:407–430, 2003.
14. M. R. Fellows, J. Gramm, and R. Niedermeier. On the parameterized intractability of motif search problems. *Combinatorica*, 26:141–167, 2006.
15. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.
16. M. Frances and A. Litman. On covering problems of codes. *Theory of Computing Systems*, 30:113–119, 1997.
17. J. Gramm, R. Niedermeier, and P. Rossmanith. Fixed-parameter algorithms for closest string and related problems. *Algorithmica*, 37:25–42, 2003.
18. J. K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. *Information and Computation*, 185:41–55, 2003.
19. K. Lucas, M. Busch, S. Mössinger, and J. A. Thompson. An improved microcomputer program for finding gene- or gene family-specific oligonucleotides suitable as primers for polymerase chain reactions or as probes. *Computer Applications in the Biosciences*, 7(4):525–529, 1991.
20. D. Marx. Closest substring problems with small distances. *SIAM Journal on Computing*, 38:1382–1410, 2008.
21. G. Pavesi, G. Mauri, and G. Pesole. An algorithm for finding signals of unknown length in DNA sequences. *Bioinformatics*, 17:S207–S214, 2001.
22. P. Pevzner and S. Sze. Combinatorial approaches to finding subtle signals in DNA strings. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology, ISMB 2000*, pages 269–278, 2000.

23 V. Proutski and E. C. Holmes. Primer master: a new program for the design and analysis of PCR primers. *Computer Applications in the Biosciences*, 12(3):253–255, 1996.

24 Markus L. Schmid. Finding consensus strings with small length difference between input and solution strings. *ACM Transactions on Computation Theory*, 9(3):13:1–13:18, 2017.

25 M. Tompa, N. Li, T. L. Bailey, G. M. Church, B. De Moor, E. Eskin, A. V. Favorov, M. C. Frith, Y. Fu, W. J. Kent, V. J. Makeev, A. A. Mironov, W. S. Noble, G. Pavesi, G. Pesole, M. Régnier, N. Simonis, S. Sinha, G. Thijs, J. van Helden, M. Vandenbogaert, Z. Weng, C. Workman, C. Ye, and Z. Zhu. Assessing computational tools for the discovery of transcription factor binding sites. *Nature Biotechnology*, 23(1):137–144, 2005.

Plain Stopping Time and Conditional Complexities Revisited

Mikhail Andreev
IPONWEB, Berlin, Germany
amishaa@mail.ru

Gleb Posobin[1]
National Research University – Higher School of Economics, Moscow, Russia
posobin@gmail.com

Alexander Shen[2]
LIRMM CNRS / University of Montpellier, France. On leave from IITP RAS, Moscow, Russia
alexander.shen@lirmm.fr
https://orcid.org/0000-0001-8605-7734

Abstract

In this paper we analyze the notion of "stopping time complexity", the amount of information needed to specify when to stop while reading an infinite sequence. This notion was introduced by Vovk and Pavlovic [9]. It turns out that plain stopping time complexity of a binary string x could be equivalently defined as (a) the minimal plain complexity of a Turing machine that stops after reading x on a one-directional input tape; (b) the minimal plain complexity of an algorithm that enumerates a prefix-free set containing x; (c) the conditional complexity $\mathrm{C}(x\,|\,x*)$ where x in the condition is understood as a prefix of an infinite binary sequence while the first x is understood as a terminated binary string; (d) as a minimal upper semicomputable function K such that each binary sequence has at most 2^n prefixes z such that $K(z) < n$; (e) as $\max \mathrm{C}^X(x)$ where $\mathrm{C}^X(z)$ is plain Kolmogorov complexity of z relative to oracle X and the maximum is taken over all extensions X of x.

We also show that some of these equivalent definitions become non-equivalent in the more general setting where the condition y and the object x may differ, and answer an open question from Chernov, Hutter and Schmidhuber [3].

2012 ACM Subject Classification Mathematics of computing → Information theory

Keywords and phrases Kolmogorov complexity, stopping time complexity, structured conditional complexity, algorithmic information theory

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.2

Related Version A full version of the paper is available at [1], `https://arxiv.org/abs/1708.08100`.

Funding Supported by RaCAF ANR-15-CE40-0016-01 RaCAF grant

Acknowledgements The authors are grateful to Alexey Chernov, Volodya Vovk, members of the ESCAPE team (LIRMM, Montpellier), Kolmogorov seminar (Moscow) and Theoretical Computer Science Laboratory (National Research University Higher School of Economics, Computer Science department, Moscow), the participants of Dagstuhl meeting where some results of the paper were presented [2], and the reviewers of preliminary versions of this paper.

[1] Supported by Russian Academic Excellence Project 5–100
[2] Supported by RaCAF ANR-15-CE40-0016-01 RaCAF grant

1 Introduction: stopping time complexity

Imagine that you explain to someone which exit on a long road she should take. You can just say "Nth exit"; for that you need $\log N$ bits. You may also say something like "the first exit after the first bridge", and this message has bounded length even if the bridge is very far away.[3]

More formally, consider a machine with one-directional read-only input tape that contains bits $x_0, x_1, \ldots x_n, \ldots$. We want to program the machine in such a way that it stops after reading bits $x_0, \ldots, x_{n-1}$ (and never sees x_n and the subsequent bits). Obviously, the complexity of this task does not depend on the values of $x_n, x_{n+1}, \ldots$, because the machine never sees them, so this complexity should be a function of a bit string $x = x_0 x_1 \ldots x_{n-1}$. It can be called the "stopping time complexity" of x.

Such a notion was introduced recently by Vovk and Pavlovic [9]. In their paper an "interactive" version of stopping time complexity is considered where even (x_{2n}) and odd (x_{2n+1}) terms are considered differently, but this is just a special case, so we do not consider this setting. It turns out that the stopping time complexity is a special case of conditional Kolmogorov complexity with structured conditions. (In this paper we consider the plain version of stopping time complexity and postpone similar questions for prefix versions.)

The Kolmogorov complexity was introduced independently by Solomonoff, Kolmogorov, and Chaitin to measure the "amount of information" in a finite object (say, in a binary string). One can also consider the *conditional* version of complexity where some other object (a *condition*) is given "for free". Later different versions of Kolmogorov complexity appeared (plain, prefix, a priori, monotone complexities). We assume that the reader is familiar with basic notions of algorithmic information theory, see, for example, [6] for a short introduction and [7] for a detailed exposition.

For the plain version of stopping time complexity we prove the equivalence between five different definitions (Section 2). First, we show that it can be equivalently defined as (1) the minimal plain complexity of a machine with one-way read-only input tape that stops after reading x, or (2) the minimal enumeration complexity of a prefix-free set that contains x. Then we show how the stopping time complexity can be expressed in terms of plain conditional complexity that is monotone with respect to conditions. Namely, we prove that (3) the stopping time complexity equals $\mathrm{C}(x \mid x*)$ where x is used both as an object and a condition. Of course, according to standard definitions, the complexity $\mathrm{C}(x \mid x)$ is $O(1)$, but now we treat these two strings x differently (and use a star in the notation to stress this). One may say that the topologies in the space of objects and the space of conditions are different. The first x (object to be described) is considered as an isolated object (terminated string). The second x (in the condition) is considered as a prefix of an infinite sequence. In [7, Section 6.3] this approach is described in general (see also [5] for even more general setting); to make this paper self-contained, we give all necessary definitions for our special case. We call this version of complexity "monotone-conditional complexity" since this function is monotone with respect to the condition. Then we provide a characterization of stopping time complexity in quantitative terms proving that (4) stopping time complexity is the minimal upper semicomputable function satisfying some restrictions (no more than 2^n prefixes of any given sequence could have complexity at most n). Finally, we point out the connections with the relativized version $\mathrm{C}^A(x)$ of plain complexity and prove that (5) the

[3] We do not allow, however, the description "the last exit before the bridge", since it uses information that is unavailable at the moment when we have to take the exit.

stopping time complexity of a binary string x is the maximal value of $C^A(x)$ for all oracles (infinite bit sequences) A that have prefix x.

Having such a robust definition for plain stopping time complexity, one may ask whether similar characterizations can be obtained for a more general notion of $C(y|x*)$ where x and y are arbitrary strings. Unfortunately, here the situation is much worse, and we prove mostly negative results (Section 3). We show that while $C(y|x*)$ can be defined as a minimal plain complexity of a prefix-stable program that maps x to y (Theorem 12), it cannot be defined as a minimal plain complexity of a prefix-free program that maps some prefix of x to y (Theorem 13; this result answers a question posed in [3]). Then we show that the attempt to define $C(y|x*)$ by quantitative restrictions also fails: we get another function that may be up to two times less (Theorem 14).

2 Equivalent definitions

2.1 Machines and prefix-free sets

Consider a Turing machine M that has one-directional read-only input tape with binary alphabet, and a work tape with arbitrary alphabet (or many work tapes). Let x be a binary string. We say that *M stops at x* if M, being started with the input tape x (and an empty work tape, as usual), reads all the bits of x and stops without trying to read more bits. (We assume that initially the input head is on the left of x, so it needs to move right before seeing the first bit of x.) For a given x, we may consider the *minimal plain Kolmogorov complexity of a machine M that stops at x*. This quantity is independent (up to $O(1)$-additive term) of the details of the definition (work tape alphabet, number of work tapes, etc.) since computable conversion algorithms exist, and a computable transformation may increase complexity only by $O(1)$. We arrive at the following definition:

▶ **Definition 1.** The *plain stopping time complexity* of x is the minimal plain Kolmogorov complexity of a Turing machine that stops at x.

Here is a machine-independent equivalent characterization of the plain stopping time complexity.

▶ **Theorem 2.** *Plain stopping time complexity of x equals (up to an $O(1)$ additive term) the minimal complexity of a program that enumerates some prefix-free set containing x.*

(A set of strings is called *prefix-free* if it does not contain a string and its proper prefix at the same time.)

Proof. One direction is simple: For a Turing machine M of the type described the set $\{x\colon M \text{ stops at } x\}$ is enumerable (we may simulate all runs) and prefix-free (if M stops at some x, then for every extension y of x the machine M will behave in the same way on tapes x and y, so M with input y stops after reading x and never reads the rest of y). This computable conversion (of a machine into an enumeration program) increases complexity at most by $O(1)$.

The other direction is a bit more complicated. Imagine that we have a program that enumerates some prefix-free set U of strings. How can we construct a machine that stops exactly at the strings in U? Initially no bits of x are read. Enumerating U, we wait until some element u of U appears. (If this never happens, the machine never stops, and this is OK.) If u is empty, machine stops. In this case U cannot contain non-empty strings (being prefix-free), so the machine's behavior is correct. If u is not empty, we know that empty

string is not in U (since U is prefix-free), so we may read the first bit of x without any risk, and get some one-bit string v. Then we wait until v or some extension of v appears in U (it may have already happened if u is an extension of v). If v itself appears, the computation stops; if a proper extension of v appears, then v is not in U and we can safely read the next bit, etc. It is easy to check that indeed this machine stops at some x if and only if x belongs to U. ◀

2.2 Monotone-conditional complexity

In this section we show how the stopping time complexity can be obtained as a special case of some general scheme [5, 8, 7]. This scheme can be used to define different versions of Kolmogorov complexity. We consider *decompressors*, called also *description modes*. In our case decompressor is a subset D of the set

$$(\text{descriptions}) \times (\text{conditions}) \times (\text{objects}).$$

Here descriptions, conditions, and objects are binary strings. If $(p, x, y) \in D$, we say that p *is a description of y given x as condition*, and define the *conditional complexity of y given x* (with respect to the description mode D) as the length of the shortest description. The different versions of complexity correspond to different topologies on the spaces involved, and imply different restrictions on description modes. This is explained in [8] or [7, Chapter 6], and we do not go into technical details here. Let us mention only that descriptions and objects can be considered as isolated entities (terminated strings, natural numbers) or prefixes of an infinite sequence (extension of a string provides more information than the string itself). In this way we get four classical versions of complexity:

	isolated descriptions	descriptions as prefixes
isolated objects	plain complexity	prefix complexity
objects as prefixes	decision complexity	monotone complexity

As noted in [7], one can also consider different structures on the condition space, thus getting eight versions of complexity instead of four in the table. In this paper we use only one of them: objects and descriptions are isolated objects, and conditions are considered as prefixes. (Vovk and Pavlovic [9] consider also another version of stopping time complexity that corresponds to the other topology on the description space, but we do not consider this version now.)

To make this paper self-contained, let us give the definitions tailored to the special case we consider (the plain version of monotone-conditional complexity). In this case the set D is called a *description mode* if it satisfies the following requirements:

- D is (computably) enumerable;
- for every p and x there exists at most one y such that $(p, x, y) \in D$;
- if $(p, x, y) \in D$ and x is a prefix of some x', then $(p, x', y) \in D$.

The last requirement reflects the idea that x is considered as a known prefix of a yet unknown infinite sequence; if x' extends x, then x' contains more information than x and can be used instead of x. To stress this kind of monotonicity, we use $*$ in the notation suggested by the following definition.

▶ **Definition 3.** For a given description mode D, we define the function

$$\mathrm{C}_D(y \,|\, x*) = \min\{|p| \colon (p, x, y) \in D\}$$

and call it *monotone-conditional complexity of y with condition x with respect to description mode D*.

By definition, if x is a prefix of some x', the same description can be used, so $\mathrm{C}_D(y\,|\,x'*) \le \mathrm{C}_D(y\,|\,x*)$. Therefore, this function is indeed monotone with respect to the condition in a natural sense.

One could also use a name *plain monotone-conditional complexity* to distinguish this notion from prefix monotone-conditional complexity that can be defined in a similar way by adding the monotonicity restriction along the p-coordinate.

▶ **Proposition 4** (Solomonoff–Kolmogorov's optimality theorem)**.** *There exists a description mode D that makes C_D minimal up to $O(1)$ additive term in the class of all functions $\mathrm{C}_{D'}$ for all description modes D'.*

Proof. As usual, we first note that description modes can be effectively enumerated. This enumeration is obtained as follows. We generate all enumerable sets of triples and then modify them in such a way that the modified set becomes a description mode and is left unchanged if it already was a description mode. Namely, when a triple (p, x, y) appears in the enumeration, we add this triple and all triples (p, x', y) for all extensions x' of x, unless the second condition would be violated after that; in the latter case we ignore (p, x, y).

Let U_n be the nth set in this enumeration. The optimal set U can be constructed as

$$U = \{(0^n 1 p, x, y) \colon (p, x, y) \in U_n\};$$

the standard argument shows that $\mathrm{C}_U \le \mathrm{C}_{U_n} + n + 1$ as required. ◀

▶ **Definition 5.** Fix some optimal description mode D provided by Proposition 4. The function $\mathrm{C}_D(y\,|\,x*)$ is denoted by $\mathrm{C}(y\,|\,x*)$ and called the (plain) *monotone-conditional complexity of y given x*, or the (plain) *conditional complexity of y given x as a prefix.*

If we omit the third requirement for description modes, we get the standard conditional complexity $\mathrm{C}(y\,|\,x)$ in the same way. The notation we use (placing $*$ after the condition) follows [3] though a different version of monotone-conditional complexity is considered there. In general, $\mathrm{C}(y\,|\,x*)$ is greater than the standard conditional complexity $\mathrm{C}(y\,|\,x)$ since we have more requirements for the description modes. One may say also that the condition now is weaker than in $\mathrm{C}(y\,|\,x)$ since we do not know where x terminates. It is easy to show that the difference is bounded by $O(\log |x|)$, since we need at most $O(\log |x|)$ bits to specify how many bits should be read in the condition x. Difference of this order is possible: for example, $\mathrm{C}(n\,|\,0^n) = O(1)$, but $\mathrm{C}(n\,|\,0^n*) = \mathrm{C}(n) + O(1)$ (the condition 0^n is a prefix of a computable sequence $000\ldots$, so it does not help at all).

The following simple result shows that the plain stopping time complexity (Definition 1) is a special case of this definition when $x = y$ (so we do not need a separate notation for the stopping time complexity).

▶ **Theorem 6.** *The complexity $\mathrm{C}(x\,|\,x*)$ is equal (up to $O(1)$ additive term) to the plain stopping time complexity of x.*

Proof. Let D be a description mode. Then for every p we may consider the set S_p of strings x such that $(p, x, x) \in D$. This set is prefix-free: if (p, x, x) and (p, x', x') belong to D and x is a prefix of x', then $(p, x', x) \in D$ according to the third condition, and then $x = x'$ according to the second condition. The algorithm enumerating S_p can be constructed effectively if p is known, so its complexity is bounded by the length of p (plus $O(1)$, as usual). Choosing the shortest p such that $(p, x, x) \in D$, we conclude that the minimal complexity of an algorithm enumerating a prefix-free set containing x does not exceed $\mathrm{C}(x\,|\,x*) + O(1)$.

Going in the other direction, consider an optimal decompressor $U(\cdot)$ that defines the (plain Kolmogorov) complexity of programs enumerating sets of strings. A standard trimming argument shows that we may modify U in such a way that all algorithms $U(p)$ enumerate only prefix-free sets of strings (not changing the sets there were already prefix-free). Then consider a set D of triples

$$(p, x, y) \in D \Leftrightarrow y \text{ is a prefix of } x \text{ and } y \text{ is enumerated by } U(p).$$

This set in obviously enumerable; the second requirement is satisfied since $D(p)$ enumerates a prefix-free set; the third requirement is true by construction, so D is a description mode. If p is the shortest description of a program that enumerates a set containing x, then $(p, x, x) \in D$, so $\mathrm{C}_D(x \,|\, x*) \leq |p|$. Switching to the optimal description mode, we get similar inequality with $O(1)$ additive term, as required. ◀

Another simple observation shows that indeed this complexity may be called the *stopping time* complexity.

▶ **Proposition 7.** *If x has length n, then* $\mathrm{C}(x \,|\, x*) = \mathrm{C}(n \,|\, x*) + O(1)$.

Proof. If D is the optimal description mode used to define $\mathrm{C}(y \,|\, x*)$, we may consider a new set $D' = \{(p, u, |x|) \colon (p, u, x) \in D\}$ that also is a description mode, and then note that $\mathrm{C}_{D'}(n \,|\, x*) \leq \mathrm{C}_D(x \,|\, x*)$. For the other direction, we consider $D' = \{(p, u, z) \colon \exists n\, [(p, u, n) \in D, |u| \geq n, \text{ and } z = (n\text{-bit prefix of } u)]\}$. ◀

▶ Remark. If $a_0 a_1 a_2 \ldots$ is a computable sequence, then

$$\mathrm{C}(a_0 \ldots a_{n-1} \,|\, a_0 \ldots a_{n-1}*) = \mathrm{C}(n \,|\, a_0 \ldots a_{n-1}*) = \mathrm{C}(n)$$

with $O(1)$-precision (the constant depends on the computable sequence, but not on n), so the stopping time complexity can be considered as a generalization of the plain complexity (of a natural number n).

2.3 Quantitative characterization

There is a well known characterization (see, e.g., [8, Section 1.1, Theorem 8]) for plain complexity in terms of upper semicomputable functions that satisfy some properties. Recall that a function is called *upper semicomputable* if it is a pointwise limit of a decreasing sequence of uniformly computable total functions. (Now we need this notion for integer-valued functions; in this case we may assume without loss of generality that the total computable functions used in the definition are also integer-valued; in the general case one needs to consider rational-valued functions.) An equivalent definition of a semicomputable natural-valued function $S(x)$ requires the set $\{(n, x) \colon S(x) < n\}$ to be enumerable.

Plain complexity function $\mathrm{C}(x)$ is upper semicomputable; we know also that

$$\#\{x \colon \mathrm{C}(x) < n\} < 2^n \qquad (*)$$

since there are less than 2^n programs of length less than n. The characterization that we mentioned says that there exist a minimal (up to $O(1)$ additive term) upper semicomputable function that satisfies the requirement $(*)$, and it coincides with the plain complexity function with $O(1)$-precision.

It turns out that this characterization can be generalized to plain stopping time complexity (though the proof becomes more involved). Consider upper semicomputable functions $S(x)$

on strings that have the following property: *for each infinite binary sequence α and for each n there exists less than 2^n prefixes x of α such that $S(x) < n$.* The following statement is true (it appears as Theorem 18 in the extended version of Vovk–Pavlovic's paper [9]).

▶ **Theorem 8.** *There exist a minimal (up to $O(1)$ additive term) function in this class; it coincides with the plain stopping time complexity $C(x\,|\,x*)$ with $O(1)$-precision.*

Proof. The easy part is to show that $C(x\,|\,x*)$ belongs to the class. It is upper semicomputable, since in general the function $C(x\,|\,y*)$ is upper semicomputable (enumerating the set D of triples, we get better and better upper bounds, finally reaching the limit value).

Let α be some infinite sequence. There are less than 2^n algorithms of complexity less than n enumerating prefix-free sets, and each of this prefix-free sets may contain at most one prefix of α. So the second condition is also true.

In the other direction we use some online (interactive) version of Dilworth's theorem (saying that a partially ordered finite set where maximal chain is of length at most k can be partitioned into k antichains) where the set is growing and splitting into antichains should be performed at each stage (and cannot be changed later). The exact statement is as follows.

Consider a game with two players. Alice and Bob alternate. Alice may at each move (irreversibly) mark a vertex of the full binary tree. The restriction is that each infinite branch should contain at most k marked vertices. Bob replies by assigning a color from $1, \ldots, k$ to the newly marked vertex. No vertices of the same color should be comparable (be on the same branch). The colors cannot be changed after they are assigned. Bob loses if he is unable to assign color at some stage (not violating the rules).

▶ **Lemma 9.** *Bob has a computable strategy that prevents him from losing.*

Proof of Lemma 9. This lemma can be proven in different ways. In the extended version of Vovk–Pavlovic's paper [9] the following simple strategy is suggested: Bob assigns the first available color. In other terms, for a new vertex x Bob chooses the first color that is not used for any vertex comparable with x. One needs to check that k colors are always enough. It is not immediately obvious, since more than k vertices could be comparable with x (being its descendants, for example). However, we may note that during the process:

- Colors of comparable vertices are different. (By construction.)
- If a vertex x gets color i, then each smaller color is used either for a predecessor of x or for a descendant of x. (By construction.)
- If x is a vertex (colored or not), T_x is the set of colors used in the subtree rooted at x (including x itself), and P_x is the set of colors used on the path to x (not including x), then T_x and P_x are disjoint and T_x is the initial segment in the complement to P_x. (Indeed, the disjointness is mentioned above. If y appears in T_x, then all smaller colors appear either below y (therefore in P_x or in T_x), or above y (therefore in T_x).
- The sets T_{x0} and T_{x1} for two brother vertices $x0$ and $x1$ are comparable with respect to inclusion. (Indeed, they are two initial segments of the same ordered set, the complement to P_{x0} or P_{x1}; note that $P_{x0} = P_{x1}$.)
- For each x the total number of colors used in T_x is minimal, i.e., this number equals the maximal number of marked vertices on some path in T_x. (Induction using the previous property.)

The last property implies that Bob never uses more than k colors, since by assumption the total number of marked vertices on a path is at most k.

A different description of the same strategy that explains why it is successful is provided in the extended version of this paper [1]. ◀

Now let us show how the lemma is used to finish the proof of Theorem 8. Let S be a function in the class; since S is upper semicomputable, Alice may, given n, enumerate strings x such that $S(x) < n$; we know that there is at most 2^n strings of this type along any branch of the tree, so Alice never violates the restriction for $k = 2^n$. The lemma then says that Bob can assign 2^n colors (represented as n-bit strings) to all the vertices in such a way that compatible vertices (a string and its prefix) never get the same color. We run these games for all n in parallel; if vertex x gets color c, we put x into an enumerable set indexed by c. The rules of the game guarantee that all these sets are prefix-free, and the algorithm enumerating cth set needs only $|c|$ bits of information. So, if $S(x) < n$, there exists an algorithm of complexity $n + O(1)$ that enumerates a prefix-free set containing x. This means that $\mathrm{C}(x\,|\,x*) \leq S(x) + O(1)$ as required. ◀

2.4 Oracles and the stopping time complexity

It is natural to compare the stopping time complexity $\mathrm{C}(x\,|\,x*)$ and the relativized complexity $\mathrm{C}^X(x)$ where X is some oracle (infinite binary sequence) that has x as a prefix. (The relativized complexity $\mathrm{C}^X(x)$ can be naturally defined as a function of two arguments, a binary string x and an infinite binary sequence X, up to $O(1)$ additive term.)

It is easy to see that

$$\mathrm{C}^X(x) \leq \mathrm{C}(x\,|\,x*)$$

for every X that has prefix x: an oracle access to entire sequence X is more powerful than a bit-by-bit sequential access to x without the right to read too much (beyond x). More formally, let D be a set of triples (p, x, y) used to define $\mathrm{C}(y\,|\,x*)$ (Definition 5). Then we say that p is a description of x with oracle X (as the definition of $\mathrm{C}^X(x)$ requires) if $(p, z, x) \in D$ for some z that is a prefix of X. For a given X every string p can be a description of only one x, since D is monotone. If $(p, x, x) \in D$ and X is an extension of x, then p is a description of x, and we get the required inequality.

The "last exit before the bridge" example shows that $\mathrm{C}^X(x)$ can be much smaller than $\mathrm{C}(x\,|\,x*)$ for *some* extensions X of x: we have $\mathrm{C}(0^n\,|\,0^n*) = \mathrm{C}(n) + O(1)$, but $\mathrm{C}^X(0^n) = O(1)$ for $X = 0^n 10^\infty$.

So it is natural to take *maximum* over all oracles X that extend a given string x. Indeed this approach works:

▶ **Theorem 10.**

$$\mathrm{C}(x\,|\,x*) = \max\{\mathrm{C}^X(x) \colon X \textit{ is an infinite extension of } x\} + O(1).$$

Proof. As we have already mentioned, $\mathrm{C}^X(x) \leq \mathrm{C}(x\,|\,x*) + O(1)$ for every infinite extension X of x. This shows that right hand side does not exceed the left hand side.

Before proving the reverse inequality, let us discuss informally its meaning (this discussion is not used in the argument below and can be omitted). The reverse inequality is a minimax-type result that shows that either (1) there exists a short program that produces x given *any* extension of x as an oracle (and never reads bits after x, but this is not so important for us now), or (2) there exists a "hard to use" extension X of x such that *any* program that computes x given X is long. In other words, it is not possible that for every extension X of x there exists a short program that computes x given X as an oracle, but these programs depend on X and only a much longer program works for all extensions.

For the proof of the reverse inequality we use the quantitative characterization of stopping time complexity (Theorem 8). Let $S(x)$ be the value of the right hand side. It is enough to

prove that S is upper semicomputable and that $S(x) < n$ cannot happen for 2^n different prefixes x of some infinite branch X.

The second claim follows directly from the definition. Let $x_1, \ldots, x_k$ be some prefixes of an infinite sequence X such that $S(x_i) < n$ for all $i = 1, \ldots, k$. We need to show that $k < 2^n$. Since $S(x_i)$ is defined as maximum and X is an extension of x_i, we know that $\mathrm{C}^X(x_i) < n$ for all i and the same X. It remains to note that the number of different programs of length less than n is smaller than 2^n (and the same programs with the same oracles give the same results).

To show that $S(x)$ is upper semicomputable, we use the standard compactness argument. As usual, it is enough to show that the binary relation $S(x) < n$ is (computably) enumerable. Indeed, for every x, the set $\{X\colon \mathrm{C}^X(x) < n\}$ is the union, taken over all strings p of length less than n, of the sets

$$\{X\colon p \text{ is a description of } x \text{ with oracle } X\}.$$

Each of these sets is an open set in the Cantor space, since every terminating oracle computation uses only a finite part of the oracle, and the intervals in the Cantor space that form these sets, can be effectively enumerated for all p and x. The inequality $S(x) < n$ means that the union of these intervals for all p of length less than n covers the Cantor space. Now compactness guarantees that this happens already at some finite stage of the enumeration, so the property $S(x) < n$ is indeed enumerable. ◀

3 Non-equivalence results

3.1 Prefix-stable or prefix-free functions?

Looking at the characterization of $\mathrm{C}(x\,|\,x*)$ as the minimal enumeration complexity of a prefix-free set containing x (Theorem 6), one can ask whether a similar characterization works for the general case, i.e., whether $\mathrm{C}(y\,|\,x*)$ can be characterized as a minimal complexity of programs (machines) with some property. The answer is 'yes', but we should be careful choosing a property of programs used in this characterization. Here are the details.

▶ **Definition 11.** A partial function f defined on binary strings is called

- *prefix-free* if its domain is prefix-free (function is never defined on a string and its extension at the same time);
- *prefix-stable* if for every x, if $f(x)$ is defined, then f is defined and has the same value on all (finite) extensions of x.

It is easy to see that the definition of $\mathrm{C}(y\,|\,x*)$ can be reformulated in terms of prefix-stable functions:

▶ **Theorem 12.** *The minimal plain complexity of a program that computes a prefix-stable function mapping x to y is equal to $\mathrm{C}(y\,|\,x*) + O(1)$.*

Proof. A description mode can be considered as a family of prefix-stable functions (indexed by the first argument p). This shows that there exist a program for a prefix stable function mapping x to y of complexity at most $\mathrm{C}(y\,|\,x*) + O(1)$. On the other hand, one can efficiently "trim" all programs to make them prefix-stable; if $\hat{u}$ is the trimmed version of a program u and U is the decompressor used to define plain complexity of programs, then the set $D = \{p, x, \widehat{U(p)}(x)\colon p, x\}$ satisfies the conditions and may be considered as a decompressor in the definition of $\mathrm{C}(y\,|\,x*)$. Using this decompressor, we get the reverse inequality. ◀

More interesting question: is a similar statement true for *prefix-free* functions instead of prefix-stable ones? As we mentioned above, Theorem 6 implies that this is the case when $x = y$. (We spoke about programs that stop at x, but we may assume without loss of generality that the output is also x.) But in the general case it is not true anymore. Let us make this statement more precise. The first idea is to consider the minimal plain complexity of a program computing a prefix-free function mapping x to y. But this quantity does not look reasonable: the complexity of empty string Λ with condition x defined in this way may be arbitrarily large (and is actually the stopping time complexity of the condition x).

A more reasonable approach is to consider function $\mathrm{C}'(y\,|\,x*)$ defined as the minimal complexity of a prefix-free program that maps *some prefix of* x to y. This approach still does not work, as the following result shows.

▶ **Theorem 13.** *The inequality* $\mathrm{C}(y\,|\,x*) \leq \mathrm{C}'(y\,|\,x*)+c$ *holds for some* c *and for all* x, y. *The reverse inequality does not: there exist strings* x_i, y_i *(for* $i = 0, 1, 2, \ldots$*) such that* $\mathrm{C}(y_i\,|\,x_i*)$ *is bounded while* $\mathrm{C}'(y_i\,|\,x_i*)$ *is unbounded.*

Proof. The first part is easy: if an algorithm computing a prefix-free function f is given, we can effectively transform it into an algorithm that computes its prefix-stable extension g such that $g(x) = y$ if $f(u) = y$ for some prefix u of x.

For the second part of the proof (using game arguments in the sense of [4]) see the extended version of this paper [1]. ◀

Theorem 13 implies that the conjecture from [3, p. 254] is false, and the function C_T defined there may exceed C_E more than by $O(1)$ additive term. We do not go into the details of the definition used in [3]; let us mention only that $C_E(y_i\,|\,x_i)$ is bounded while $C_T(y_i\,|\,x_i)$ is not: for every twice prefix machine (as defined in [3, p. 252]) we get a prefix-free function if we fix the first argument (denoted there by p).

3.2 Quantitative characterization of $\mathrm{C}(y\,|\,x*)$ works only up to factor 2

In Section 2.3 we provided a quantitative characterization of stopping time complexity, or $\mathrm{C}(x\,|\,x*)$, with $O(1)$-precision (Theorem 8). The natural question is whether a similar characterization works in the general case, i.e., for $\mathrm{C}(x\,|\,y*)$. As we will see, the answer is negative.

For $\mathrm{C}(x\,|\,y)$ (the standard version of conditional complexity, with no monotonicity requirement) such a characterization is well known: $\mathrm{C}(x\,|\,y)$ is the minimal upper semicomputable function of two arguments $K(x, y)$ such that for every string y and every number n there is at most 2^n different strings x such that $K(x, y) < n$.

The natural approach is to keep this restriction and add the monotonicity requirements:

$$K(x, y0) \leq K(x, y) \text{ and } K(x, y1) \leq K(x, y), \text{ for every } x \text{ and } y.$$

We get some class of functions (that are upper semicomputable, satisfy the cardinality restriction and are monotone in the sense described). Can we characterize $\mathrm{C}(x\,|\,y*)$ as the minimal function in this class? No, as the following theorem shows.

▶ **Theorem 14.**

(a) *Function* $\mathrm{C}(x\,|\,y*)$ *belongs to this class.*

(b) *There exists a minimal (up to* $O(1)$ *additive term) function in this class;*

(c) *Function* $\mathrm{C}(x\,|\,y*)$ *is not minimal in this class: there exist a function* K *in this class, and sequences of strings* x_n *and* y_n *such that* $K(x_n, y_n) \leq n$, *but* $\mathrm{C}(x_n\,|\,y_n*) \geq 2n - c$ *for some* c *and for every* n.

(d) *The factor* 2 *that appears in the previous statement is optimal: if* K *is a function in the class (for example, the minimal one), then* $\mathrm{C}(x\,|\,y*) \leq 2K(x,y) + c$ *for some* c *and for all* x *and* y.

Proof. The statements (a) and (b) are "good news", while the statement (c) is "bad news" showing that our characterization does not work. (It is possible *a priori* that one can get a natural characterization of $\mathrm{C}(x\,|\,y*)$ by adding some other restrictions, but it is quite unclear what kind of restrictions could help here.) Finally, the statement (d) partly saves the situation and shows that the minimal function in the class and $\mathrm{C}(x\,|\,y*)$ differ at most by factor 2.

The statement (a) is obvious; note that $\mathrm{C}(x\,|\,y*)$ is bigger than $\mathrm{C}(x\,|\,y)$, so the cardinality restriction remains true. Other requirements immediately follow from the definition.

The statement (b) can be proved in a standard way. We can enumerate all functions in the class and get a uniformly computable sequence of functions $K_m(x,y)$. For that we enumerate all monotone upper semicomputable functions and then "trim" them by deleting small values that make the cardinality restriction false. Then we construct the minimal function $K(x,y)$ by letting

$$K(x,y) = \min_m (K_m(x,y) + m + 1).$$

It is upper semicomputable and monotone; for every y, the set of x such that $K(x,y) < n$ is the union of sets $\{x\colon K_m(x,y) < n - m - 1\}$ that have cardinality at most 2^{n-m-1}, and $2^{n-1} + 2^{n-2} + \ldots < 2^n$. The function K is minimal, since $K \leq K_m + m + 1$.

For the proofs of statements (c) and (d) see the extended version of this paper [1]. These proofs use game arguments in the sense of [4]. ◀

4 Questions

▶ Question 1. Imagine Turing machines with two read-only input tapes; for such a machine M consider a function f_M such that $f_M(x,y) = z$ if M stops at x and y on first and second tape respectively (reading all bits and not more) and produces z. Could we characterize the functions f_M (called *twice prefix free* in [3, page 242]) or at least their domains? Such a domain is an enumerable set of pairs that does not contain two pairs (x,y) and (x',y') where x is compatible with x' (one is a prefix of the other) and y is compatible with y'. Still this necessary condition is not sufficient, as the following argument shows. Let z_i be a computable sequence of pairwise incompatible strings (say, $z_i = 0^i1$). Let P and Q be two enumerable sets that are inseparable (do not have a decidable separating set). Consider the set of pairs that contains

- (z_i0, z_i0) for all i;
- (z_i, z_i1) for $i \in P$;
- (z_i1, z_i) for $i \in Q$.

This set satisfies the necessary condition above (does not contain two compatible pairs). However, assume that some twice prefix free machine has this set as a domain. Then it should terminate after reading z_i0 on the first tape and z_i0 on the second tape. Consider the last zero bits on both tapes. One of these bits should be read first (if they are read simultaneously, we may choose any of two). If this is the first bit, then $i \in P$ is impossible (since the machine cannot read 1 on the second tape before reading 0 on the first tape). For the same reason, $i \in Q$ is impossible if the second bit is read first. Therefore, a decidable separator exists.

Can we add some conditions to get a characterization of domains of twice prefix free machines? What do we get if we define stopping time complexity for pairs using machines of

this type? Does it have some equivalent description (for example, can it be defined using monotone-conditional complexity with pairs as conditions, Section 2.2)?

▶ Question 2. Do we have $\mathrm{C}(x\,|\,x*) = \max_z \mathrm{C}(x\,|\,z) + O(1)$ where the maximum is taken over all *finite* extensions z of x? (The problem is that the compactness argument does not work anymore.)

▶ Question 3. One may consider the function

$$K(x, y) = \max\{\mathrm{C}^Y(x) \colon Y \text{ is an infinite extension of } y\}$$

We have shown that for $x = y$ it coincides with $\mathrm{C}(x\,|\,x*)$, showing that it does not exceed $\mathrm{C}(x\,|\,x*)$ and satisfies the quantitative restrictions of Theorem 8. Both arguments remain valid (with minimal changes) for the general case, and we conclude that $K(x, y)$ defined in this way does not exceed $\mathrm{C}(x\,|\,y*)$ and also satisfies the cardinality restrictions of Theorem 14, (b). However, now these upper bound and lower bound differ, and we do not know where the function $K(x, y)$ defined as shown above lies. Does it coincide with its upper bound $\mathrm{C}(x\,|\,y*)$ for arbitrary x and y, or with its lower bound, the minimal upper semicomputable function that satisfies the cardinality requirements (see Theorem 14), or neither?

References

1 Mikhail Andreev, Gleb Posobin, and Alexander Shen. Plain stopping time and conditional complexities revisited. *CoRR*, abs/1708.08100, 2017. `arXiv:1708.08100`.

2 Mikhail Andreev, Gleb Posobin, and Alexander Shen. Stopping time complexity, abstract. *Dagstuhl Reports, Computability Theory, Dagstuhl Seminar 17081, February 2017*, page 97, 2017. `doi:10.4230/DagRep.7.2.89`.

3 Alexey V. Chernov, Marcus Hutter, and Jürgen Schmidhuber. Algorithmic complexity bounds on future prediction errors. *Information and Computation*, 205(2):242–261, 2007.

4 Andrei A. Muchnik, Ilya Mezhirov, Alexander Shen, and Nikolay Vereshchagin. Game interpretation of Kolmogorov complexity. *CoRR*, abs/1003.4712, 2010. `arXiv:1003.4712`.

5 Alexander Shen. Algorithmic variants of the notion of entropy. *Soviet Mathematics Doklady*, 29(3):569–573, 1984.

6 Alexander Shen. Around Kolmogorov complexity: Basic notions and results. In Vladimir Vovk, Harris Papadopoulos, and Alexander Gammerman, editors, *Measures of Complexity: Festschrift for Alexey Chervonenkis*, chapter 7, pages 75–116. Springer, Cham, 2015. `doi:10.1007/978-3-319-21852-6_7`.

7 Alexander Shen, Vladimir A. Uspensky, and Nikolay Vereshchagin. *Kolmogorov complexity and algorithmic randomness*, volume 220 of *Mathematical Surveys and Monographs*. American Mathematical Society, `http://www.lirmm.fr/~ashen/kolmbook.pdf`, 2017.

8 Vladimir A. Uspensky and Alexander Shen. Relations between varieties of Kolmogorov complexities. *Mathematical Systems Theory*, 29(3):271–292, Jun 1996. `doi:10.1007/BF01201280`.

9 Vladimir Vovk and Dusko Pavlovic. Universal probability-free conformal prediction. In Alexander Gammerman, Zhiyuan Luo, Jesús Vega, and Vladimir Vovk, editors, *Conformal and Probabilistic Prediction with Applications*, pages 40–47, Cham, 2016. Springer, see also `https://arxiv.org/pdf/1603.04283.pdf` (March 2016; extended version, April 2017).

Error-Tolerant Non-Adaptive Learning of a Hidden Hypergraph

Hasan Abasi
Department of Computer Science, Technion, Haifa, 32000, Israel
hassan@cs.technion.ac.il

Abstract

We consider the problem of learning the hypergraph using edge-detecting queries. In this model, the learner is allowed to query whether a set of vertices includes an edge from a hidden hypergraph. Except a few, all previous algorithms assume that a query's result is always correct. In this paper we study the problem of learning a hypergraph where α-fraction of the queries are incorrect. The main contribution of this paper is generalizing the well-known structure CFF (Cover Free Family) to be Dense (we will call it DCFF - Dense Cover Free Family) while presenting three different constructions for DCFF. Later, we use these constructions wisely to give a polynomial time non-adaptive learning algorithm for a hypergraph problem with at most α-fracion incorrect queries. The hypergraph problem is also known as both monotone DNF learning problem, and complexes group testing problem.

2012 ACM Subject Classification Theory of computation → Boolean function learning

Keywords and phrases Error Tolerant Algorithm, Hidden Hypergraph, Montone DNF, Group Testing, Non-Adaptive Learning

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.3

1 Introduction

The classical group testing model consists of a set of n items, where s of these items are defective (positive) while others are good (negative) items. The problem is to identify all defective (positive) items with a small number of group tests. A group test is a test with a negative/positive result, where you can choose an arbitrary group $S \subseteq [n]$ and ask whether S contains at least one defective item. The outcome is negative if all the items within the group are good (negative), and positive otherwise (See [13, 12] for more details about group testing).

Torney [20] was the first who generalized the group testing model to the complex group testing model, where we have the same set of n items, as in the first model, but instead of s defective elements, we have s defective groups $M_1, \cdots, M_s$, that are known as positive complexes, where $M_i \subseteq [n]$ and for any $i_1, i_2 \in [s]$ where $i_1 \neq i_2$ $M_{i_1} \not\subseteq M_{i_2}$ and $M_{i_2} \not\subseteq M_{i_1}$. In complex group testing model, a complex group test is a test with a negative/positive result, where you can choose an arbitrary group $S \subseteq [n]$ and ask whether S contains at least one positive complex. The outcome is negative if all the complexes within the group are negative, and positive otherwise. A usual assumption is that each positive complex has at most r items ($|M_i| \leq r$).

This problem is also known as graph testing, where a hypergraph $H = (V, E)$ with s edges and n vertices is given, our objective is to learn the hypergraph by asking edge-detecting queries (denote query function by Q). An edge detecting-query is a test with a negative/positive result, where you can choose an arbitrary group of vertices $S \subseteq [n]$ and ask whether this group of vertices has at least one edge or not. A usual assumption is that each edge has at most r vertices. It is easy to see that this model is equivalent to the complex

group testing model (see [1]). In addition to complex group testing and graph testing, this problem is also known as the learning monotone DNF problem with s monomials, where each monomial contains at most r variables. In this paper, we will use the latter terminology rather than complexes or graphs.

Tests can be *adaptive* (sequential), *non-adaptive* or *multi-stage*. Adaptive tests are tests that may be affected by the outcome of earlier tests. Non-adaptive tests, all tests are executed in parallel without knowledge of any other tests' outcome. Multi-stage tests are divided into stages, and all tests within each stage are executed in parallel, but different stages may be affected by the outcome of earlier stages.

This problem has many applications in chemical reactions, molecular biology and genome sequencing (see [15, 14, 7, 17, 6]). In these types of applications, an experiment corresponding to a group test could take several hours or even several days. Thus, non-adaptive algorithms are most desirable. In addition to the prolonged process, errors in applications are unavoidable. And as the number of tests increases, the number of errors occurring is increased as well. Therefore, it is very important to construct error-tolerant algorithms, where the number of the errors is a fraction of the number of queries. In all of the above applications the size of each term (rank of the hypergraph) is much smaller than the number of terms (edges), and both are much smaller than the number of variables (vertices) n. Therefore, throughout the paper, we will assume that $r < s < n$ (this assumption will be used in the constructions in Section 3) and will denote $d \triangleq r + s$.

Both adaptive and non-adaptive learning of monotone DNF problem are well-known and well-studied. In the Adaptive model, Angluin et al. [5] was the first to present a deterministic optimal adaptive algorithm for learning s-term 2-MDNF (s terms each of size 2). Later, Abasi et al., [1], gave an almost optimal adaptive algorithm for the general s-term r-MDNF case and they proved that any algorithm that learns s-term r-MDNF needs to ask at least

$$\begin{cases} r > s & \Omega\left((r/s)^{s-1} + rs\log n\right) \text{ Queries} \\ r \le s & \Omega\left((2s/r)^{r/2} + rs\log n\right) \text{ Queries} \end{cases}$$

On the non-adaptive side many works studied the s-term r-MDNF. Abasi et al. [2], gave the first deterministic algorithm which runs in polynomial time and non-adaptively learns a hypergraph that asks an almost optimal number of queries. On the other hand, only few algorithms discussed the error-tolerant problems. Stinson and Wei, [18], proved a lower bound

$$0.7c\frac{d\binom{d}{r}}{\log\binom{d}{r}}\log n + \frac{c(z-1)}{2}\binom{d}{r} = s\left(\frac{es}{r}\right)^{r+o(1)}\log n + z\left(\frac{es}{r}\right)^{r+o(1)},$$

where z is constant and the algorithm is tolerant to $z/2$ errors (constant number of errors). In the same paper they proved an upper bound of

$$O\left(z\binom{d}{r}(rs)^{\log^* n}\log n\right) = z\left(\frac{es}{r}\right)^{r+o(1)}(rs)^{\log^* n}\log n.$$

Chen et al., [11], improved the last upper bound to

$$z\left(\frac{d}{r}\right)^{r}\left(\frac{d}{s}\right)^{s}\left(1+\ln\left(\frac{n}{d}+1\right)\right) = z\left(\frac{es}{r}\right)^{r+o(1)}\log n.$$

Lang et al., [16], considered the problem where z errors occur in every m tests. They gave a random algorithm which identifies all the target function's terms with high probability. All

previous algorithms either run in exponential time, are non-deterministic or are tolerant to a constant number of errors.

In this paper we introduce the first algorithm, to the best of our knowledge, that handles α-fraction of incorrect queries which is deterministic and runs in polynomial time. In [2], the authors use the $(n,(s,r))$-cover-free family ($(n,(s,r))$-CFF). This family is a set $A \subseteq \{0,1\}^n$ of assignments such that for every distinct $i_1,\ldots,i_s,j_1,\ldots,j_r \in \{1,\ldots,n\}$, there is an assignment $a \in A$ such that

$$a_{i_1} = \cdots = a_{i_s} = 0 \text{ and } a_{j_1} = \cdots = a_{j_r} = 1.$$

In this paper we extend the definition of $(n,(s,r))$-CFF to be

$$(n,(s,r),\alpha)\text{- } \textbf{Dense CFF (DCFF)},$$

where the DCFF is a CFF with an additional requirement, which is that for every distinct $i_1,\ldots,i_s,j_1,\ldots,j_r \in \{1,\ldots,n\}$, an ***α*-fraction** of the assignments $a \in A$ satisfy the following:

$$a_{i_1} = \cdots = a_{i_s} = 0 \text{ and } a_{j_1} = \cdots = a_{j_r} = 1.$$

Improving techniques that were used in [2], by customizing them to comply with DCFF, results in the needed algorithm. In the improvement process we used what we defined above as DCFF. In order to do so, we present three constructions of DCFF, each of which has an advantage over the other. One of these constructions is superior in terms of density, while the other two have better time complexities. One of them is better when $r < T(s)$, while the other is better when $r \geq T(s)$, where T is a function of s. In addition to the extension described above, we also prove that α in $(n,(s,r),\alpha)$-DCFF cannot exceed

$$DNS(s,r) \triangleq \frac{1}{\binom{r+s}{r}} = \frac{1}{\binom{d}{r}}.$$

It is also known, [19], that any non-adaptive learning algorithm must ask at least

$$\Omega\left(\left(\frac{es}{r}\right)^r \log n\right) = \Omega\left(s^{r(1+o(1))} \log n\right)$$

queries, and therefore any algorithm must run in time $poly\,(n,s^r)$. An algorithm that runs in time $poly\,(n,s^r)$ is called efficient algorithm.

This paper is organized as follows. Section 2 gives some definitions and preliminary results that will be used throughout the paper. Section 3 gives three different algebraic constructions for Dense Cover Free family (see Section 3 for full definition). Section 4 gives an inefficient algorithm by using DCFF directly. Section 5 gives a reduction to reduce any non-linear α-error tolerant algorithm that depends on d variables at most to be linear in $\log n$. And finally in Section 6 we use all the constructions and the reduction to generate an efficient algorithm that asks

$$O\left(r^{11}\,(4s)^{r+7} \log n\right) = O\left(s^{r(1+o(1))} \log n\right)$$

noisy membership queries and runs in $poly\,(n,s^r)$, and handles

$$\Omega\left(DNS(s,r)^{1+o(1)}\right) \text{ fraction of incorrect queries.}$$

1.1 Optimality for $r = 2$ (or any constant r)

The learning problem of a graph (when $r = 2$) has been considered and well-studied. Angluin et al., [4, 5], studied the graph problem for the adaptive, nonadaptive and multistage models. Our construction and algorithm are almost optimal when $r = 2$ (or any constant r) in terms of density and query complexity, where the density in case r is constant is $\Omega\left(DNS(s,r)\right)$.

1.2 Algorithm idea overview

One can easily see that using $(n,(s,r),\alpha)$-DCFF solves the problem of s-term r-MDNF when less than $\alpha/2$-fraction of the queries can be incorrect. That is true because for each subset of s terms there are at least $\alpha/2$-fraction of the DCFF's assignments that satisfy one term (r 1's) and assigns 0 to all other terms ($s-1$ 0's) (see Section 4 for more details).
The problem with using DCFF directly is the time complexity, and that is because we need to find out whether each possible subset of s terms each of size at most r satisfies a given set of conditions or not (again, see Section 4 for more details).
In order to achieve a polynomial time complexity, we use bit-wise conjunction of assignments in each DCFF. The general idea is to create two DCFFs. One of them spreads the set of terms over different sets, and the other is used afterward with other structures and techniques to find each term explicitly (see Section 6 and Section 5 for more details).

2 Definitions and Preliminary Results

2.1 Monotone Boolean Functions

For a vector w, we denote by w_i the ith entry of w. For a positive integer j, we denote by $[j]$ the set $\{1,2,\ldots,j\}$. For two assignments $a,b\in\{0,1\}^n$ we denote by $(a\wedge b)\in\{0,1\}^n$ the bitwise AND assignment. That is, $(a\wedge b)_i = a_i\wedge b_i$ for all $i\in[n]$.

Let $f(x_1,x_2,\ldots,x_n)$ be a Boolean function from $\{0,1\}^n$ to $\{0,1\}$. We say that the variable x_i is *relevant* in f if $f|_{x_i\leftarrow 0}\not\equiv f|_{x_i\leftarrow 1}$. A variable x_i is *irrelevant* in f if it is not relevant in f. We say that the class of functions C is *closed under variable projections* if for every function $f\in C$ and every two variables x_i and x_j, $i,j\leq n$, we have $f|_{x_i\leftarrow x_j}\in C$.

For two assignments $a,b\in\{0,1\}^n$, we write $a\leq b$ if for every $i\in[n]$, $a_i\leq b_i$. A Boolean function $f:\{0,1\}^n\to\{0,1\}$ is said to be *monotone* if for every two assignments $a,b\in\{0,1\}^n$, if $a\leq b$ then $f(a)\leq f(b)$. In other words, there is no negated variables in f. Every monotone Boolean function f has a unique representation as a *reduced monotone* DNF, [3].

An *s-term r-MDNF* is a monotone DNF with at most s monomials, where each monomial contains at most r variables. It is easy to see that the class s-term r-MDNF is closed under variable projections.

For a function $f=M_1\vee\cdots\vee M_s$ that belongs to an s-term r-MDNF class, we say that $b\in\{0,1\}^n$ **separates** a term M_i in f from other terms if $M_i(b)=1$ and $M_j(b)=0$ for any $i\neq j$. In the same way, we say that $b\in\{0,1\}^n$ **separates** the terms $M_{i_1},\ldots,M_{i_k}$ in f from other terms if $M_j(b)=1$ for any $j\in\{i_1,\ldots,i_k\}$, and otherwise $M_j(b)=0$.

2.2 α-Error Tolerant Learning from Noisy Membership Queries

Consider a *teacher* that has a *target function* $f:\{0,1\}^n\to\{0,1\}$ that is an s-term r-MDNF. The teacher can answer *noisy membership queries*. That is, when receiving $a\in\{0,1\}^n$, it returns $\overline{f(a)}=1-f(a)$ for α-fraction of the queries, and $f(a)$ for the rest. This teacher is called *α-Liar Teacher*. An *α-error tolerant learning algorithm* is an algorithm that can

ask the α-*Liar Teacher* noisy membership queries. The goal of the learning algorithm is to *exactly learn* (exactly find) f with a minimum number of noisy membership queries and optimal time complexity.

2.3 Dense Perfect Hash Function, Cover Free Family and Dense Cover Free Family

▶ **Definition 1** (Perfect Hash Family)**.** Let H be a family of functions $h : [n] \to [q]$. For $d \leq q$ we say that H is an $(n, q, d, 1-\epsilon)$- *dense perfect hash family* ($(n, q, d, 1-\epsilon)$-DPHF) if for every subset $S \subseteq [n]$ of size $|S| = d$ there are at least $(1-\epsilon)|H|$ *hash functions* $h \in H$ such that $h|_S$ is injective (one-to-one) on S, i.e., $|h(S)| = d$.

▶ **Lemma 2** ([8])**.** *Let q be a power of prime. If $\epsilon > 4(d(d-1)/2+1)/q$, then there is a $(n, q, d, 1-\epsilon)$-DPHF of size*

$$O\left(\frac{d^2 \log n}{\epsilon \log \frac{\epsilon q}{d^2}}\right),$$

that can be constructed in linear time.

▶ **Definition 3** (Cover Free Family)**.** An $(n, (s, r))$- *cover free family* ($(n, (s, r))$-CFF), is a set $A \subseteq \{0, 1\}^n$, such that for every $1 \leq i_1 < i_2 < \cdots < i_d \leq n$ where $d = s + r$ and every $J \subseteq [d]$ of size $|J| = s$, there is $a \in A$ such that $a_{i_k} = 0$ for all $k \in J$ and $a_{i_j} = 1$ for all $j \in [d]\backslash J$. Denote by $N(n, (s, r))$ the minimum size of such set.

▶ **Definition 4** (Dense Cover Free Family)**.** An $(n, (s, r), \alpha)$- *dense cover free family* ($(n, (s, r), \alpha)$-DCFF), is a set $A \subseteq \{0, 1\}^n$, such that for every $1 \leq i_1 < i_2 < \cdots < i_d \leq n$ where $d = s + r$ and every $J \subseteq [d]$ of size $|J| = s$, the following holds for α-fraction vectors $a \in A$: $a_{i_k} = 0$ for all $k \in J$ and $a_{i_j} = 1$ for all $j \in [d]\backslash J$. Denote by $N(n, (s, r), \alpha)$ the minimum size of such set.

▶ **Lemma 5.** *Let A be an $(n, (s, r), \alpha)$ -DCFF, then $\alpha \leq \frac{1}{\binom{d}{r}} \triangleq DNS(s, r)$.*

Proof. For any $d = r + s$ variables, we must have exactly $\binom{d}{r}$ different assignments to cover all the possible assignments over d variables. Otherwise, one assignments at least will be missed. Denote these assignments by R. If there is at least one assignment $r_0 \in R$ that appears among $(n, (s, r), \alpha)$-DCFF more than $1/\binom{d}{r}$ times, then by pigeonhole principle, there exists at least one assignment $r_1 \in R$, that appears less than $1/\binom{d}{r}$ times. Therefore, α cannot exceed $1/\binom{d}{r}$. ◀

For completeness we show:

▶ **Lemma 6.** *Let H_1 be an $(n, q, d, 1-\epsilon)$-DPHF, and H_0 is a $(q, (r, s), \alpha)$-DCFF, then $DC = \{h_0(h_1)|h_0 \in H_0, h_1 \in H_1\}$ is an $(n, (r, s), \alpha(1-\epsilon))$-DCFF of size $|H_0||H_1|$.*

Proof. We will show that for every $1 \leq i_1 < i_2 < \cdots < i_d \leq n$ where $d = s + r$ and for every $J \subseteq [d]$ of size $|J| = s$ there are $\alpha(1-\epsilon)|H_0||H_1|$ vectors $h \in DC$ such that $h(i_k) = 0$ for all $k \in J$, and $h(i_j) = 1$ for all $j \in [d]\backslash J$.
Since H_1 is $(n, q, d, 1-\epsilon)$-DPHF, then there are $(1-\epsilon)|H_1|$ functions $h_1 \in H$ where $h_1(i_1), \ldots, h_1(i_d)$ are distinct. And since H_0 is a $(q, (r, s), \alpha)$-DCFF, then for each $h_1(i_1), \ldots, h_1(i_d)$ distinct values there are $\alpha|H_0|$ vectors $h_0 \in H_0$, where for every $J \subseteq [d]$ of size $|J| = s$, such that $h_0(h_1(i_k)) = 0$ for all $k \in J$ and $h_0(h_1(i_j)) = 1$ for all $j \in [d]\backslash J$. Accordingly, there are $(1-\epsilon)|H_1| \cdot \alpha|H_0|$ vectors $h \in DC$ where for every $1 \leq i_1 < i_2 < \cdots < i_d \leq n$ where $d = s + r$ and for every $J \subseteq [d]$ of size $|J| = s$, $h(i_k) = 0$ for all $k \in J$ and $h(i_j) = 1$ for all $j \in [d]\backslash J$. ◀

Table 1 Dense CFF, assuming $r \leq O\left((\log^2 d)/(\log\log d)\right)$.

Constr.	Construction Time	Size	Density
I	$O\left(d^{\sqrt{\frac{r}{\log r}}}\left(\frac{d}{r}\right)^r n\log n\right)$	$O\left(d^{\sqrt{\frac{r}{\log r}}}\left(\frac{d}{r}\right)^r \log n\right)$	$\Omega\left(DNS(s,r)d^{-\sqrt{\frac{r}{\log r}}}\right)$
II	$O\left(c^d \cdot d^{2d+6}s^2\left(\frac{es}{r}\right)^{2r} n\log n\right)$	$O\left(d^3 s^2\left(\frac{es}{r}\right)^r \log n\right)$	$\Omega\left(DNS(s,r)\right)$
III	$O\left(d^3\left(\frac{ed^3}{r}\right)^{r+2}(4s)^{2r+4} n\log n\right)$	$O\left(d^3(4s)^{r+2}\log n\right)$	$\Omega\left(\frac{DNS(s,r)}{(c\cdot r)^r}\right)$

3 Explicit Dense Cover Free Family Constructions

In this section we will show three different constructions of $(n,(r,s),\alpha)$-DCFF - each of which has an advantage on the other. Before we introduce the constructions, we will state the results for each construction in **Table** 1.

Note that construction II is the best construction in terms of density. When r is non-constant then construction I and III are much better than construction II in terms of time complexity. Accordingly, when

$$\omega(1) < r < O\left(\frac{\log^2 d}{\log\log d}\right),$$

then we prefer construction III over I. Otherwise we prefer construction I over III.

In our algorithms, we will distinguish between the cases r is constant and r is non-constant. When r is non-constant, we will only deal with the case where $\omega(1) < r < O\left((\log^2 d)/(\log\log d)\right)$, and it can be done similarly for the other case.

3.1 Explicit Construction

Before we start our explicit constructions, we will start with lemmas from [10, 9], that are necessary for our constructions.

▶ **Lemma 7** ([9] - Derandomization lemma). *Let S be a finite sample space with a probability distribution D.*
Let $X_1,\ldots,X_N$ be random variables over S that take values from $\{0,1\}$. Suppose that for every $s \in S = S_1 \times S_2 \times \cdots \times S_n$, the values $X_1(s),\ldots,X_N(s)$ can be computed in $\tilde{O}(N)$[1]. Let

- $N' \leq N, 0 < \epsilon \leq \frac{1}{2}$.
- $\lambda_i = (1-\epsilon)p_i$, *where* $0 < p_i \leq E_{s\sim D}[X_i(s)]$ *for all* $i \in [N'] := \{1,\ldots,N'\}$.
- $\lambda_j = (1+\epsilon)p_j$, *where* $1 > p_j \geq E_{s\sim D}[X_j(s)]$ *for all* $j \in (N',N] := \{N'+1,\ldots,N\}$.
- $\alpha = \min_i \min\left(1/(1-p_i), 1/(1-\lambda_i)\right)$ *and* $m \geq \frac{4\ln N}{\min_i p_i\epsilon^2}$.

If any expectation of the form $E[X_i(x_1,\ldots,x_n)|x_1=\xi_1,\ldots,x_j=\xi_j]$ can be computed in time T, then there is an algorithm that runs in time

$$\tilde{O}\left(T\left(|S_1|+\cdots+|S_n|\right)\cdot Nm^2\log\alpha\right), \text{ and outputs } S' = \{s_1,\ldots,s_{m+1}\} \subseteq S$$

such that for all $i \in [N']$ and $j \in (N',N]$: $E_{s\sim U_{S'}}[X_i(s)] \geq \lambda_i, E_{s\sim U_{S'}}[X_j(s)] \leq \lambda_j$, where $U_{S'}$ is the uniform distribution over S'.

[1] $\tilde{O}(N)$ is $O(N\cdot poly(\log T))$ where T is the time complexity of the construction.

In other words, the lemma says if you have a set of random variables over a big range S, in our case $|S| = 2^n$, then there is a smaller set S' that can replace S while keeping the expectation of each variable the same to the factor of $(1 \pm \epsilon)$. This lemma will help us convert a random algorithm to a deterministic one, since the set S' is "small" and we can go over all possible values in a polynomial time.

▶ **Lemma 8** ([10] - CFF explicit construction). *Fix any integers $r < s < d$ with $d = r+s$, there is an almost optimal $(n, (r, s))$-CFF, i.e., of size $N(r, s)^{1+o(1)} \log n$, where $N(r, s) = \frac{d\binom{d}{r}}{\log\binom{d}{r}}$, that can be constructed in linear time.*

▶ **Corollary 9.** *There is an $\left(n, (r, s), \frac{1}{N(r,s)^{1+o(1)} \log n}\right)$-DCFF of size $N(r, s)^{1+o(1)} \log n$, that can be constructed in linear time.*

3.2 First Construction

From the proof of Lemma 8, we infer that the accurate value of the size is

$$N(n) = t(d) \left(\frac{d}{r}\right)^r \log n.$$

where

$$t(d) = \begin{cases} 2^{O(r \log\log d)} & r > O\left(\frac{\log^2 d}{\log\log d}\right) \\ 2^{O\left(\frac{\sqrt{r}\log d}{\sqrt{\log r}}\right)} & \text{Otherwise} \end{cases}$$

Now we can build the first DCFF by Lemma 2, Lemma 6 and Lemma 8.
Simply choose $\epsilon = 0.01$ (or any small constant) and a prime $q = O(d^3)$. By Lemma 2, there is a $(n, q, d, \Omega(1))$-DPHF of size $O\left(d^2 \log n\right)$. By Corollary 9 and the claim above, for $q = O\left(d^3\right)$, there is

$$\text{an } \left(O\left(d^3\right), r, s, \Omega\left(\frac{DNS(s,r)}{t(d)}\right)\right)\text{-DCFF, that can be constructed in linear time.}$$

By using Lemma 6, we can get

$$\left(n, r, s, \Omega\left(\frac{DNS(s,r)}{t(d)}\right)\right)\text{-DCFF of size } O\left(t(d)\left(\frac{d}{r}\right)^r \log n\right).$$

▶ **Theorem 10.** *There is an $\left(n, (r, s), \Omega\left(\frac{DNS(s,r)}{t(d)}\right)\right)$-DCFF of size $O\left(d^2 \log d N(r, s)^{1+o(1)} \log n\right)$ that can be constructed in linear time.*

3.3 Second Construction

▶ **Theorem 11.** *There is an $(n, (r, s), \Omega\left(DNS(s, r)\right))$-DCFF of size $O\left(s\left(\frac{es}{r}\right)^r \log n\right)$ that can be constructed in poly $\left(n^d\right)$.*

Proof. We define a probability space over the vectors $x = (x_1, \ldots, x_n) \in \{0, 1\}^n$, where

$$x_i = \begin{cases} 0, & \text{with probability } \frac{s}{d} \\ 1, & \text{with probability } \frac{r}{d} \end{cases}$$

For every couple of sets $\{i_1, \ldots, i_r\}$, $\{j_1, \ldots, j_s\}$, we define the random variable $X_{i_1,\cdots,i_r,j_1,\cdots,j_s}$ where

$$X_{i_1,\cdots,i_r,j_1,\cdots,j_s}(x) = \begin{cases} 1, & \text{If } x_{i_1} = \cdots = x_{i_r} = 1, x_{j_1} = \cdots = x_{j_s} = 0. \\ 0, & \text{Otherwise} \end{cases}$$

Note that the number of random variables is $N = \binom{n}{d}\binom{d}{r}$, and that the expectation of each random variable is $E[X_{i_1,\cdots,i_r,j_1,\cdots,j_s}] = \left(\frac{r}{d}\right)^r \left(\frac{s}{d}\right)^s$.
We now use Lemma 7. Note that

$$N' = N, \epsilon = \tfrac{1}{2}, p_{i_1,\cdots,i_r,j_1,\cdots,j_s} = p := \left(\tfrac{r^r s^s}{d^d}\right).$$

$$\lambda_i = \lambda := \tfrac{1}{2}p, \quad m \geq \frac{4\ln\binom{n}{d}\binom{d}{r}}{\frac{p}{4}} = O\left(s\left(\tfrac{es}{r}\right)^r \log n\right).$$

Therefore, according to Lemma 7, there is an algorithm that runs in

$$\tilde{O}(nNm^2\log\alpha) = O\left(n\left(\tfrac{en}{d}\right)^d s^2 \left(\tfrac{es}{r}\right)^{2r}\log^2 n\right) \text{ time, and outputs}$$

$S' = \{s_1,\ldots,s_{m+1}\} \subseteq S$, such that, for all $i = (i_1,\ldots,i_r,j_1\ldots,j_s) \in [n]^d$, holds:
$E_{s\sim U_{S'}}[X_i(s)] \geq \frac{1}{2}\left(\frac{r}{d}\right)^r \left(\frac{s}{d}\right)^s = \Omega\left(DNS(s,r)\right)$ ◀

The size of the construction is almost optimal, while the running time is exponential. In the third construction we will reduce the exponent from d to r.

3.4 Third Construction

▶ **Theorem 12.** *There is an* $\left(n,(r,s),\Omega\left(\frac{DNS(s,r)}{(cr)^r}\right)\right)$*-DCFF of size* $O\left((4s)^{r+2}\log n\right)$ *that can be constructed in poly* (n^r).

Through applying the same technique from the **Second construction** wisely, we reduce the number of random variables this time, leading to a reduced construction time.

Proof. We define a probability space over the vectors $x = (x_1,\ldots,x_n) \in \{0,1\}^n$, where

$$x_i = \begin{cases} 1, & \text{with probability} \frac{1}{4s} \\ 0, & \text{with probability} \frac{4s-1}{4s}. \end{cases}$$

For every couple of sets $\{i_1,\ldots,i_r\}$, $\{j_1,\ldots,j_s\}$, we define the following random variables:

$$X_{i_1,\cdots,i_r,j_1,\cdots,j_s}(x) = \begin{cases} 1, & \text{If } x_{i_1} = \cdots = x_{i_r} = 1, x_{j_1} = \cdots = x_{j_s} = 0. \\ 0, & \text{Otherwise.} \end{cases}$$

$$Y_{i_1,\cdots,i_r,j}(x) = \begin{cases} 1, & \text{If } x_{i_1} = \cdots = x_{i_r} = x_j = 1. \\ 0, & \text{Otherwise.} \end{cases}, j \in \{j_1,\ldots,j_s\}$$

$$Z_{i_1,\cdots,i_r}(x) = \begin{cases} 1, & \text{If } x_{i_1} = \cdots = x_{i_r} = 1. \\ 0, & \text{Otherwise.} \end{cases}$$

Note that we can bound the random variable X, by an expression of Y and Z in the following way:

$$X_{i_1,\cdots,i_r,j_1,\cdots,j_s} \geq Z_{i_1,\cdots,i_r} - \textstyle\sum_{k=1}^{s} Y_{i_1,\cdots,i_r,j_k}$$

The correctness of the above equation follows immediately from the definition.
Note that instead of $\binom{n}{d}$ that we used in the previous construction, now we can do a similar construction by using the following number of random variables

$$N = \tbinom{n}{r+1}(r+1) + \tbinom{n}{r}$$

and the expectation of each random variable is

$$E[X_{i_1,\cdots,i_r,j_1,\cdots,j_s}] \geq E[Z_{i_1,\cdots,i_r}] - \sum_{k=1}^{s} E[Y_{i_1,\cdots,i_r,j_k}] = \frac{3}{4}\frac{1}{(4s)^r}$$

Again we use Lemma 7 for the random variables Z and Y. Note that

1. $N' = \binom{n}{r}$, $\epsilon = \frac{1}{2}$.
2. $p_{i_1,\cdots,i_r} = p_0 := \left(\frac{1}{4s}\right)^r$.
3. $p_{i_1,\cdots,i_r,j} = p_1 := \left(\frac{1}{4s}\right)^{r+1}$.
4. $\lambda_{i_1,\cdots,i_r} = \lambda_0 := \frac{1}{2}p_0$.
5. $\lambda_{i_1,\cdots,i_r,j} = \lambda_1 := \frac{3}{2}p_1$.
6. $m \geq \frac{4\ln\left(\binom{n}{r+1}(r+1)+\binom{n}{r}\right)}{\frac{p_1}{4}} = O\left((4s)^{r+2}\log n\right)$.

Therefore, according to Lemma 7, there is an algorithm that runs

$$\tilde{O}\left(nNm^2\log\alpha\right) = O\left(n\left(\frac{en}{r}\right)^{r+2}(4s)^{2r+4}\log^2 n\right), \text{ and outputs}$$

$$S' = \{s_1,\ldots,s_{m+1}\} \subseteq S,$$

such that for all $i = (i_1,\ldots,i_r) \in [n]^r, j = (j_1,\ldots,j_r,j_{r+1}) \in [n]^{r+1}$:

$$E_{s\sim U_{S'}}[Z_i(s)] \geq \tfrac{1}{2}(\tfrac{1}{4s})^r \text{ and } E_{s\sim U_{S'}}[Y_j(s)] \leq \tfrac{3}{2}(\tfrac{1}{4s})^{r+1}, \text{so}$$

$$E[X_{i_1,\cdots,i_r,j_1,\cdots,j_s}] \geq E[Z_{i_1,\cdots,i_r}] - \sum_{k=1}^{s} E[Y_{i_1,\cdots,i_r,j_k}] \geq \tfrac{1}{2}(\tfrac{1}{4s})^r - s\cdot\tfrac{3}{2}(\tfrac{1}{4s})^{r+1}$$

$$E[X_{i_1,\cdots,i_r,j_1,\cdots,j_s}] \geq \tfrac{1}{8}(\tfrac{1}{4s})^r$$ ◀

Note that we can apply Lemma 2 and Lemma 6 on the second and third constructions, the same way we did in the first construction to get the needed results.

4 Direct Usage of DCFF: α-Error Tolerant Algorithm

In this section we discuss and explain the intuition behind using DCFF, and show how we can use it directly to present an α-error tolerant algorithm of a hidden hypergraph. Unfortunately though, the time complexity of the algorithm will not be polynomial. The issue of the time complexity will be solved in the following sections by wise use of the DCFF combined with other algebraic structures.

We assume our function is $f = M_1 \vee \cdots \vee M_{s'}$, $s' \leq s$, and each M_i is of size at most r. For simplicity we assume that $s' = s$ and each term is of size exactly r. Denote $S_{terms} = \{x_{i_1}\cdots x_{i_r} | 1 \leq i_1 < \ldots < i_r \leq n \text{ and } x_{i_1}\cdots x_{i_r} \notin \{M_1,\ldots,M_s\}\}$.
In order to find M_1 we need a set of assignments A_{M_1} such that:

1. $\exists a \in A_{M_1}$ such that $M_1(a) = 1, M_i(a) = 0$ $(1 < i \leq s)$.
2. $\forall t \in S_{terms}$ $\exists a \in A_{M_1}$ where $t(a) = 1$ and $f(a) = 0$.

If all the above is satisfied, then we can find M_1 simply by trying all the possible terms of size r. One can easily see that any $(n,(s,r),\alpha)$-DCFF(CFF) has a subset that satisfies all the above requirements. To satisfy the first requirement we need r $1's$ (to satisfy M_1) and $s-1$ $0's$ (to falsify $M_i, 1 < i \leq s$), and to satisfy the second requirement we need s $0's$ (to falsify $M_i, 1 \leq i \leq s$) and r $1's$ (to satisfy a given term $t \in S_{terms}$).
Now we present the details of the algorithm:

We construct an $(n,(s,r),\beta)$-DCFF A, where $\beta = 2\alpha + \frac{1}{|A|}$. Denote by $A'(M)$ the set $\{a \in A | M(a) = 1 \text{ and } Q'(a) = 1\}$. Now, take every monomial M of size at most r where $|A'(M)| \leq \alpha|A|$. The disjunction of all such monomials is equivalent to the target function.

This follows from the following two facts: (1) for any monomial M' such that $M' \not\Rightarrow f$, there are at least $\left(\alpha + \frac{1}{|A|}\right)|A|$ assignments $a \in A$ such that $Q'(a) = 0$ and $M'(a) = 1$ (2) for any monomial $M' \Rightarrow f$, there are at least $\left(\alpha + \frac{1}{|A|}\right)|A|$ assignments $a \in A$ such that $Q'(a) = 1$ and $M'(a) = 1$. Since only $\alpha|A|$ queries can be incorrect, and there are $\beta|A|$ assignments where $M'(a) = 1$ and $Q(a) = 0$ for each $M' \not\Rightarrow f$, then there are more than $\alpha|A|$ assignments $a \in A$ where $Q(a) = 0$ and $M'(a) = 1$. Similarly, for each $M' \Rightarrow f$ there are at most $\alpha|A|$ assignments $a \in A$ where $Q(a) = 0$ and $M'(a) = 1$.

▶ **Lemma 13.** *If an* $\left(n, (s, r), 2\alpha + \frac{1}{|A|}\right)$*-DCFF* A *of size* N *can be constructed in time* T*, then there is an algorithm that learns the class* s*-term* r*-MDNF with* N *noisy membership queries in time*

$$O\left(T + Nr\sum_{r'=0}^{r}\binom{n}{r'}\right).$$

Correctness and time complexity of the algorithm follows from the above explanation and from the correctness of the error-free folklore algorithm [2].

5 α-Error Tolerant Reduction

In this section, we give a reduction to reduce the query complexity of any α-error tolerant algorithm that depends on d variables to become linear in $\log n$. The idea behind this is to map the n-dimensional variables space to $g(d)$-dimensional space, where g is a polynomial function of d. Then we run the original algorithm over the new space. The main challenge here is to recover the original function efficiently (in terms of time complexity), while keeping the algorithm $\Omega(\alpha)$-error tolerant. This can be done by using PHF in the following way: by the definition of PHF, it contains a function h such that h maps each variable of MDNF to a distinct variable. After finding this h, we use it to reduce the variables space from n-dimensional to $g(d)$-dimensional. Here is the main theorem of this section:

▶ **Theorem 14.** *Let* H *be a class of boolean functions that is closed under variable projection. And suppose there is an algorithm that, given* $f \in H$ *as an input, finds the relevant variables of* f *in time* $R(n)$*.*

If H *is non-adaptively learnable in time* $T(n)$ *with* $Q'(n)$ *noisy membership queries, and* $\alpha Q'(n)$*-errors might occur, then* H *is non-adaptively learnable in time*

$$O\left(\frac{d^2 n\log n}{\epsilon\log(\epsilon q/d^3)} + \frac{d^2\log n}{\epsilon\log(\epsilon q/d^3)}(T(q) + Q'(q)n + R(q))\right), \text{ with}$$

$$Q_{new}(n) = O\left(\frac{d^2 Q'(q)}{\epsilon\log(\epsilon q/d^3)}\log n\right)$$

noisy membership queries where less than $\frac{1-\epsilon}{2}\alpha Q_{new}(n)$ *errors might occur, and* d *is an upper bound on the number of relevant variables in* $f \in C$ *and* q *is any integer such that* $q \geq 2(d+1)^2$*.*

Consider the algorithm in Figure 1.

Let $\mathcal{A}(n, \alpha)$ be a non-adaptive algorithm that learns H in $T(n)$ time with $Q'(n)$ noisy membership queries, where $\alpha Q'(n)$ errors might occur. Let $f \in H_n$ be the target function. Consider the $(n, q, d+1, 1-\epsilon)$-DPHF P that was constructed in Lemma 2 (Step 1 in the algorithm).

The following lemma follows from the fact that H is closed under variable projection:

Reduction
$\mathcal{A}(n,\alpha)$ is a non-adaptive α error tolerant learning algorithm for H.
1) Construct an $(n, q, d+1, (1-\epsilon))$-DPHF P.
2) For each $h \in P$
    Run $\mathcal{A}(q,\alpha)$ to learn $f_h := f(x_{h(1)}, \ldots, x_{h(n)})$.
    Let $f'_h \in H$ be the output of $\mathcal{A}(q,\alpha)$.
3) For each $h \in P$
    $V_h \leftarrow$ the relevant variables in f'_h
4) $G(h) = |\{h' || V_{h'}| = |V_h|, h' \in P\}|$, $H' = \{h \in H | G(h) \geq \frac{1-\epsilon}{2}|P|\}$
    $d_{max} \leftarrow \max_{h \in H'} |V_h|$.
5) $X \leftarrow \{x_1, x_2, \ldots, x_n\}$.
6) For each $i \in [n]$, $W(i) = |\{h \in H | x_{h(i)} \notin V_h, |V_h| = d_{max}\}|$
    If $W(i) \geq \frac{1-\epsilon}{2}|P|$, then $X \leftarrow X \backslash \{x_i\}$
7) Take all $h \in H$ with $|V_h| = d_{max}$
8) Replace each relevant variable x_i in f'_h by $x_j \in X$ where $h(j) = i$.
9) Output the function that appears at least $\frac{1-\epsilon}{2}|P|$ times from step (8).

Figure 1 Reduction.

▶ **Lemma 15.** *For every $h \in P$, the function $f_h := f(x_{h(1)}, \ldots, x_{h(n)})$ is in H_q.*

▶ **Lemma 16.** *The total number of queries after the reduction is $Q_{new} = |P|Q'(q)$.*

Proof. For each $h \in P$, we run $\mathcal{A}(q,\alpha)$ to learn f_h. It is known that $\mathcal{A}(q,\alpha)$ generates $Q'(q)$ queries, so overall we have $|P|Q'(q)$ queries. ◀

▶ **Lemma 17.** *(Step 2 in the algorithm) Let f'_h be the output of algorithm $\mathcal{A}(q,\alpha)$ when it runs on f_h, and let $K = \{h | h \in P \text{ and } f'_h \not\equiv f_h\}$, then $K < \frac{1-\epsilon}{2}|P|$.*

Proof. The lemma follows from the fact that the number of times when $\alpha Q'(q)$ errors appear while running $A(q,\alpha)$ is less than $\frac{1-\epsilon}{2}|P|$. Otherwise, the number of errors will be at least $\frac{1-\epsilon}{2}\alpha|P|Q'(q) = \frac{1-\epsilon}{2}\alpha Q_{new}(n)$. ◀

(Step 3 in the algorithm) Note that if the number of errors when running $A(q,\alpha)$ is less than $\alpha Q'(q)$-errors, then the algorithm finds the correct relevant variables of f'_h, by the definition of $A(q,\alpha)$.

▶ **Lemma 18.** *(Step 4 in the algorithm) Suppose $x_{i_1}, \ldots, x_{i_{d'}}$, $d' \leq d$ are the relevant variables in the target function f, then $d_{max} = d'$.*

Proof. Let V_h be the set of relevant variables of f'_h and let $d_{max} = \max_{h \in H'} |V_h|$. By the definition of P there are at least $(1-\epsilon)|P|$ maps $h' \in P$ such that $h'(i_1), \ldots, h'(i_{d'})$ are distinct, and since more than $\alpha Q'(q)$ errors might occur in less than $\frac{1-\epsilon}{2}|P|$ functions $h \in H$ (as mentioned above), then there are at least $(1-\epsilon)|P| - \frac{1-\epsilon}{2}|P| = \frac{1-\epsilon}{2}|P|$ functions $h' \in P$ where $h'(i_1), \ldots, h'(i_{d'})$ are distinct and their $V_{h'}$ equals d'. Such h' satisfies $G(h') \geq \frac{1-\epsilon}{2}|P|$, so $h' \in H'$. Stemming from the same fact, there are less than $\frac{1-\epsilon}{2}|P|$ functions $h \in P$ where the learned function f'_h can have number of relevant variables greater than d'. Such h satisfies $G(h) < \frac{1-\epsilon}{2}|P|$, so $h \notin H'$. And therefore, $d_{max} = d'$. ◀

▶ **Lemma 19.** *(Step 6 in the algorithm) If x_i is an irrelevant variable, then for at least $\frac{1-\epsilon}{2}|P|$ maps $h \in P$ $|V_h| = d_{max}$ and $x_{h(i)} \notin V_h$.*

Learn(f_{MDNF})
1) Construct an $(n,(1,r),\alpha_1)$-DCFF A and an $(n,(s-1,r),\alpha_2)$-DCFF B.
2) Ask noisy membership queries for all $a \wedge b$, $a \in A$ and $b \in B$.
3) For every $b \in B$.
4) $\quad T_b \leftarrow 1$.
5) $\quad$ For every $i \in [n]$.
6) $\quad\quad$ If for at most $\beta_1|A|$ assignments $a \in A$, $(a \wedge b)_i < Q'(a \wedge b)$
7) $\quad\quad\quad$ then $T_b \leftarrow T_b \wedge x_i$.
8) $\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{T_b\}$.
9) Remove from $\mathcal{T}$ all the terms that appear at most $\beta_2|B|$-times.
10) Remove from $\mathcal{T}$ the term $\wedge_{i\in[n]}x_i$
$\quad$ and all subterms of a larger term.

Figure 2 An algorithm for learning s term r MDNF.

Proof. Consider any irrelevant variable $x_j \notin \{x_{i_1},\ldots,x_{i_{d'}}\}$. Since P is $(n,q,d+1,1-\epsilon)$-DPHF, there are $(1-\epsilon)|P|$ functions $h'' \in P$ such that $h''(j)$ and $h''(i_1),\ldots,h''(i_{d'})$ are distinct. Again since αQ_{new} errors might occur, then $f'_{h''}$ depends on $x_{h''(i_1)},\ldots,x_{h''(i_{d'})}$, and not on $x_{h''(j)}$ and $|V_h| = d_{max}$ for at least $\frac{1-\epsilon}{2}|P|$ functions $h'' \in P$. ◀

This way the irrelevant variables can be eliminated. Since the above is true for every irrelevant variable, the set X contains only the relevant variables of f, after Step 6 in the algorithm. Then in Steps 7 and 8, we find all functions f for which the number of relevant variables is d'. The target function appears at least $\frac{1-\epsilon}{2}|P|$ times, and any other function appears less than $\frac{1-\epsilon}{2}|P|$ times. This is Step 9 in the algorithm. The correctness of the algorithm follows immediately from the above lemmas and explanation.

6 Efficient α-Error Tolerant Algorithm

In Section 4 we used DCFF directly to present a non-polynomial algorithm. In order to improve the time complexity we will now use two DCFFs. One of them is used to spread the set of terms over different sets, and the other is used afterwards to find each term explicitly. The complexity of this process is the bit-wise conjunction of assignments in each DCFF is $f(r,s)\log^2 n$, where $f(r,s)$ is a monotone function of s and r. In order to make our algorithm linear in $\log n$, we use the reduction from the previous section.

▶ **Theorem 20.** *Let A be an $(n,(1,r),\alpha_1)$-DCFF and B be an $(n,(s-1,r),\alpha_2)$-DCFF, and let $\beta_1 \triangleq \frac{\alpha_1}{2} - \frac{1}{|A|}, \beta_2 \triangleq \frac{\alpha_2}{2} - \frac{1}{|B|}$. There is a non-adaptive α-error tolerant learning algorithm for s-term r-MDNF that asks all the queries $A \wedge B$ and finds the target function, when $\alpha = \beta_1\beta_2$.*

This gives the algorithm in Figure 2.

▶ **Lemma 21.** *If $b \in B$ **separates** a term T in f from other terms, and at most $\beta_1|A|$ errors can occur, then $T_b = T$.*

Proof. According to the definition of A, for each distinct $i_1,\ldots,i_r$ and j_1 there are $\alpha_1|A|$ assignments $a \in A$, where $a_{i_1} = \cdots = a_{i_r} = 1$ and $a_{j_1} = 0$. Since $\beta_1|A|$ of the queries might have incorrect results, then each $x_i \in T$ can appear as $x_i = 1$ while also $Q = 1$ in at least

$\alpha_1 - \beta_1 > \beta_1$ fraction of the assignments. Each $x_i \notin T$ can appear as $x_i = 0$ while also $Q = 1$ in at least $\alpha_1 - \beta_1 > \beta_1$ fraction of the assignments. So by the definition of T_b, $T_b = T$. ◀

▶ **Lemma 22.** *Let $A_b = \{a \wedge b | a \in A\}$. If $\alpha \triangleq \beta_1\beta_2$ is the fraction of queries that can be incorrect, then for at most $\beta_2|B|$ assignments $b \in B$, A_b contains more than $\beta_1|A|$ errors.*

Proof. If there are more than $\beta_2|B|$ $b \in B$ such that A_b contains more than $\beta_1|A|$ errors, then the fraction of errors is bigger than $\frac{(\beta_2|B|)(\beta_1|A|)}{|A||B|} = \beta_1\beta_2$. ◀

Now we are ready to prove Theorem 20

Proof. Let $f = M_1 \vee M_2 \vee \cdots \vee M_s$ be the target function. For every $b \in B$, let $F_b(i) = \{a | a \in A, (a \wedge b)_i < Q'(a \wedge b)\}$, $A_b = \{a \wedge b | a \in A\}$, $I_b = \{i || F_b(i)| \leq \beta_1|A|\}$, and T_b be the following term: $T_b := \wedge_{i \in I_b} x_i$. We will show:

1. For each T in f, there are more than $\beta_2|B|$ assignments $b \in B$, such that $T_b = T$.
2. Every other term T_b that appears more than $\beta_2|B|$ times, is either equal to $\wedge_{i\in[n]}x_i$ or to a subterm of one of the terms in f.

In case no errors occur, the term that can be learned from each block A_b is either one of the following:

1. In case b separates a term in f then I_b will be the separated term.
2. In case b separates k terms in f then I_b will be a subterm of one of the terms in f (the intersection of the k terms).
3. In case b separates no term in f then I_b will be $\wedge_{i\in[n]}x_i$.
   (For more details see Lemma 13 in [2]).

By Lemma 22, the number of assignments $b \in B$ where A_b has more than $\beta_1|A|$ errors is $\beta_2|B|$, so any term different than a term in f, a subterm of one of the terms in f and $\wedge_{i\in[n]}x_i$ appears at most $\beta_2|B|$ times.

Now we prove that each term T in f appears more than $\beta_2|B|$ times.

Since there are $\alpha_2|B|$ assignments $b \in B$ that separate T from all other terms in f, among which at most $\beta_2|B|$ assignments A_b can have more than $\beta_1|A|$ errors, then by Lemma 21 $(\alpha_2 - \beta_2)|B| > \beta_2|B|$ of A_b will return $T_b = T$. ◀

▶ **Theorem 23.** *Fix any integers $r < s < d$ with $d = r + s$, and let $d \leq n$. There is a non-adaptive proper α error tolerant learning algorithm for s-term r-MDNF that asks $O\left(r^9(4s)^{r+5}\log^2 n\right)$ queries and runs in time $poly(n, s^r)$, with $\alpha = \Omega\left(\frac{1}{r^r}DNS(1,r)DNS(s-1,r)\right)$ -fraction of the queries might be incorrect.*

Proof. By Theorem 11 (**second construction**), we can construct a $(n, (1, r), \alpha_1)$-DCFF, of size $|A| = O\left(r^6 \log n\right)$, where $\alpha_1 = \Omega\left(DNS(1, r)\right)$. And by Theorem 12 (**third construction**), we can construct a $(n, (s-1, r), \alpha_2)$-DCFF of size $|B| = O\left(d^3\,(4s)^{r+2}\log n\right)$, where $\alpha_2 = \Omega\left(\left(\frac{1}{r^r}\right)DNS(s-1,r)\right)$.

The construction of the above DCFFs takes $poly\,(n, |A|, |B|)$ time. By Theorem 20, the learning takes $|A \wedge B| \cdot n = poly\ (n, |A|, |B|)$ time. The number of queries of the algorithm is $|A \wedge B| \leq |A| \cdot |B| = O\left(r^9(4s)^{r+5}\log^2 n\right)$. ◀

We are now ready to prove the main result:

▶ **Theorem 24.** *Fix any integers $r < s < d$ with $d = r + s$, and let $d \leq n$. There is a non-adaptive proper α error tolerant learning algorithm for s-term r-MDNF that asks*

$$O(r^{11}(4s)^{r+7}\log n)$$

queries where $\alpha = \Omega\left(\frac{1}{r^r}DNS(1,r)DNS(s-1,r)\right)$-fraction of the queries might return incorrect result, and runs in time $(n \log n) \cdot poly(s^r)$.

Proof. We use Theorem 14. H is the class of s-term r-MDNF. This class is closed under variable projection. Given f that is s-term r-MDNF, one can find all the relevant variables in $R(n) = O(sr)$ time. The algorithm in the previous section runs in time $T(n) = poly(n, s^r)$ and asks $Q'(n) = O(r^9(4s)^{r+5}\log^2 n)$ queries. The number of variables in the target is bounded by $d = rs$. Let $q = O\left(d^3\right) \geq 2d^2$. By Theorem 14, there is a non-adaptive algorithm that runs in time $O\left(qd^2 n\log n + \frac{d^2\log n}{\log(q/d^2)}(T(q)n + R(q))\right) = (n\log n)poly(s^r)$, and asks $O\left(\frac{d^2Q'(q)}{\log(q/d^2)}\log n\right) = O(r^{11}(4s)^{r+7}\log n)$, noisy membership queries. ◀

References

1 Hasan Abasi, Nader H. Bshouty, and Hanna Mazzawi. On exact learning monotone DNF from membership queries. In Peter Auer, Alexander Clark, Thomas Zeugmann, and Sandra Zilles, editors, *Algorithmic Learning Theory - 25th International Conference, ALT 2014, Bled, Slovenia, October 8-10, 2014. Proceedings*, volume 8776 of *Lecture Notes in Computer Science*, pages 111–124. Springer, 2014. doi:10.1007/978-3-319-11662-4_9.

2 Hasan Abasi, Nader H. Bshouty, and Hanna Mazzawi. Non-adaptive learning of a hidden hypergraph. *Theor. Comput. Sci.*, 716:15–27, 2018. doi:10.1016/j.tcs.2017.11.019.

3 Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987. doi:10.1007/BF00116828.

4 Dana Angluin and Jiang Chen. Learning a hidden hypergraph. *Journal of Machine Learning Research*, 7:2215–2236, 2006. URL: http://www.jmlr.org/papers/v7/angluin06a.html.

5 Dana Angluin and Jiang Chen. Learning a hidden graph using o(logn) queries per edge. *J. Comput. Syst. Sci.*, 74(4):546–556, 2008. doi:10.1016/j.jcss.2007.06.006.

6 Richard Beigel, Noga Alon, Simon Kasif, Mehmet Serkan Apaydin, and Lance Fortnow. An optimal procedure for gap closing in whole genome shotgun sequencing. In Thomas Lengauer, editor, *Proceedings of the Fifth Annual International Conference on Computational Biology, RECOMB 2001, Montréal, Québec, Canada, April 22-25, 2001*, pages 22–30. ACM, 2001. doi:10.1145/369133.369152.

7 Mathilde Bouvel, Vladimir Grebinski, and Gregory Kucherov. Combinatorial search on graphs motivated by bioinformatics applications: A brief survey. In Dieter Kratsch, editor, *Graph-Theoretic Concepts in Computer Science, 31st International Workshop, WG 2005, Metz, France, June 23-25, 2005, Revised Selected Papers*, volume 3787 of *Lecture Notes in Computer Science*, pages 16–27. Springer, 2005. doi:10.1007/11604686_2.

8 Nader H. Bshouty. Linear time constructions of some d -restriction problems. In Vangelis Th. Paschos and Peter Widmayer, editors, *Algorithms and Complexity - 9th International Conference, CIAC 2015, Paris, France, May 20-22, 2015. Proceedings*, volume 9079 of *Lecture Notes in Computer Science*, pages 74–88. Springer, 2015. doi:10.1007/978-3-319-18173-8_5.

9 Nader H. Bshouty. Derandomizing chernoff bound with union bound with an application to k-wise independent sets. *CoRR*, abs/1608.01568, 2016. arXiv:1608.01568.

10 Nader H. Bshouty and Ariel Gabizon. Almost optimal cover-free families. In Dimitris Fotakis, Aris Pagourtzis, and Vangelis Th. Paschos, editors, *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, volume 10236 of *Lecture Notes in Computer Science*, pages 140–151, 2017. doi:10.1007/978-3-319-57586-5_13.

11 Hong-Bin Chen, Hung-Lin Fu, and Frank K. Hwang. An upper bound of the number of tests in pooling designs for the error-tolerant complex model. *Optimization Letters*, 2(3):425–431, 2008. doi:10.1007/s11590-007-0070-5.

12 Hong-Bin Chen and Frank K. Hwang. A survey on nonadaptive group testing algorithms through the angle of decoding. *J. Comb. Optim.*, 15(1):49–59, 2008. doi:10.1007/s10878-007-9083-3.

13 Dingzhu Du, Frank K Hwang, and Frank Hwang. *Combinatorial group testing and its applications*, volume 12. World Scientific, 2000.

14 Vladimir Grebinski and Gregory Kucherov. Reconstructing a hamiltonian cycle by querying the graph: Application to DNA physical mapping. *Discrete Applied Mathematics*, 88(1-3):147–165, 1998. doi:10.1016/S0166-218X(98)00070-5.

15 Hwang Frank Kwang-ming and Du Ding-zhu. *Pooling designs and nonadaptive group testing: important tools for DNA sequencing*, volume 18. World Scientific, 2006.

16 Weiwei Lang, Yuexuan Wang, James Yu, Suogang Gao, and Weili Wu. Error-tolerant trivial two-stage group testing for complexes using almost separable and almost disjunct matrices. *Discrete Math., Alg. and Appl.*, 1(2):235–252, 2009. doi:10.1142/S1793830909000191.

17 Anthony J. Macula and Leonard J. Popyack. A group testing method for finding patterns in data. *Discrete Applied Mathematics*, 144(1-2):149–157, 2004. doi:10.1016/j.dam.2003.07.009.

18 Douglas R. Stinson and Ruizhong Wei. Generalized cover-free families. *Discrete Mathematics*, 279(1-3):463–477, 2004. doi:10.1016/S0012-365X(03)00287-5.

19 Douglas R. Stinson, Ruizhong Wei, and Lie Zhu. Some new bounds for cover-free families. *J. Comb. Theory, Ser. A*, 90(1):224–234, 2000. doi:10.1006/jcta.1999.3036.

20 David C Torney. Sets pooling designs. *Annals of Combinatorics*, 3(1):95–101, 1999.

From Expanders to Hitting Distributions and Simulation Theorems

Alexander Kozachinskiy[1]
National Research University Higher School of Economics, Moscow, Russia
Moscow, 3 Kochnovsky Proezd, Russia
akozachinskiy@hse.ru
https://orcid.org/0000-0002-9956-9023

Abstract

In this paper we explore hitting distributions, a notion that arose recently in the context of deterministic "query-to-communication" simulation theorems. We show that any expander in which any two distinct vertices have at most one common neighbor can be transformed into a gadget possessing good hitting distributions. We demonstrate that this result is applicable to affine plane expanders and to Lubotzky-Phillips-Sarnak construction of Ramanujan graphs . In particular, from affine plane expanders we extract a gadget achieving the best known trade-off between the arity of outer function and the size of gadget. More specifically, when this gadget has k bits on input, it admits a simulation theorem for all outer function of arity roughly $2^{k/2}$ or less (the same was also known for k-bit Inner Product). In addition we show that, unlike Inner Product, underlying hitting distributions in our new gadget are "polynomial-time listable" in the sense that their supports can be written down in time $2^{O(k)}$, i.e. in time polynomial in size of gadget's matrix.

We also obtain two results showing that with current technique no better trade-off between the arity of outer function and the size of gadget can be achieved. Namely, we observe that no gadget can have hitting distributions with significantly better parameters than Inner Product or our new affine plane gadget. We also show that Thickness Lemma, a place which causes restrictions on the arity of outer functions in proofs of simulation theorems, is unimprovable.

2012 ACM Subject Classification Theory of computation → Communication complexity

Keywords and phrases simulation theorems, hitting distributions, expanders

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.4

1 Introduction

Assume that we have a Boolean function $f : \{0,1\}^n \to \{0,1\}$ called *outer function* and a Boolean function $g : A \times B \to \{0,1\}$ called *gadget*. Consider a composed function $f \circ g : A^n \times B^n \to \{0,1\}$, defined as follows:

$$(f \circ g)((a_1, \ldots, a_n), (b_1, \ldots, b_n)) = f(g(a_1, b_1), \ldots, g(a_n, b_n)).$$

How can we deal with deterministic communication complexity of $f \circ g$, denoted below by $D^{cc}(f \circ g)$? Obviously, we have the following inequality:

$$D^{cc}(f \circ g) \leq D^{dt}(f) \cdot D^{cc}(g),$$

where $D^{dt}(f)$ stands for deterministic query complexity of f. Indeed, we can transform a decision tree for f making q queries into a protocol of communication cost $q \cdot D^{cc}(g)$ by

[1] Supported in part by RFBR grapnt 16-01-00362 and by the Russian Academic Excellence Project "5-100".

simulating each query to f with $D^{cc}(g)$ bits. It turns out that for some gadgets g and for all f of arity at most some function of g's size this simple protocol is essentially optimal. The first gadget for which this was proved is the Indexing Function

$$\mathsf{IND}_k : \{1, 2, \ldots, k\} \times \{0,1\}^k \to \{0,1\}, \qquad g(x, y) = y_x.$$

More specifically, in 2015 Göös et al. ([4]) proved that for all $n \leq 2^{k^{1/20}}$ and for all $f : \{0,1\}^n \to \{0,1\}$ it holds that

$$D^{cc}(f \circ \mathsf{IND}_k) = \Omega(D^{dt}(f) \log k). \tag{1}$$

Actually, instead of f we can have not only a Boolean function but any relation $R \subset \{0,1\}^n \times C$. The work of Göös et al. was a generalization of the theorem of Raz and McKenzie ([9]), who in 1997 established (1) for a certain class of outer relations, called DNF-Search problems.

Theorems of this kind, called usually *simulation theorems*, can be viewed as a new method of proving lower bounds in communication complexity. Namely, lower bound on communication complexity of a composed function reduces to lower bound on query complexity of an outer function, and usually it is much easier to deal with the latter. As was shown by Raz and McKenzie, this method turns out to be powerful enough to separate monotone NC-hierarchy. Moreover, as was discovered by Göös et al., this method can be quadratically better than the logarithm of the partition number, another classical lower bound method in deterministic communication complexity.

There are simulation theorems not only for deterministic communication and query complexities, but for other models too, see, e.g., [2, 5, 6, 3].

Note that input length of a gadget in (1) is even bigger than input length of an outer function. Göös et al. in [4] asked, whether it is possible to prove a simulation theorem for a gadget which input length is logarithmic in input length of an outer function. This question was answered positively by Chattopadhyay et al. ([1]) and independently by Wu et al. ([12]). Moreover, Chattopadhyay et al. significantly generalized the proof of Göös et al., having discovered a certain property of a gadget $g : A \times B \to \{0,1\}$ which can be used as a black-box to show new simulation theorems: once g satisfies this property, we have a simulation theorem for g. This property can be defined as follows. Let μ be probability distribution over rectangles $U \times V \subset A \times B$. Distribution μ is called (δ, h)-*hitting*, where $\delta \in (0,1)$ and h is a positive integer, if for every $X \subset A$ of size at least $2^{-h}|A|$ and for every $Y \subset B$ of size at least $2^{-h}|B|$ we have that

$$\Pr_{U \times V \sim \mu}[U \times V \cap X \times Y \neq \varnothing] \geq 1 - \delta.$$

It turns out that if for every $b \in \{0,1\}$ there is (δ, h)-hitting distribution over b-monochromatic rectangles of g , then there is a simulation theorem for g. The smaller δ and the bigger h, the better simulation theorem. More precisely, Chattopadhyay et al. proved the following theorem.

▶ **Theorem 1.** *Assume that $\varepsilon \in (0,1)$ and an integer h are such that $h \geq 6/\varepsilon$. Then the following holds. For every (possibly partial) Boolean function $g : A \times B \to \{0,1\}$ that has two $(\frac{1}{10}, h)$-hitting distribution, the one over 0-monochromatic rectangles and the other over 1-monochromatic rectangles, for every $n \leq 2^{h(1-\varepsilon)}$ and $f : \{0,1\}^n \to \{0,1\}$ it holds that*

$$D^{cc}(f \circ g^n) \geq \frac{\varepsilon h}{4} \cdot D^{dt}(f).$$

Further, they showed that Inner Product and Gap Hamming Distance gadgets on k bits have $(o(1), \Omega(k))$-hitting distributions for both kinds of monochromatic rectangles. More precisely, for every constant $\gamma > 0$ and for all large enough k they constructed $(o(1), (1/2 - \gamma)k)$-hitting distributions for k-bit Inner Product (denoted below by IP_k) . Due to Theorem 1 this yields the following simulation theorem for IP_k: for every constant $\gamma > 0$ and for all k large enough

$$D^{cc}(f \circ \mathsf{IP}_k) = \Omega(D^{dt}(f) \cdot k),$$

where f is any Boolean function depending on at most $2^{(1/2-\gamma)k}$ variables. Other gadgets studied until this work do not achieve the same trade-off between the size of outer functions and the size of gadget. Namely, for k-bit Gap Hamming Distance the lower bound $D^{cc}(f \circ \mathsf{GHD}) = \Omega(D^{dt}(f) \cdot k)$ is shown in [1] only for f depending on roughly $2^{0.45k}$ variables or less. For Indexing gadget, as we saw, this trade-off is exponentially worse.

We also touch upon the following question. Theorem 1 transforms a communication protocol for $f \circ g^n$ into decision tree for f (here n is the arity of f). It is an interesting open question whether resulting decision tree can be made efficient, provided that the initial protocol is efficient. More precisely, a decision tree should decide which variable to query in time polynomial in n (provided that messages of the initial protocol can be computed by players in time polynomial in n). Let us note here that we are mostly interested in the regime when n is exponential in k (the size of input to g) and hence in this regime the size of g's matrix is polynomial in n.

Unfortunately, known constructions does not provide this – resulting decision trees work in exponential time. Among other obstacles to resolve this issue there is a particular step in transformation which deals with hitting distributions. Namely, a tree constantly runs a subroutine which, having a family of subrectangles (not necessarily monochromatic) of g's matrix on input, outputs single monochromatic rectangle which intersects most of them. Namely, a subroutine samples rectangle at random from a hitting distribution, which with good probability gives us a correct answer. To derandomize this procedure instead of sampling a rectangle we can just try all the rectangles from the support of hitting distribution. If the support is of size $2^{O(k)}$ (i.e. polynomial in size of g's matrix) and, moreover, the support can be listed in time $2^{O(k)}$, we call a corresponding hitting distribution (or, more precisely, family of distributions) *polynomial-time listable*.

For example, hitting distribution from [1] for k-bit Gap Hamming Distance gadget are polynomial-time listable (roughly speaking, we just have to list all Hamming balls of a certain radius). At the same time, hitting distributions for k-bit Inner Product from [1] are not polynomial-time listable. Namely, their supports are of size $2^{\Omega(k^2)}$ (this number corresponds to the number of $k/2$-dimensional subspaces of $\mathbb{F}_2^k$). Though due to Chernoff bound it is possible to transform any $(0.1, h)$-hitting distribution into , say, $(0.2, h)$-hitting distribution with support size $2^{O(k)}$ (see Proposition 7 below), this does not give explicit construction.

1.1 Our results

We show how to transform any explicit expander satisfying one additional restriction into a gadget with polynomial-time listable hitting distributions. The transformation is as follows. Assume that we have a graph $G = (V, E)$ and a coloring $c : V \to \{0, 1\}$. For $v \in V$ let $\Gamma(v)$ denote the set of all $u \in V$ such that u and v are connected by en edge in G. Assume further that for any two distinct $u, v \in V$ it holds that $|\Gamma(u) \cap \Gamma(v)| \leq 1$. Then the following partial

function is well defined:

$$g(G,c) : V \times V \to \{0,1\},$$
$$g(G,c)(u,v) = \begin{cases} 1 & u \neq v \text{ and there is } w \in \Gamma(u) \cap \Gamma(v) \text{ s.t. } c(w) = 1, \\ 0 & u \neq v \text{ and there is } w \in \Gamma(u) \cap \Gamma(v) \text{ s.t. } c(w) = 0, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Call c balanced if each color is used at least $|V|/3$ times in c. It turns out that if G is a good expander and if c is balanced, then $g(G,c)$ possesses good hitting distributions:

▶ **Theorem 2.** *Assume that $G = (V,E)$ is a (m,d,γ)-spectral expander in which for any two distinct $u,v \in V$ it holds that $|\Gamma(u) \cap \Gamma(v)| \leq 1$ and $c : V \to \{0,1\}$ is a balanced coloring of G. Assume also that $m \geq 1/\gamma^2$. Then for any $b \in \{0,1\}$ there is a $\left(\frac{1}{10}, \lfloor 2\log_2(1/\gamma)\rfloor - 100\right)$-hitting distribution μ_b over b-monochromatic rectangles of $g(G,c)$. All the probabilities of μ_b are rational. Moreover, there is a deterministic Turing machine which, having b, G and c on input, in time $m^{O(1)}$ lists all the rectangles from the support of μ_b, together with probabilities μ_b assigns to them.*

Provided that G's adjacency matrix and c's truth table can be computed in time $m^{O(1)}$, from Theorem 2 we obtain polynomial-time listable family of hitting distributions.

In particular, we apply Theorem 2 to the following explicit family of expanders. If q is a power of prime, let AP_q denote a graph in which vertices are pairs of elements of $\mathbb{F}_q$ and in which $(a,b),(x,y) \in \mathbb{F}_q^2$ are connected by an edge if and only if $ax = b + y$. It is known that AP_q is a $(q^2, q, 1/\sqrt{q})$-spectral expander. It can be easily shown that for any two distinct vertices u,v of AP_q it holds that $|\Gamma(u) \cap \Gamma(v)| \leq 1$.

▶ **Corollary 3.** *Let q be a power of prime. Then in AP_q for any two distinct vertices u,v it holds that $|\Gamma(u) \cap \Gamma(v)| \leq 1$. Moreover, for all $n \leq 2^{\log_2 q - 200}$ and $f : \{0,1\}^n \to \{0,1\}$ the following holds: if c is a balanced coloring of AP_q, then*

$$D^{cc}(f \circ g(AP_q, c)) \geq \frac{\log_2(q/n) - 200}{4} \cdot D^{dt}(f)$$

(in $g(AP_q,c)$ each party receives $2\log_2 q$ bits).

We also give an example of a natural-looking gadget for which Corollary 3 implies a simulation theorem. Our gadget is the following one: Alice gets $a \in \mathbb{F}_{q^2}$ and Bob gets $b \in \mathbb{F}_{q^2}$. Here q is a power of an odd prime. Their goal is to output 1, if $a - b$ is a square in $\mathbb{F}_{q^2}$ (by that we mean that there is $c \in \mathbb{F}_{q^2}$ such that $a - b = c^2$), and 0 otherwise. Let us denote this gadget by SQR^q.

Since $\mathbb{F}_{q^2}$ is a linear space over $\mathbb{F}_q$, we can naturally identify inputs to SQR^q with $\mathbb{F}_q^2$, i.e SQR^q can be viewed as a function of the form $\mathsf{SQR}^q : \mathbb{F}_q^2 \times \mathbb{F}_q^2 \to \{0,1\}$.

▶ **Proposition 4.** *For all large enough q the following holds. If q is a power of an odd prime, then there exists a balanced covering c of AP_q such that $g(AP_q,c)$ is a sub-function of SQR^p, i.e. whenever $g(AP_q,c)(a,b)$ is defined, we have $g(AP_q,c)(a,b) = \mathsf{SQR}^q(a,b)$. A truth table of c can be computed in time $q^{O(1)}$.*

This Proposition implies a simulation theorem for SQR^q, with the same parameters as in Corollary 3 and with polynomial-time listable underlying hitting distributions.

Next we observe that any spectral expander "similar" to AP_q automatically satisfies restrictions of Theorem 2.

▶ **Proposition 5.** *Assume that $G = (V, E)$ is a (m, d, γ)-spectral expander and*

$$2d + 4 > d^2 \left(2\gamma^2 + \frac{4(1-\gamma^2)}{m}\right).$$

Then for any two distinct vertices $u, v \in V$ it holds that $|\Gamma(u) \cap \Gamma(v)| \leq 1$.

In particular, all $(m^2, m, 1/\sqrt{m})$-spectral expanders satisfy these restrictions. However, Proposition 5 is by no means a necessary condition. For example, Theorem 2 can be also applied Lubotzky-Phillips-Sarnak construction of Ramanujan graphs ([8]). More specifically, if p, q are unequal primes, $p, q \equiv 1 \pmod 4$ and p is a quadratic residue modulo q, the paper [8] constructs an explicit graph $X^{p,q}$ which, in particular, is a $(q(q^2-1)/2, p+1, 2\sqrt{p}/(p+1))$-spectral expander and in which the shortest cycle is of length at least $2\log_p q$. It can also be easily shown that provided $p < q^2$ there are no self-loops in $X^{p,q}$. Thus if $p < \sqrt{q}$, then any two distinct vertices of $X^{p,q}$ have at most one common neighbor, while inequality from Proposition 5 is false for $X^{p,q}$.

We then obtain some results related to the following question: what is the best possible trade-off between the arity of outer functions and the size of gadget in deterministic simulation theorems? Once again, consider SQR^q. Note that in SQR^q each party receives $k = 2\log_2 q$ bits. Corollary 3 lower bounds $D^{cc}(f \circ \mathsf{SQR}^q)$ whenever arity of f is at most $2^{k/2-O(1)}$. In this regime the lower bound is of the form $\Omega(D^{dt}(f))$ (compare it to the $O(k \cdot D^{dt}(f))$ upper bound). In turn, if the arity of f is at most $2^{(1/2-\Omega(1))k}$, the lower bound becomes tight, namely $\Omega(k \cdot D^{dt}(f))$. Thus SQR^q achieves the same trade-off between the arity of f and the size of a gadget as k-bit Inner Product (while underlying hitting distributions for SQR^q, unlike Inner Product, are polynomial-time listable).

Ramanujan graphs yield gadgets with much worse trade-off. Namely, if p is of order $\sqrt{q}$ and c is a balanced coloring of $X^{p,q}$, then $g(X^{p,q}, c)$ is a gadget on $k \approx 3\log_2 q$ bits which admits a simulation theorem for all outer functions of arity roughly $2^{\log_2 p} = 2^{k/6}$.

This raises the following question: for a given k what is the maximal h such that there is a gadget on k bits having two $(\frac{1}{10}, h)$-hitting distributions, the one over 0-monochromatic rectangles and the other over 1-monochromatic rectangles? Above discussion shows that h can be about $k/2$. In the following Proposition we observe that it is impossible to obtain any constant bigger than $1/2$ before k.

▶ **Proposition 6.** *For every $g : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}$ and for every integer $h \geq 1$ there exists $b \in \{0,1\}$ such that the following holds. For every probability distribution μ over b-monochromatic rectangles of g there are $X, Y \subset \{0,1\}^k$ of size at least 2^{k-h} such that*

$$\Pr_{R \sim \mu}[R \cap X \times Y \neq \varnothing] \leq 2^{k-2h+1}.$$

In addition we show the following simple proposition, studying the minimal possible support size of hitting distributions.

▶ **Proposition 7.** *For every $g : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}$ the following holds*

- *if there is $\left(\frac{1}{20}, h\right)$-hitting distribution over b-monochromatic rectangles of g for some $b \in \{0,1\}$, then there is $\left(\frac{1}{10}, h\right)$-hitting distribution over b-monochromatic rectangles of g which support is of size $2^{O(k)}$.*
- *Assume that for some $\delta < 1$ and $h \in \mathbb{N}$ there are two (δ, h)-hitting distributions μ_0, μ_1, where μ_b is over b-monochromatic rectangles of g. Then the support of μ_b is of size at least 2^h, for every $b \in \{0,1\}$.*

So it is impossible to improve a trade-off between the size of outer functions and the size of gadgets simply by improving hitting distributions. However, until now we only spoke about improving gadgets. What about outer functions? What causes a restriction on the arity of f in Theorem 1? It can be verified that the only place in which arity of f appears in the proof is so-called Thickness Lemma. Let us state this Lemma.

Assume that A is a finite set and X is a subset of A^n. Here n corresponds to the arity of f. Let $X_{[n]/\{i\}}$ denote the projection of X onto all the coordinates except the i-th one. Define the following auxiliary bipartite graph $G_i(X)$. Left side vertices of $G_i(X)$ are taken from A, right side vertices of $G_i(X)$ are taken from $X_{[n]/\{i\}}$. We connect $a \in A$ with $(x_1, \dots, x_{i-1}, x_{i+1}, \dots x_n) \in X_{[n]/\{i\}}$ if and only if

$$(x_1, \dots, x_{i-1}, a, x_{i+1}, \dots x_n) \in X.$$

Clearly, there are $|X|$ edges in $G_i(X)$.

Let $MinDeg_i(X)$ denote the minimal possible degree of a right side vertex of $G_i(X)$. Similarly, let $AvgDeg_i(X)$ denote the average degree of a right side vertex of $G_i(X)$. There are $|X|$ edges and $|X_{[n]/\{i\}}|$ right side vertices, hence it is naturally to define $AvgDeg_i(X)$ as

$$AvgDeg_i(X) = \frac{|X|}{|X_{[n]/\{i\}}|}.$$

Thickness Lemma relates this two measures. Namely, it states that if for every i *average* degree of $G_i(X)$ is big, then there is a large subset $X' \subset X$ such that for every i *minimal* degree of $G_i(X')$ is big. The precise bounds can be found in the following

▶ **Lemma 8** ([9]). *Consider any $\delta \in (0,1)$. Assume that for every $i \in \{1, 2, \dots, n\}$ we have that $AvgDeg_i(X) \geq d$. Then there is $X' \subset X$ of size at least $(1-\delta)|X|$ such that for every $i \in [n]$ it holds that $MinDeg_i(X') \geq \frac{\delta d}{n}$.*

One possible way to improve a trade-off between the arity of f and the size of gadget is to improve Thickness Lemma. For example, if we could replace $\frac{\delta d}{n}$ with $\frac{\delta d}{\sqrt{n}}$ in Lemma 8 , this would mean that k-bit Inner Product and k-bit SQR-gadget admit simulation theorems for all outer functions of arity roughly 2^k (rather than $2^{k/2}$).

However, such an improvement is impossible and the bounds given in Lemma 8 are near-optimal. Note that Thickness Lemma says nothing about whether there even exists a *non-empty* subset $X' \subset X$ such that for all $i \in [n]$ it holds that $MinDeg_i(X')$ is larger, say, by a constant than $\frac{d}{n}$. And indeed, we show that for some X there is no such X' at all. More precisely, we show the following

▶ **Theorem 9.** *For every $\varepsilon > 0$ and for all integer $n \geq 2, s \geq 1$ there exists $m \in \mathbb{N}$ and a non-empty set $X \subset \{0, 1, \dots, m-1\}^n$ such that*

- *for all $i \in [n]$ it holds that $AvgDeg_i(X) \geq s(n-\varepsilon)$;*
- *there is no non-empty $Y \subset X$ such that for all $i \in [n]$ it holds that $MinDeg_i(Y) \geq s+1$.*

1.2 Organization of the paper

The rest of the paper is organized as follows.

In Section 2 we give Preliminaries. In Section 3 we prove Theorem 2 and derive Corollary 3. In Section 4 we prove Proposition 4. In Section 5 we prove Theorem 9. In Section 6 we prove Proposition 5. In section 7 we prove proposition 6. The proof of Proposition 7 is omitted due to space constraints.

2 Preliminaries

2.1 Sets notations

Let $[n]$ be the set $\{1, 2, \ldots, n\}$.

Assume that A is a finite set, X is a subset of A^n and $S = \{i_1, \ldots, i_k\}$, where $i_1 < i_2 < \ldots < i_k$, is a subset of $[n]$. Let X_S denote the following set:

$$X_S = \{(x_{i_1}, \ldots, x_{i_k}) : (x_1, \ldots, x_n) \in X\} \subset A^{|S|}.$$

Given $X \subset A^n$ and $i \in [n]$, consider the following bipartite graph $G_i(X) = (A, X_{[n]\setminus\{i\}}, E)$, where

$$E = \{(x_i, (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)) : (x_1, \ldots, x_n) \in X\}.$$

Vertices of $G_i(X)$ which are from A will be called left vertices. Similarly, vertices of $G_i(X)$ which are from $X_{[n]\setminus\{i\}}$ will be called right vertices.

Define $MinDeg_i(X)$ as minimal d such that there is a right vertex of $G_i(X)$ with degree d. Define $AvgDeg_i(X) = |X|/|X_{[n]\setminus\{i\}}|$.

2.2 Communication and query complexity

For introduction in both query and communication complexities see, e.g., [7]. We will use the following notation.

For a Boolean function $f : \{0,1\}^n \to \{0,1\}$ let $D^{dt}(f)$ denote f's deterministic query complexity, i.e. minimal d such that there is a deterministic decision tree of depth d computing f. For a (possibly partial) Boolean function $g : A \times B \to \{0,1\}$, where A, B are some finite sets, let $D^{cc}(g)$ denote g's deterministic communication complexity, i.e. minimal d such that there is a deterministic communication protocol of depth d, computing g. Let us stress that in the case when g is partial by "deterministic communication protocol computes g" we mean only that a protocol outputs 0 on (a, b) whenever $g(a, b) = 0$ and outputs 1 on (a, b) whenever $g(a, b) = 1$; on inputs on which g is not defined the protocol may output anything.

If f, g are as above, let $f \circ g$ denote the following (possibly partial) function:

$$f \circ g : A^n \times B^n \to \{0,1\},$$
$$(f \circ g)((a_1, \ldots, a_n), (b_1, \ldots, b_n)) = f(g(a_1, b_1), \ldots, g(a_n, b_n)).$$

We can also measure $D^{cc}(f \circ g)$, deterministic communication complexity of $f \circ g$, assuming that Alice's input is $(a_1, \ldots, a_n) \in A^n$ and Bob's input is $(b_1, \ldots, b_n) \in B^n$.

2.3 Hitting distributions

Fix a (possibly partial) Boolean function $g : A \times B \to \{0,1\}$. A set $R \subset A \times B$ is called *rectangle* if there are $U \subset A, V \subset B$ such that $R = U \times V$. If $b \in \{0,1\}$, then we say that rectangle R is b-monochromatic for g if $g(a, b) = b$ whenever $(a, b) \in R$. We stress that if g is partial, then in the definition of b-monochromatic rectangle we require that g is everywhere defined on R.

Let δ be positive real and h be positive integer. A probability distribution μ over rectangles $R \subset A \times B$ is called (δ, h)-hitting if for all $X \subset A, Y \subset B$ such that $|X| \geq 2^{-h}|A|, |Y| \geq 2^{-h}|B|$ it holds that

$$\Pr_{R \sim \mu}[R \cap X \times Y \neq \varnothing] \geq 1 - \delta.$$

In this paper we are focused only on those μ such that there exists $b \in \{0,1\}$ for which all rectangles from the support of μ are b-monochromatic for g. In this case we simply say that μ is over b-monochromatic rectangles of g.

Let $g_t : \{0,1\}^{k_t} \times \{0,1\}^{k_t} \to \{0,1\}$ be family of gadgets and μ_t be family of probability distributions, where μ_t is over rectangles of g_t. We call μ_t polynomial-time listable if the following holds:

- the size of the support of μ_t is $2^{O(k_t)}$;
- all the probabilities of μ_t are rational;
- there is a deterministic Turing machine which, having k_t on input, in time $2^{O(k_t)}$ computes g_t's matrix and lists all the rectangles from the support of μ_t, together with probabilities μ_t assigns to them.

2.4 SQR-gadget

Consider a finite field of size q, denoted below by $\mathbb{F}_q$. We call $a \in \mathbb{F}_q$ a *square* if there is $b \in \mathbb{F}_q$ such that $a = b^2$ in $\mathbb{F}_q$. Let SQR^q denote the following Booelan function:

$$\mathsf{SQR}^q : \mathbb{F}_{q^2} \times \mathbb{F}_{q^2} \to \{0,1\}, \qquad \mathsf{SQR}^q(a,b) = \begin{cases} 1 & \text{if } a-b \text{ is a square in } \mathbb{F}_{q^2}, \\ 0 & \text{if } a-b \text{ is not a square in } \mathbb{F}_{q^2}. \end{cases}$$

2.5 Expanders

We consider undirected graphs which may have parallel edges and self-loops. We assume that a self-loop at vertex v contributes 1 to degree of v. A graph is called d-regular if each its vertex has degree d.

A coloring of a graph $G = (V,E)$ is a function $c : V \to \{0,1\}$. It is called balanced if $|V|/3 \le |c^{-1}(1)| \le 2|V|/3$. For any $A \subset V$ let $\Gamma(A)$ denote the set of all $v \in V$ such that there is $u \in A$ connected with v by an edge of G. If $v \in V$, define $\Gamma(v) = \Gamma(\{v\})$.

Fix graph $G = (V,E)$ and a coloring $c : V \to \{0,1\}$. Assume that for any two distinct $u, v \in V$ it holds that $|\Gamma(u) \cap \Gamma(v)| \le 1$. Then the following partial function is well defined:

$$g(G,c) : V \times V \to \{0,1\},$$
$$g(G,c)(u,v) = \begin{cases} 1 & u \neq v \text{ and there is } w \in \Gamma(u) \cap \Gamma(v) \text{ s.t. } c(w) = 1, \\ 0 & u \neq v \text{ and there is } w \in \Gamma(u) \cap \Gamma(v) \text{ s.t. } c(w) = 0, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Let M_G be an adjacency matrix of a d-regular graph $G = (V,E)$ with $|V| = m$. Note that d is an eigenvalue of M_G. A graph G is called (m,d,γ)-spectral expander if M_G satisfies the following conditions:

- multiplicity of an eigenvalue d is 1;
- absolute value of any other eigenvalue of M_G is at most γd.

▶ **Proposition 10** ([11], Theorem 4.6). *Assume that a graph $G = (V,E)$ is (m,d,γ)-spectral expander. Then for any $A \subset V$:*

$$\frac{|\Gamma(A)|}{|A|} \ge \frac{1}{\gamma^2 + (1-\gamma^2)\frac{|A|}{m}}.$$

Assume the q is a power of prime. Let AP_q denote the following graph. Vertices of AP_q are pairs of elements of $\mathbb{F}_q$ so that the number of vertices is q^2. We connect (x, y) with (a, b) by an edge if and only if $ax = b + y$ in $\mathbb{F}_q$. It is easy to see that AP_q is q-regular.

▶ **Proposition 11** ([10], Lemma 5.1). *AP_q is $(q^2, q, 1/\sqrt{q})$-spectral expander.*

2.6 k-wise independent hash functions

We will need the following

▶ **Proposition 12** ([11], Corollary 3.34). *For every $n, k \in \mathbb{N}$ there exists a polynomial-time computable function $\psi : \{0,1\}^{kn} \times \{0,1\}^n \to \{0,1\}$ such for all distinct $x_1, \ldots, x_k \in \{0,1\}^n$ and for all $b_1, \ldots, b_k \in \{0,1\}$ the following holds:*

$$\Pr[\psi(s, x_1) = b_1, \ldots, \psi(s, x_k) = b_k] = 2^{-k},$$

where the probability is over uniformly random $s \in \{0,1\}^{kn}$.

2.7 Some useful facts

We will use the following inequality involving binomial coefficients:

▶ **Lemma 13.** *For every k, m the following holds: if $k \leq m/2$, then $\binom{m-k}{k}/\binom{m}{k} \geq 1 - \frac{k^2}{m-k}$.*

Proof. Observe that:

$$\frac{\binom{m-k}{k}}{\binom{m}{k}} = \frac{m-k}{m} \cdot \frac{m-k-1}{m-1} \cdot \ldots \cdot \frac{m-2k+1}{m-k+1} \geq \left(\frac{m-2k}{m-k}\right)^k$$
$$= \left(1 - \frac{k}{m-k}\right)^k \geq 1 - \frac{k^2}{m-k}.$$

The first inequality here is due to the fact that for all positive i we have:

$$\frac{k}{m-k+i} \leq \frac{k}{m-k} \implies 1 - \frac{k}{m-k+i} \geq 1 - \frac{k}{m-k}$$
$$\implies \frac{m-2k+i}{m-k+i} \geq \frac{m-2k}{m-k}.$$

The second inequality here is Bernoulli's inequality. It is legal to apply this inequality because $k \leq m/2$ and hence $\frac{k}{m-k} \leq 1$. ◀

Note that $\mathbb{F}_{q^2}$ contains a subfield of size q. Namely, $\mathbb{F}_q = \{x \in \mathbb{F}_{q^2} : x^q = x\}$.

▶ **Lemma 14.** *Assume that q is a power of an odd prime. Let α be a primitive root of $\mathbb{F}_{q^2}$. Then the following holds:*

- *$0, \alpha^2, \alpha^4, \ldots, \alpha^{q^2-1}$ are the only squares in $\mathbb{F}_{q^2}$;*
- *all the elements of $\mathbb{F}_q$ are squares in $\mathbb{F}_{q^2}$.*

Proof. Let us prove the first statement of the lemma. Assume that $j \in \{1, 2, \ldots, q^2 - 1\}$ is an odd integer. We will show that α^j is not a square. Indeed, assume for contradiction there is a non-zero $y \in \mathbb{F}_{q^2}$ such that $\alpha^j = y^2$. Therefore for some integer i we have that $\alpha^{j-2i} = 1$. Since α is the primitive root of $\mathbb{F}_{q^2}$, this means that $j - 2i$ is divisible by $q^2 - 1$. But $j - 2i$ is odd and $q^2 - 1$ is even.

To show the second statement of the lemma assume that $x = \alpha^k$ is a non-zero root of $x^q = x$. Then we have that $\alpha^{k(q-1)} = 1$. Due to the same argument as above $k(q-1)$ is divisible by $q^2 - 1$. This implies that k is divisible by $q + 1$. Hence k is even and $x = \alpha^k$ is a square. ◀

3 Transforming Expanders into Gadgets

In this section we prove Theorem 2 and derive Corollary 3.

Proof of Theorem 2. Fix $b \in \{0,1\}$ and set $h = \lfloor 2\log_2(1/\gamma) \rfloor - 100$. Let us define a $(\frac{1}{10}, h)$-hitting distribution μ_b over b-monochromatic rectangles of $g(G,c)$. Take $v \in c^{-1}(b)$ uniformly at random. Split $\Gamma(v)$ into two disjoints subsets A, B randomly according to 10-wise independent hash function $\psi : \{0,1\}^{10\cdot\lceil\log_2 m\rceil} \times \{0,1\}^{\lceil\log_2 m\rceil} \to \{0,1\}$ from Proposition 12. Namely, take $s \in \{0,1\}^{10\lceil\log_2 m\rceil}$ uniformly at random. An element $u \in \Gamma(v)$ goes into A if $\psi(s,u) = 0$ and into B if $\psi(s,u) = 1$. By definition $A \times B$ is a b-monochromatic rectangle of $g(G,c)$. Indeed, any two distinct vertices from $\Gamma(v)$ have a common neighbor colored in b. It remains to show that for all $S, T \subset V$ of size at least $2^{-h}m$ with probability at least 0.9 we have that $A \times B \cap S \times T \neq \varnothing$. It is enough to show that $\Pr[A \cap S \neq \varnothing] \geq 0.96$ and $\Pr[B \cap T \neq \varnothing] \geq 0.96$. Let us show that the first inequality holds, the proof of the second inequality is exactly the same. Actually we will show that $\Pr[|\Gamma(v) \cap S| \geq 10] \geq 0.97$. This is enough for our purposes: conditioned on $[|\Gamma(v) \cap S| \geq 10]$ the probability that A is disjoint with S is at most 2^{-10} (due to proposition 12 this is the probability that $\psi(s,\cdot)$ sends 10 fixed points of $\Gamma(v)$ into B). Therefore $\Pr[A \cap S] \geq (1 - 2^{-10}) \Pr[|\Gamma(v) \cap S| \geq 10] \geq 0.999 \cdot 0.97 > 0.96$.

The size of S is at least $2^{100}\gamma^2 m$. Partition S into 10 disjoint subsets $S_1, \ldots, S_{10}$, each of size at least $2000\lfloor\gamma^2 m\rfloor$. Since $m \geq 1/\gamma^2$, we also have $|S_1|, \ldots, |S_{10}| \geq 1000\gamma^2 m$. If $|\Gamma(v) \cap S| < 10$, then $\Gamma(v)$ is disjoint with S_i for some $i \in [10]$. Hence

$$\Pr[|\Gamma(v) \cap S| < 10] \leq \sum_{i=1}^{10} \Pr[\Gamma(v) \cap S_i = \varnothing].$$

If we show for all $i \in [10]$ that $\Pr[\Gamma(v) \cap S_i = \varnothing] \leq 0.003$, we are done. Observe that $\Gamma(v)$ is disjoint with S_i if and only if $v \notin \Gamma(S_i)$. This implies that

$$\Pr[\Gamma(v) \cap S_i = \varnothing] = \frac{|c^{-1}(b) \setminus \Gamma(S_i)|}{|c^{-1}(b)|} \leq \frac{m - |\Gamma(S_i)|}{\frac{m}{3}}. \tag{2}$$

In the last inequality we use the fact that c is balanced. By Proposition 10 we get

$$|\Gamma(S_i)| \geq \frac{|S_i|}{\gamma^2 + \frac{|S_i|}{m}} \geq \frac{|S_i|}{\frac{|S_i|}{1000 \cdot m} + \frac{|S_i|}{m}} \geq \frac{1000 \cdot m}{1001} > 0.999m.$$

Here in the second inequality we use the fact that $|S_i| \geq 1000\gamma^2 m$. Due to (2) this means that $\Pr[\Gamma(v) \cap S_i = \varnothing] \leq 0.003$ and thus the proof that μ_b is $\left(\frac{1}{10}, h\right)$-hitting is finished.

Let us now show that μ_b can be "written down" in time $m^{O(1)}$ from G and c. First of all, note that $g(G,c)$ is a gadget on $k = \lceil\log_2 m\rceil$ bits. To specify a rectangle from a support of μ_b we need to specify a vertex of G and a "seed" s of length $10k$. This shows that the support of μ_b is of size $m^{O(1)} = 2^{O(k)}$. This observation also allows us to list all the rectangles from the support of μ_b in time $2^{O(k)}$ – just go through all vertices from $c^{-1}(b)$ and all seeds. Further, the μ_b-probability of $A \times B$ can be computed as follows:

$$\mu_b(A \times B) = \frac{|\{v \in V : \Gamma(v) = A \cup B\}|}{|c^{-1}(b)|} \cdot \frac{\left|\{s \in \{0,1\}^{10k} : \phi(s,\cdot) \text{ splits } A \cup B \text{ into } A \text{ and } B\}\right|}{2^{10k}}.$$

This probability is rational and can be computed in time $2^{O(k)}$, again by exhaustive search over all vertices and seeds. ◀

Now let us derive Corollary 3. Indeed, AP_q is $(q^2, q, 1/\sqrt{q})$-spectral expander by Proposition 11. Thus theorem 2, applied to AP_q, states that for any balanced coloring c of AP_q and for any $b \in \{0,1\}$ there exists $\left(\frac{1}{10}, \lfloor \log_2(q) \rfloor - 100\right)$-hitting distribution over b-monochromatic rectangles of $g(AP_q, c)$. Apply Theorem 1 to these hitting distributions with $\varepsilon = 1 - \log_2(n)/(\lfloor \log_2(q) \rfloor - 100)$.

We only need to check that in AP_q for any two distinct vertices u, v is holds that $|\Gamma(u) \cap \Gamma(v)| \leq 1$. Assume that (x, y) and (u, v) are distinct vertices of AP_q. Take any $(a, b) \in \Gamma((x, y)) \cap \Gamma((u, v))$. Then

$$\begin{pmatrix} x & -1 \\ u & -1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} y \\ v \end{pmatrix}. \tag{3}$$

If $x \neq u$, then $\det \begin{pmatrix} x & -1 \\ u & -1 \end{pmatrix} \neq 0$ and hence system (3) has exactly one solution. If $x = u$, then $y \neq v$ and system (3) has no solution. Therefore $|\Gamma((x, y)) \cap \Gamma((u, v))| \leq 1$.

4 SQR^q Gadget

In this section we prove Proposition 4.

Fix $w \in \mathbb{F}_{q^2}$ such that $\{1, w\}$ is a basis of $\mathbb{F}_{q^2}$ over $\mathbb{F}_q$. Consider the following coloring of AP_q: set $c((a, b)) = 1$ if and only if $1 + wa$ is a square in $\mathbb{F}_{q^2}$; clearly a truth table of such c can be computed in time $q^{O(1)}$. Note that $g(AP_q, c)((x, y), (u, v))$ is defined if and only if $(x, y), (u, v)$ are distinct and there is $(a, b) \in \Gamma((x, y)) \cap \Gamma((u, v))$. Let us show that for any such $(x, y), (u, v)$ it holds that

$$g(AP_q, c)((x, y), (u, v)) = c((a, b)) = \mathsf{SQR}^q(x + yw, u + vw). \tag{4}$$

Indeed, we have that $ax = b + y, au = b + v$. This means that $y - v = a(x - u)$. Moreover, due to distinctness of $(x, y), (u, v)$ we have that $x \neq u$. Further,

$$x + yw - (u + vw) = (x - u) + w(y - v) = (x - u)(1 + wa).$$

Note that $x - u$ is a non-zero element of $\mathbb{F}_q$. By the second item of Lemma 14 this implies that $x + yw - (u + vw)$ is a square if and only if $1 + wa$ is a square. Hence (4) is true for all $(x, y), (u, v)$ from the domain of $g(AP_q, c)$.

It remains to show that c is balanced. Take $(a, b, \lambda) \in \mathbb{F}_q \times \mathbb{F}_q \times (\mathbb{F}_q \setminus \{0\})$ uniformly at random. Note that $c((a, b)) = 1$ if and only if $1 + wa$ is a square. Thus $|c^{-1}(1)| = q^2 \Pr[1 + wa \text{ is a square}]$. Due to the second item of Lemma 14 we have that $1 + wa$ is a square if and only if $\lambda(1 + wa)$ is a square. Note that $\lambda(1 + wa) = \lambda + \lambda aw$ is distributed uniformly in $\{i + wj : i, j \in \mathbb{F}_q, i \neq 0\}$ (this is because for any λ_0 the distribution of λa given $\lambda = \lambda_0$ is uniform in $\mathbb{F}_q$). Due to the first item of Lemma 14 for all large enough q there are at least $0.4q^2$ squares and at least $0.4q^2$ non-squares in $\{i + wj : i, j \in \mathbb{F}_q, i \neq 0\}$. This means that $1/3 \leq \Pr[\lambda(1 + wa) \text{ is a square}] \leq 2/3$ for all large enough q. Hence $q^2/3 \leq |c^{-1}(1)| \leq 2q^2/3$ and c is balanced.

5 Unimprovability of Thickness Lemma

Consider any set $X \subset \{0, 1, \ldots, m-1\}^n$ and take any $i \in [n]$. Let us say that $x \in X$ is *i-unique in X* if there is no other $x' \in X$ such that

$$x_1 = x'_1, \ldots, x_{i-1} = x'_{i-1}, x_{i+1} = x'_{i+1}, \ldots, x_n = x'_n.$$

Call a set $X \subset \{0,1,\dots,m-1\}^n$ *reducible* if for all non-empty $Y \subset X$ there is $i \in [n]$ such that $MinDeg_i(Y) = 1$. Note that X is reducible if and only if for all non-empty $Y \subset X$ there is $y \in Y$ which is i-unique in Y for some $i \in [n]$.

▶ **Lemma 15.** *For every $\varepsilon > 0$ and for every $n \geq 2$ there exists $m > 0$ and a reducible set $X \subset \{0,1,\dots,m-1\}^n$ such that for all $i \in [n]$ it holds that $AvgDeg_i(X) \geq n - \varepsilon$.*

Proof. Take any $m > 0$. Consider the following sequence of sets $X_2, X_3, \dots$, where X_n is a subset of $\{0,1,\dots,m-1\}^n$:

$$X_2 = \{(j,j) : j \in \{0,1,\dots,m-1\}\} \cup \{(j,j+1) : j \in \{0,1,\dots,m-2\}\},$$

$$X_{\ell+1} = \{(x,j) : x \in X_\ell,\ j \in \{0,1,\dots,m-1\}\} \\ \cup \left\{(y,0) : y \in \{0,1,\dots,m-1\}^\ell / X_\ell\right\}.$$

We have the following relation between the size of $X_{\ell+1}$ and the size of X_ℓ:

$$|X_{\ell+1}| = m \cdot |X_\ell| + m^\ell - |X_\ell| = m \cdot (|X_\ell| - 1) + m^\ell.$$

Let us show by induction on n that $|X_n| \geq nm^{n-1} - n(1 + m + \ldots + m^{n-2})$. Indeed, for $n = 2$ this inequality is true: $|X_2| = 2m - 1 > 2m - 2$. Now, assume that for $n = \ell$ this inequality is proved, i.e. $|X_\ell| \geq \ell m^\ell - \ell(1 + m + \ldots + m^{\ell-2})$. Then

$$\begin{aligned} |X_{\ell+1}| &= m \cdot (|X_\ell| - 1) + m^\ell \\ &\geq m \cdot \left(\ell m^{\ell-1} - \ell(1 + m + \ldots + m^{\ell-2}) - 1\right) + m^\ell \\ &\geq (\ell+1) \cdot m^\ell - (\ell+1) \cdot (1 + m + \ldots + m^{\ell-1}). \end{aligned}$$

This means that for every n and $i \in [n]$ it holds that

$$AvgDeg_i(X_n) = \frac{|X_n|}{|(X_n)_{[n]/\{i\}}|} \geq \frac{nm^{n-1} - n(1 + m + \ldots + m^{n-2})}{m^{n-1}},$$

and the latter tends to n as $m \to \infty$. Thus to show the lemma it is sufficient to show that X_n is reducible. Once again, we will show it by induction on n.

Consider $n = 2$ and take any non-empty $Y \subset X_2$. Let $y \in Y$ be the smallest element of Y in lexicographical order. If $y = (j,j)$, then y is 1-unique in Y and hence $MinDeg_1(Y) = 1$. If $y = (j,j+1)$, then y is 2-unique in Y and hence $MinDeg_2(Y) = 1$.

Further, assume that for $n = \ell$ the statement is proved, i.e. X_ℓ is reducible. Consider any non-empty $Y \subset X_{\ell+1}$. Assume that Y intersects $\left\{(y,0) : y \in \{0,1,\dots,m-1\}^\ell / X_\ell\right\}$ and hence for some $y \notin X_\ell$ it holds that $(y,0) \in Y$. Then $MinDeg_{\ell+1}(Y) = 1$. Indeed, in this case $(y,0)$ is $(\ell+1)$-unique in Y, because if $(y,j) \in Y \subset X_{\ell+1}$ for some $j > 0$, then $y \in X_\ell$, contradiction.

Now assume that Y is a subset of $\{(x,j) : x \in X_\ell,\ j \in \{0,1,\dots,m-1\}\}$. Then for some $j \in \{0,1,\dots,m-1\}$ a set $Y' = \{x \in X_\ell : (x,j) \in Y\}$ is non-empty. Since by induction hypothesis X_ℓ is reducible, there is $y \in Y'$ which is i-unique in Y' for some $i \in [\ell]$. Let us show that (y,j) is i-unique in Y (this would mean that $MinDeg_i(Y) = 1$). Indeed, assume that there is $(y',j') \in Y$ which coincides with (y,j) on all the coordinates except the i^{th} one. Then $j = j'$ and $y' \in Y'$. Due to i-uniqueness of $y \in Y'$ we also have that $y = y'$. ◀

▶ **Definition 16.** Let s, m, n be positive integers and assume that X is a subset of $\{0,1,\dots,m-1\}^n$. Let $In(X,s) \subset \{0,1,\dots,sm-1\}^n$ denote the following set:

$$In(X,s) = \{(sx_1 + r_1, sx_2 + r_2, \dots, sx_n + r_n) : \\ (x_1,\dots,x_n) \in X,\ r_1,\dots,r_n \in \{0,1,\dots,s-1\}\}.$$

Observe that for every $(y_1, \ldots, y_n) \in In(X, s)$ there is exactly one $(x_1, \ldots, x_n) \in X$ such that for some $r_1, \ldots, r_n \in \{0, 1, \ldots, s-1\}$ it holds that

$$y_1 = sx_1 + r_1, \ldots, y_n = sx_n + r_n.$$

▶ **Lemma 17.** *For every $i \in \{1, 2, \ldots, n\}$ it holds that $AvgDeg_i(In(X, s)) = s \cdot AvgDeg_i(X)$.*

Proof. Lemma follows from the following two equalities:

$$|In(X, s)| = s^n \cdot |X|, \qquad |In(X, s)_{[n]/\{i\}}| = s^{n-1} \cdot |X_{[n]/\{i\}}|.$$ ◀

▶ **Lemma 18.** *Assume that $X \subset \{0, 1, \ldots, m-1\}^n$ is reducible. Then for all non-empty $Y \subset In(X, s)$ there is $i \in [n]$ such that $MinDeg_i(Y) \leq s$.*

Proof. Let Y' be the set of all $(x_1, \ldots, x_n) \in X$ for which there are

$$r_1, \ldots, r_n \in \{0, 1, \ldots, s-1\}$$

such that $(sx_1 + r_1, \ldots, sx_n + r_n) \in Y$. Clearly Y' is non-empty. Let $x' = (x'_1, \ldots, x'_n) \in Y'$ be a string which is i-unique in Y'. Let us show that $MinDeg_i(Y) \leq s$. By definition there are $r_1, \ldots, r_n \in \{0, 1, \ldots, s-1\}$ such that $(y'_1, \ldots, y'_n) = (sx'_1 + r_1, \ldots, sx'_n + r_n) \in Y$. It is easy to see that $(y'_1, \ldots, y'_{i-1}, y'_{i+1}, \ldots, y'_n) \in Y_{[n]/\{i\}}$ is connected with at most s left vertices of $G_i(Y)$. More precisely, the only possible neighbors of $(y'_1, \ldots, y'_{i-1}, y'_{i+1}, \ldots, y'_n)$ are

$$sx'_i, sx'_i + 1, \ldots, sx'_i + s - 1.$$

Indeed, otherwise there is $x_i \neq x'_i$ such that $(x'_1, \ldots, x'_{i-1}, x_i, x'_{i+1}, \ldots, x'_n) \in Y$. The latter contradicts the fact that x' is i-unique in Y. ◀

Proof of Theorem 9. Due to Lemma 15 there is a reducible $X' \subset \{0, 1, \ldots, m-1\}^n$ such that for every $i \in [n]$ we have $AvgDeg_i(X') \geq n - \varepsilon$. By Lemma 17, applied to $X = In(X', s)$ for every $i \in [n]$ we have: $AvgDeg_i(X) \geq s(n - \varepsilon)$. Finally, due to Lemma 18, for all non-empty $Y \subset X$ there is $i \in [n]$ such that $MinDeg_i(Y) \leq s$. ◀

6 Expanders Similar to AP_q

In this section we prove Proposition 5. Let us stress that this Proposition is just a slight improvement of Proposition 10 for sets of size 2. Proposition 10 itself is not strong enough to conclude that in all $(m^2, m, 1/\sqrt{m})$-spectral expanders any two distinct vertices have at most 1 common neighbor.

For $S \subset V$ let $\mathbb{I}_S \in \mathbb{R}^{|V|}$ denote characteristic vector of a set S. Assume for contradiction that there are distinct $u, v \in V$ such that $|\Gamma(u) \cap \Gamma(v)| \geq 2$. Then the size of $\Gamma(\{u, v\})$ is at most $2d - 2$. Assume that M is the adjacency matrix of G. Denote $w = \{u, v\}$. Let us show that

$$\|M\mathbb{I}_w\|^2 \leq d^2 \left(2\gamma^2 + \frac{4(1-\gamma^2)}{m}\right). \tag{5}$$

Indeed, observe that $\mathbb{I}_w = \frac{2}{m}\mathbb{I}_V + (\mathbb{I}_w - \frac{2}{m}\mathbb{I}_V)$ and $(\mathbb{I}_w - \frac{2}{m}\mathbb{I}_V)$ is perpendicular to $\mathbb{I}_V$. Since

G is a (m, d, γ)-spectral expander, this implies that

$$\begin{aligned}\|M\mathbb{I}_w\|^2 &= \|M\left(\frac{2}{m}\mathbb{I}_w\right)\|^2 + \|M\left(\mathbb{I}_w - \frac{2}{m}\mathbb{I}_V\right)\|^2 \leq \frac{4d^2}{m} + \gamma^2 d^2 \|\left(\mathbb{I}_w - \frac{2}{m}\mathbb{I}_V\right)\|^2 \\ &= \frac{4d^2}{m} + \gamma^2 d^2\left(2\left(1-\frac{2}{m}\right)^2 + (m-2)\frac{4}{m^2}\right) \\ &= \frac{4d^2}{m} + \gamma^2 d^2\left(2 - \frac{4}{m}\right) = d^2\left(2\gamma^2 + \frac{4(1-\gamma^2)}{m}\right),\end{aligned}$$

and thus (5) is proved.

To obtain a contradiction it is enough to show the following inequality

$$\|M\mathbb{I}_w\|^2 \geq 2d + 4. \tag{6}$$

Assume that there are $t \leq 2d-2$ non-zero coordinates in $M\mathbb{I}_w$. Let $\xi_1, \ldots, \xi_t$ be the values of these coordinates. Their sum is $2d$. We need to show that $\xi_1^2 + \ldots + \xi_t^2 \geq 2d+4$. Observe that $\xi_1 - 1, \ldots, \xi_t - 1$ are non-negative integers and their sum is $2d - t \geq 2$. Clearly this implies that $(\xi_1-1)^2 + \ldots + (\xi_t - 1)^2 \geq 2$. Indeed, otherwise the sum of $\xi_1 - 1, \ldots, \xi_t - 1$ is either 0 or 1. Hence

$$\xi_1^2 + \ldots + \xi_t^2 = (\xi_1 - 1)^2 + \ldots + (\xi_t - 1)^2 + 4d - t \geq 2 + 4d - t \geq 2d + 4.$$

7 Proof of Proposition 6

Denote $s = 2^{k-h}$. Assume that there is a 0-monochromatic rectangle $A \times B$ of g such that $|A| \geq s$ and $B \geq s$. Then clearly the proposition is true for $b = 1$ and $X = A, Y = B$.

Now assume that if $A \times B$ is a 0-monochromatic rectangle of g, then either $|A| < s$ or $B < s$. Take $\mathcal{X}, \mathcal{Y}$ independently and uniformly at random from the set of all s-element subsets of $\{0,1\}^k$. Fix any 0-monochromatic rectangle $A \times B$ of g. Let us show that $\mathcal{X} \times \mathcal{Y}$ intersects $A \times B$ with probability at most 2^{k-2h+1}. Indeed, assume WLOG that $|A| < s$. Then

$$\Pr[\mathcal{X} \times \mathcal{Y} \cap A \times B \neq \varnothing] \leq \Pr[\mathcal{X} \cap A \neq \varnothing] = 1 - \frac{\binom{2^k - |A|}{s}}{\binom{2^k}{s}} \leq 1 - \frac{\binom{2^k - s}{s}}{\binom{2^k}{s}}$$

Since $h \geq 1$, we have that $s \leq 2^k/2$. Applying Lemma 13 we obtain:

$$\Pr[\mathcal{X} \times \mathcal{Y} \cap A \times B \neq \varnothing] \leq \frac{s^2}{2^k - s} \leq \frac{s^2}{2^k/2} = 2^{k-2h+1}.$$

Due to the standard averaging argument this means that for any probability distribution μ over 0-monochromatic rectangles of g it is possible to fix $\mathcal{X} = X, \mathcal{Y} = Y$ in such a way that

$$\Pr_{R \sim \mu}[R \cap X \times Y \neq \varnothing] \leq 2^{k-2h+1}.$$

References

1. Arkadev Chattopadhyay, Michal Kouckỳ, Bruno Loff, and Sagnik Mukhopadhyay. Simulation theorems via pseudorandom properties. *arXiv preprint arXiv:1704.06807*, 2017.
2. Susanna F de Rezende, Jakob Nordström, and Marc Vinyals. How limited interaction hinders real communication (and what it means for proof and circuit complexity). In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 295–304. IEEE, 2016.
3. Mika Goos, Shachar Lovett, Raghu Meka, Thomas Watson, and David Zuckerman. Rectangles are nonnegative juntas. *SIAM Journal on Computing*, 45(5):1835–1869, 2016.
4. Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 1077–1088. IEEE, 2015.
5. Mika Göös, Toniann Pitassi, and Thomas Watson. Query-to-communication lifting for bpp. *arXiv preprint arXiv:1703.07666*, 2017.
6. Hamed Hatami, Kaave Hosseini, and Shachar Lovett. Structure of protocols for xor functions. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 282–288. IEEE, 2016.
7. Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 2006.
8. Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.
9. Ran Raz and Pierre McKenzie. Separation of the monotone nc hierarchy. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 234–243. IEEE, 1997.
10. Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of mathematics*, pages 157–187, 2002.
11. Salil P Vadhan et al. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1–3):1–336, 2012.
12. Xiaodi Wu, Penghui Yao, and Henry Yuen. Raz-mckenzie simulation with the inner product gadget. In *Electronic Colloquium on Computational Complexity (ECCC)*, 2017.

Balance Problems for Integer Circuits

Titus Dose
Institute of Computer Science, Julius-Maximilians Universität Würzburg, Germany
titus.dose@uni-wuerzburg.de

Abstract

We investigate the computational complexity of *balance problems* for $\{-, \cdot\}$-circuits computing finite sets of natural numbers. These problems naturally build on problems for integer expressions and integer circuits studied by Stockmeyer and Meyer (1973), McKenzie and Wagner (2007), and Glaßer et al. (2010).

Our work shows that the balance problem for $\{-, \cdot\}$-circuits is undecidable which is the first natural problem for integer circuits or related constraint satisfaction problems that admits only one arithmetic operation and is proven to be undecidable.

Starting from this result we precisely characterize the complexity of balance problems for proper subsets of $\{-, \cdot\}$. These problems turn out to be complete for one of the classes L, NL, and NP.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness, Theory of computation → Computability

Keywords and phrases computational complexity, integer expressions, integer circuits

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.5

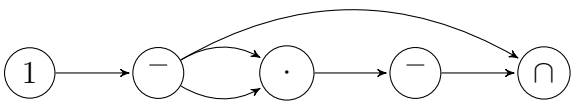
Related Version For a more comprehensive presentation of the results we refer to the technical report [6], `https://eccc.weizmann.ac.il/report/2018/055`.

1 Introduction

In 1973, Stockmeyer and Meyer [18] defined and studied membership and equivalence problems for *integer expressions*. They considered expressions built up from single natural numbers by using set operations ($\cup$, $\cap$, $\overline{\ }$), pairwise addition ($+$), and pairwise multiplication ($\cdot$). For example, $\overline{\overline{1} \cdot \overline{1}} \cap \overline{1}$ describes the set of primes $\mathbb{P}$.

The *membership problem for integer expressions* asks whether some given number is contained in the set described by a given integer expression, whereas the *equivalence problem for integer expressions* asks whether two given integer expression describe the same set. Restricting the set of allowed operations results in problems of different complexities.

Wagner [20] studied a more succinct way to represent such expressions, namely *circuits over sets of natural numbers*, also called integer circuits. Each input gate of such a circuit is labeled with a natural number, the inner gates compute set operations and arithmetic operations ($\cup$, $\cap$, $\overline{\ }$, $+$, $\cdot$). The following circuit with only 4 inner gates computes the set of primes.



Starting from this circuit, one can use integer circuits to express fundamental number theoretic questions: thus, a circuit describing the set of all twin primes or the set of all

Sophie Germain primes can be constructed. McKenzie and Wagner [15] constructed a circuit C computing a set that contains 0 if and only if the Goldbach conjecture holds.

Wagner [20], Yang [21], and McKenzie and Wagner [15] investigated the complexity of membership problems for circuits over natural numbers: here, for a given circuit C, one has to decide whether a given number n belongs to the set described by C. Travers [19] and Breunig [2] considered membership problems for circuits over integers and positive integers, respectively. Glaßer et al. [9] studied *equivalence problems for circuits over sets of natural numbers*, i.e., the problem of deciding whether two given circuits compute the same set.

Satisfiability problems for circuits over sets of natural numbers, investigated by Glaßer et al. [11], are a generalization of the membership problems investigated by McKenzie and Wagner [15]: the circuits can have *unassigned input gates* and the question is: on input of a circuit C with gate labels from $\mathcal{O} \subseteq \{\cup, \cap, \bar{}, +, \cdot\}$ and a natural number b, does there exist an assignment of the unassigned input gates with natural numbers such that b is contained in the set described by the circuit?

Barth et al. [1] investigated emptiness problems for integer circuits. Here, for both circuits with unassigned inputs and circuits without unassigned inputs, the question of whether an integer circuit computes the empty set (for some/all assignment(s) if the circuits allow unassigned inputs) is raised and investigated.

Apart from the mentioned research on circuit problems there has been work on related variants like functions computed by circuits [17] and constraint satisfaction problems (csp) over natural numbers [10, 5]. The constraint satisfaction problems by Glaßer, Jonsson, and Martin [10] can be considered as conjunctions of equations of integer expressions with variables standing for singleton sets of natural numbers. Here the question is whether there is an assignment of the variables such that all equations are satisfied. These constraint satisfaction problems have the peculiarity that expressions describe sets of integers whereas variables can only store singleton sets of natural numbers. Dose [5] addressed this and studied constraint satisfaction problems over finite subsets of $\mathbb{N}$, consequently replaced the set complement $\bar{}$ with the set difference $-$, and allowed the variables to describe arbitrary finite subsets of $\mathbb{N}$.

Our Model and Contributions

The definition of the circuits investigated in this paper follows the definition of previous papers such as [15, 9, 11, 1]. Yet there are some differences:

Our circuit problems are about *balanced sets* where a finite and non-empty set $S \subseteq \mathbb{N}$ is balanced if $|S| = |\{0, 1, \ldots, \max(S)\} - S|$. Analogously, S is unbalanced if $|S| \neq |\{0, 1, \ldots, \max(S)\} - S|$. That means, the maximum of a set marks the relevant area and then we ask whether there are as many elements inside the set as outside of it. As the notion of balanced sets only makes sense for finite sets, our circuits should solely compute finite sets. Due to that we replace the commonly used set complement $\bar{}$ with the set difference $-$. Now, as the circuits only work over the domain of finite subsets of $\mathbb{N}$, it suggests itself to also allow the input gates of a circuit to compute arbitrary finite subsets of $\mathbb{N}$ and not only singleton sets (cf. Dose [5] where the analogous step was made for constraint satisfaction problems).

For such circuits we ask: is there an assignment of the unassigned inputs with arbitrary finite subsets of $\mathbb{N}$ under which the circuit computes a balanced set? This problem is denoted by $\mathrm{BC}(\mathcal{O})$, where $\mathcal{O} \subseteq \{\cup, \cap, -, +, \cdot\}$ is the set of allowed operations.

The notion of balance is important in computational complexity. It occurs when considering counting classes [12] like $\mathrm{C_{=}L}$ or $\mathrm{C_{=}P}$ for instance. There, the question is whether for some problem A there is a non-deterministic logarithmic space or polynomial-time machine

M accepting A, where M accepts some input x if and only if the number of accepting paths equals the number of rejecting paths.

Balance problems for integer circuits are interesting for another reason. To our knowledge, there is no natural decision problem for integer circuits or constraint satisfaction problems over sets of natural numbers that allows only one arithmetic operation and is known to be undecidable. In this paper, however, it is shown that $\mathrm{BC}(-,\cdot)$ is undecidable.

Starting from this undecidable problem $\mathrm{BC}(-,\cdot)$, we also investigate $\mathrm{BC}(\mathcal{O})$ for arbitrary proper subsets of $\{-,\cdot\}$ and precisely characterize the complexity of each such problem. It turns out that all these problems are in NP. In detail, we show that $\mathrm{BC}(\cdot)$ is NL-complete, $\mathrm{BC}(-)$ is NP-complete, and $\mathrm{BC}(\emptyset) \in \mathrm{L}$.

2 Preliminaries

Basic Notions

Let $\mathbb{N}$ denote the set of natural numbers. $\mathbb{N}^+ = \mathbb{N} - \{0\}$ is the set of positive naturals. Moreover, the set of primes is denoted by $\mathbb{P}$.

We extend the arithmetical operations $+$ and $\cdot$ to sets of naturals: for $A, B \subseteq \mathbb{N}$ define $A + B = \{a + b \mid a \in A, b \in B\}$ and $A \cdot B = \{a \cdot b \mid a \in A, b \in B\}$. In contrast to previous papers, in this paper the multiplication of sets is not denoted by $\times$ but by $\cdot$. Instead, $\times$ denotes the cartesian product. Furthermore, for arbitrary sets, the operations $\cup$, $\cap$, and $-$ define the union, intersection, and set difference, respectively. The power set of a set M is denoted by $\mathcal{P}(M)$ whereas $\mathcal{P}_{\mathrm{fin}}(M) = \{A \in \mathcal{P}(M) \mid A \text{ finite}\}$. For a finite and non-empty set S let $\max(S)$ (resp., $\min(S)$) denote the maximum (resp., minimum) number of S. Finite intervals $\{x \mid a \leq x \leq b\}$ for $a, b \in \mathbb{Z}$ are denoted by $[a, b]$.

L, NL, and NP denote standard complexity classes [16] and RE is the set of computably enumerable problems.

For problems A and B we say that A is (logarithmic-space) many-one reducible to B if there is some (logarithmic-space) computable function f with $c_A(x) = c_B(f(x))$, where c_X for a set X is the characteristic function of X. We denote this by $A \leq_{\mathrm{m}} B$ (resp., $A \leq_{\mathrm{m}}^{\log} B$).

For pairs (A, B) and (C, D) with $A \cap B = C \cap D = \emptyset$ we say that (A, B) is many-one reducible to (C, D) (denoted as $(A, B) \leq_{\mathrm{m}} (C, D)$) if there is a computable function f with $x \in A \Rightarrow f(x) \in C$ and $x \in B \Rightarrow f(x) \in D$. Note that if $B = \overline{A}$ and $D = \overline{C}$ this coincides with the usual many-one reducibility, i.e., $(A, \overline{A}) \leq_{\mathrm{m}} (C, \overline{C}) \Leftrightarrow A \leq_{\mathrm{m}} C$.

CSAT is the circuit satisfiability problem, i.e., the problem of determining whether a given Boolean circuit has an assignment of the unassigned inputs that makes the output gate true. The problem is $\leq_{\mathrm{m}}^{\log}$-complete for NP via a trivial reduction from SAT which itself can be shown to be $\leq_{\mathrm{m}}^{\log}$-complete for NP via a construction by Cook [3].

Balanced Sets

A finite and non-empty set $S \subseteq \mathbb{N}$ is *balanced* (resp., *unbalanced*) if $|S| = |\{0, 1, \ldots, \max(S)\} - S|$ (resp., $|S| \neq |\{0, 1, \ldots, \max(S)\} - S|$). Intuitively spoken, $\max(S)$ defines the universe $\{0, 1, \ldots, \max(S)\}$ and then S is balanced if it contains the same number of elements as its complement. Note that the notion of balance/unbalance only makes sense if there is some maximum element defining the universe. Hence the empty set is neither balanced nor unbalanced.

The following lemma immediately follows from the definition.

▶ **Lemma 1.** *Let $S \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ be balanced. Then $S \neq \emptyset$ and $\max(S)$ is odd.*

Moreover, we say that S is *subbalanced* if $|S| < (\max(S)+1)/2$ which is equivalent to $|S| \leq \max(S)/2$. As we want to investigate the complexity of balance problems with respect to deterministic logarithmic-space reductions, it is important to see that the test of whether some input set is balanced can be done in deterministic logarithmic space. Define $\mathrm{Bal} = \{S \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N}) \mid S \text{ is balanced}\}$ and the slightly more general problem $\mathrm{Bal}_M = \{S \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N}) \mid M \cdot S \text{ is balanced}\}$ for a non-empty and finite set M. Standard arguments yield the following proposition.

▶ **Proposition 2.** $\mathrm{Bal}_M \in \mathrm{L}$. *In particular,* $\mathrm{Bal} \in \mathrm{L}$.

Circuits and Balance Problems for Circuits

In previous papers such as [1] it was differentiated between completely and partially assigned circuits. As we restrict on partially assigned circuits in this paper, we define circuits in general as partially assigned circuits.

A *circuit* C is a triple (V, E, g_C) where (V, E) is a finite, non-empty, directed, acyclic graph with a designated vertex $g_C \in V$ and a topologically ordered vertex set $V \subseteq \mathbb{N}$, i.e., if $u, v \in V$ are vertices with $u < v$, then there is no edge from v to u. Here, graphs may contain multi-edges and are not necessarily connected. But we require that C is topologically ordered. Note that the test of whether a graph is topologically ordered or not is possible in deterministic logarithmic space. Consequently, we are able to check in deterministic logarithmic space whether an input graph is acyclic. Hence there is a deterministic logarithmic-space algorithm that on input of a graph tests whether the input is a circuit. Therefore, when presenting algorithms for circuits we may always assume that the input is a valid circuit.

Let $\mathcal{O} \subseteq \{\cup, \cap, -, +, \cdot\}$. An *$\mathcal{O}$-circuit* (or *circuit* for short if $\mathcal{O}$ is apparent from the context) is a quintuple $C = (V, E, g_C, \alpha, \beta)$ where (V, E, g_C) is a circuit whose nodes are labeled by the *labeling function* $\alpha : V \to \mathcal{O} \cup \mathcal{P}_{\mathrm{fin}}(\mathbb{N}) \cup \{\square\}$ such that each node has indegree 0 or 2, nodes with indegree 0 have a label from $\mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ (encoded as a list of all the numbers in the set) or from $\{\square\}$, and nodes with indegree 2 have labels from $\mathcal{O}$. Moreover, β is a function $E \to \{\mathrm{l}, \mathrm{r}\}$ and we require that for each node u with predecessors u_1 and u_2 it holds $\{\beta(u_1), \beta(u_2)\} = \{\mathrm{l}, \mathrm{r}\}$. Thus, β marks whether an edge starts in the left or right predecessor of the node it points to.

In the context of circuits, nodes are also called gates. A gate with indegree 0 is called *input gate*, all other nodes are *inner gates*, the designated gate g_C is also called *output gate*. Input gates with a label from $\mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ are *assigned input gates* whereas input gates with label $\square$ are *unassigned input gates*.

$\mathcal{O}$-circuits are also called *integer circuits*. If g is some gate of C with $\alpha(g) = \otimes \in \mathcal{O}$ and with predecessors g' and g'' satisfying $\beta(g') = \mathrm{l}$ and $\beta(g'') = \mathrm{r}$, then we also write $g = g' \otimes g''$.

For an $\mathcal{O}$-circuit C with unassigned input gates $g_1 < \cdots < g_n$ and $X_1, \ldots, X_n \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N})$, let $C(X_1, \ldots, X_n)$ be the circuit that arises from C by modifying the labeling function α such that $\alpha(g_i) = X_i$ for every $1 \leq i \leq n$.

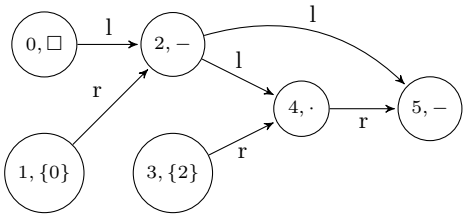
For a circuit $C = (V, E, g_C, \alpha)$ without unassigned input gates we inductively define the set $I(g; C)$ computed by a gate $g \in V$ for $g = 1, \ldots, |V|$ by

$$I(g; C) = \begin{cases} \alpha(g) \subseteq \mathbb{N} & \text{if } g \text{ has indegree 0,} \\ I(g', C) \otimes I(g'', C) & \text{if } g = g' \otimes g''. \end{cases}$$

The set computed by the circuit is denoted by $I(C)$ and defined to be the set computed by the output gate $I(g_C; C)$.

It is convenient to introduce notations for basic constructions of circuits. For $X \in \mathcal{P}_{\text{fin}}(\mathbb{N})$ we use X as an abbreviation for the circuit $(\{1\}, \varnothing, \{1\}, 1 \mapsto X)$. For $\mathcal{O}$-circuits C, C' for some $\mathcal{O}$ and $\otimes \in \{\cup, \cap, , -+, \cdot\}$ let $C \otimes C'$ be the circuit obtained from C' and C'' by feeding their output gates to the new output gate $\otimes$. This is possible in logarithmic space.

As an example, for an unassigned input gate $g = 0$, consider the circuit $C = (g - \{0\}) - ((g - \{0\}) \cdot \{2\})$, which is the following circuit



where each node is given by its number and its label. The node 5 is the output gate and it computes the set $\{1\}$ if and only if $I(2; C)$ is a set of the form $\{2^0, 2^1, 2^2, \ldots, 2^r\}$ for $r \in \mathbb{N}$.

Now we define the problems this paper focuses on.

▶ **Definition 3.** Let $\mathcal{O} \subseteq \{-, \cup, \cap, +, \cdot\}$ and define

$$\text{BC}(\mathcal{O}) = \{C \mid C \text{ is an } \mathcal{O}\text{-circuit with } n \text{ unassigned inputs and there exist } X_1, \ldots, X_n \in \mathcal{P}_{\text{fin}}(\mathbb{N}) \text{ such that } I(C(X_1, \ldots, X_n)) \text{ is balanced}\}.$$

We use the following abbreviations if confusions are impossible: we write g or $I(g)$ for $I(g; C)$, where C is a circuit and g is a gate of C; we write C for $I(C)$, where C is a circuit; we write $\text{BC}(-, \cdot)$ for $\text{BC}(\{-, \cdot\})$ and the like.

3 Set Difference and Multiplication Lead to Undecidability

This section contains our main result: the undecidability of $\text{BC}(-, \cdot)$ which is achieved by a reduction from a famously known RE-complete problem. According to the Matiyasevich-Robinson-Davis-Putnam theorem [14, 4] the problem of determining whether there is a solution for a given Diophantine equation is RE-complete. It can be derived by standard arguments that also the following problem is RE-complete (with regard to $\leq_{\text{m}}$).

$$\text{DE} = \{(p(x_1, \ldots, x_n), q(x_1, \ldots, x_n)) \mid \exists a_1, \ldots, a_n \in \mathbb{N}^+,\ p(a_1, \ldots, a_n) = q(a_1, \ldots, a_n) \text{ for multivariate polynomials } p \text{ and } q \text{ with coefficients from } \mathbb{N}^+\}.$$

▶ **Theorem 4.** $\text{BC}(-, \cdot)$ *is* RE*-complete.*

Let for the remainder of this section $\mathcal{O} = \{-, \cdot\}$ unless stated differently. For the sake of brevity, we make use of intersection gates but note that $A \cap B$ is just an abbreviation for $A - (A - B)$. Further abbreviated notations are $A - \bigcup_{i=1}^{n} B_i$ for $(\ldots((A - B_1) - B_2) - \ldots) - B_n$ and $A - (\bigcup_{i=1}^{n} B_i - \{1\})$ for $(\ldots((A - (B_1 - \{1\})) - (B_2 - \{1\})) - \ldots - (B_n - \{1\})$.

In order to prove Theorem 4 we define a slightly different version of the problem $\text{BC}(-, \cdot)$ which can be reduced to the original version in logarithmic space.

▶ **Definition 5.** Define

$$\text{BC}'(\mathcal{O}) = \{(C, Q) \mid C \text{ is a partially assigned } \mathcal{O}\text{-circuit, } Q \text{ is a subset of the nodes of } C, \text{ and there exist } X_1, \ldots, X_n \in \mathcal{P}_{\text{fin}}(\mathbb{N}^+) \text{ such that } I(C(X_1, \ldots, X_n)) \text{ is balanced and } I(K; C(X_1, \ldots, X_n)) = \{1\} \text{ for all } K \in Q\}.$$

For the sake of simplicity, we call instances of $\text{BC}'(\mathcal{O})$ $\mathcal{O}$-circuits as well.

▶ **Lemma 6.** *1. For $K \in \mathcal{P}_{\text{fin}}(\mathbb{N})$ with $\kappa := \max(K) \geq 3$ it holds $|K \cdot K \cdot K| < \kappa^3/2$.*
2. $\text{BC}'(\mathcal{O}) \leq_{\text{m}}^{\log} \text{BC}(\mathcal{O})$ for $\mathcal{O} = \{-, \cdot\}$.

Proof sketch. We only sketch the proof of statement 2. Let C be a partially assigned $\mathcal{O}$-circuit with output node g_C and let Q be a subset of the nodes of C. Starting with this circuit, we build a new circuit and denote this modified circuit by C':

For each assigned or unassigned input node g, add a node g' of type $-$ which computes the set $g - \{0\}$, replace all edges (g, h) with (g', h), and in case $g \in Q$, remove g from Q and add g'. Then add a new output node $g_{C'} = g_C \cdot \prod_{K \in Q}(K \cdot K \cdot K)$.

Making use of Statement 1, it can be proved that $\text{BC}'(\mathcal{O}) \leq_{\text{m}}^{\log} \text{BC}(\mathcal{O})$ via $C \mapsto C'$. ◀

Before proving Theorem 4 we introduce some $\mathcal{O}$-circuits which will be used extensively as components of circuits expressing Diophantine equations.

▶ **Lemma 7.** *For every finite $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{P}$ with $n = |P| \geq 1$ there is an $\mathcal{O}$-circuit (C_P, Q_P) containing gates $g_P^1, \ldots, g_P^n$ satisfying the following properties:*

1. *For an arbitrary assignment with values from $\mathcal{P}_{\text{fin}}(\mathbb{N}^+)$ it holds:*
   if $K = \{1\}$ for all $K \in Q_P$, then $\exists_{m \in \mathbb{N}} \, \forall_{i=1,\ldots,n} \, g_P^i = \{1, p_i, \ldots, p_i^m\}$.
2. *For each $m \in \mathbb{N}$ there is an assignment with values from $\mathcal{P}_{\text{fin}}(\mathbb{N}^+)$ under which $g_P^i = \{1, p_i, \ldots, p_i^m\}$ and $K = \{1\}$ for all $K \in Q_P$.*

Proof sketch. Construct the $\mathcal{O}$-circuit (C_P, Q_P) as follows.

- For each $p \in P$ insert an input gate X_p and gates $h_p = X_p - (X_p \cdot \{p\})$ and $h'_p = (\{1, p\} \cdot X_p) - (X_p - \{1\})$. Put all the nodes h_p into Q_P.
- Similarly, for $k \in \{p_1 \cdot p_2, p_2 \cdot p_3, \ldots, p_{n-1} \cdot p_n\}$ insert an input gate X_k and gates $h_k = X_k - (X_k \cdot \{k\})$ and $h'_k = (\{1, k\} \cdot X_k) - (X_k - \{1\})$. Insert all nodes h_k into Q_P.
- For each $k = p_i \cdot p_{i+1}$ with $i \in \{1, \ldots, n-1\}$ add a node $\gamma_k = h'_k - \big((h'_{p_i} \cdot h'_{p_{i+1}}) - \{1\}\big)$ and let Q_P contain all these nodes.
- Denote $g_P^i = X_{p_i}$.

It can be shown that (C_P, Q_P) satisfies the requirements of the lemma. ◀

By adding nodes of the form $\prod_{j=1}^{i} g_P^j$ for some i we receive the following.

▶ **Lemma 8.** *For every finite $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{P}$ with $n = |P| \geq 1$ there is an $\mathcal{O}$-circuit (D_P, Q_P) with gates $g_P^0, g_P^1, \ldots, g_P^n$ satisfying the following properties:*

1. *For an arbitrary assignment with values from $\mathcal{P}_{\text{fin}}(\mathbb{N}^+)$ it holds*

$$\forall_{K \in Q_P} \, K = \{1\} \quad \Rightarrow \quad \exists_{m \in \mathbb{N}^+} \forall_{i=0,\ldots,n} \, |g_P^i| = m^i, \; 1 \in g_P^i, \text{ and the prime divisors of numbers in } g_P^i \text{ are all in } P.$$

2. *For each $m \in \mathbb{N}^+$ there is an assignment with values from $\mathcal{P}_{\text{fin}}(\mathbb{N}^+)$ under which $|g_P^i| = m^i$ and $1 \in g_P^i$ for all i, the prime divisors of numbers in g_P^i are all in P, and $K = \{1\}$ for all $K \in Q_P$.*

Proof of Theorem 4. Due to Lemma 6 it suffices to show $\text{DE} \leq_{\text{m}} \text{BC}'(-, \cdot)$. Instead of showing this reduction directly we define an intermediate problem, the cardinality circuit problem CC given by

$\{(C, Q, s, t) \mid C = (V, E, g_C, \alpha, \beta)$ is a $\{-, \cdot\}$-circuit, $Q \subseteq V$, $s, t \in V$, and there exists an assignment with values from $\mathcal{P}_{\text{fin}}(\mathbb{N}^+)$ under which

1. $|I(s)| = |I(t)|$
2. $1 \in I(s) \cap I(t)$
3. $I(K) = \{1\}$ for all $K \in Q$
4. $I(s)$ and $I(t)$ only contain numbers whose prime divisors are all $> 3\}$.

Moreover, define

$$\mathcal{C} = \{(C,Q,s,t) \mid C = (V,E,g_C,\alpha,\beta) \text{ is a } \{-,\cdot\}\text{-circuit, } Q \subseteq V,\ s,t \in V, \text{ for all assignments with values from } \mathcal{P}_{\text{fin}}(\mathbb{N}^+) \text{ satisfying } \forall_{K\in Q}\ K = \{1\} \text{ it holds that } s \geq t \text{ and } s \text{ and } t \text{ only contain numbers whose prime divisors are } > 3\},$$

i.e., for all circuits in $\mathcal{C}$ each relevant assignment maps s to a set with higher or equal cardinality than the set it maps t to and each relevant assignment maps s and t to sets that do not contain any numbers with prime divisors ≤ 3. For the sake of simplicity, we also call tuples (C,Q,s,t) $\{-,\cdot\}$-circuits.

The proof will be given in the two steps

1. $(\text{DE}, \overline{\text{DE}}) \leq_{\text{m}} (\text{CC}, \overline{\text{CC}} \cap \mathcal{C})$
2. $(\text{CC}, \overline{\text{CC}} \cap \mathcal{C}) \leq_{\text{m}} (\text{BC}'(-,\cdot), \overline{\text{BC}'(-,\cdot)})$.

Thus the function composition of the two reduction functions shows $\text{DE} \leq_{\text{m}} \text{BC}'(-,\cdot)$.

1. Roughly speaking, the first of the two reductions generates a circuit computing two sets whose cardinalities express the results of two multivariate polynomials.

Let q and q' be multivariate polynomials with variables $x_1,\dots,x_n$. Then for any assignment with positive natural numbers $a_1,\dots,a_n$ it holds $q(a_1,\dots,a_n) = q'(a_1,\dots,a_n)$ if and only if $q(a_1,\dots,a_n)^2 + q'(a_1,\dots,a_n)^2 = 2\cdot q(a_1,\dots,a_n)\cdot q'(a_1,\dots,a_n)$. Observe that here because of $(q(a_1,\dots,a_n) - q'(a_1,\dots,a_n))^2 \geq 0$ we have $q(a_1,\dots,a_n)^2 + q'(a_1,\dots,a_n)^2 \geq 2\cdot q(a_1,\dots,a_n)\cdot q'(a_1,\dots,a_n)$ for any assignment. Due to that we may assume that we are given multivariate polynomials q and q' with variables $x_1,\dots,x_n$ such that $q \geq q'$ for all assignments of the variables with values from $\mathbb{N}^+$. Let

$$q = \sum_{i=1}^{m} a_i \cdot \prod_{j=1}^{n} x_j^{d_{i,j}} \quad \text{and} \quad q' = \sum_{i=1}^{m'} a'_i \prod_{j=1}^{n} x_j^{d'_{i,j}}$$

for positive numbers m, m', a_i, and a'_i and natural numbers $d_{i,j}$ and $d'_{i,j}$. Moreover, for each variable x_j define $e_j = \max(\{d_{1,j},\dots,d_{m,j},d'_{1,j},\dots,d'_{m',j}\})$, i.e., e_j denotes the maximum exponent of the variable x_j occurring in a monomial of q or q'.

We now successively build the output circuit (C,Q,s,t). For the single steps we give intuition which is written italic.

1. For each variable x_j select a set $P_j = \{p_{j,1},\dots,p_{j,e_j}\}$ of primes greater than 3 such that $|P_j| = e_j$ and $P_j \cap P_{j'} = \emptyset$ for $j \neq j'$. Then insert a circuit (C_{P_j}, Q_{P_j}) according to Lemma 8 and for all P_j, insert the nodes of Q_{P_j} into Q.
   We will make use of the notation of Lemma 8, in particular of the nodes $g^0_{P_j},\dots,g^{e_j}_{P_j}$. That means, for any assignment which satisfies $K = \{1\}$ for all $K \in Q \supseteq Q_{P_j}$, it holds $|g^i_{P_j}| = m^i_j$ for $m_j \in \mathbb{N}^+$ and for all $i \leq e_j$. Moreover, in that case all primes dividing some number of $g^i_{P_j}$ are in P_j.
   For intuition, think of the node $g^i_{P_j}$ as a set whose cardinality describes x^i_j.
2. **a.** Choose a prime $p > 3$ not used before and insert gates $h_i = \{1,p,\dots,p^{a_i-1}\}\cdot\prod_{j=1}^{n} g^{d_{i,j}}_{P_j}$ for all $i = 1,\dots,m$.
   Loosely speaking, the cardinality of h_i describes the value of the i-th monomial of q.
   b. For each node h_i choose a prime $p_i > 3$ not used before and insert a node $h'_i = (\{1,p_i\}\cdot h_i) - \big(h_i - \{1\}\big)$.
   As addition is supposed to be simulated by union, we need to make sure that the sets standing for distinct monomials are disjoint. Still, for a technical reason we have to keep 1 in each set. So the idea is to let h'_i consist of 1 and a copy of h_i multiplied with an additional prime factor.

   c. For $i = 1, \dots, m$ add an unassigned input node z_q. Finally add nodes $z_q - \left(\bigcup_{i=1}^{m} h'_i - \{1\}\right)$ and $h'_i - (z_q - \{1\})$ (for $i = 1, \dots, m$) and insert these nodes into Q.
   Roughly speaking, z_q describes the value of $q + 1$ as it is the union of all the h'_i.
3. Do the same as in step 2 but for q'. In particular a node $z_{q'}$ is added.
4. Define $s = z_q$ and $t = z_{q'}$.

First, observe that the function $(q, q') \mapsto (C, Q, s, t)$ is computable. In order to show

$$(q, q') \in \mathrm{DE} \Rightarrow (C, Q, s, t) \in \mathrm{CC} \quad \text{and} \quad (q, q') \notin \mathrm{DE} \Rightarrow (C, Q, s, t) \in \overline{\mathrm{CC}} \cap \mathcal{C}$$

we make the following central observation.

▶ **Claim 9.**

1. *For each $y_1, \dots, y_n \in \mathbb{N}^+$ there is an assignment of the circuit (C, Q) with values from $\mathcal{P}_{\mathrm{fin}}(\mathbb{N}^+)$ such that s (resp., t) consists of $1 + q(y_1, \dots, y_n)$ (resp., $1 + q'(y_1 \dots, y_n)$) numbers whose prime divisors are greater than 3, $1 \in s \cap t$, and $K = \{1\}$ for all $K \in Q$.*
2. *If $K = \{1\}$ for all $K \in Q$ under some assignment with values from $\mathcal{P}_{\mathrm{fin}}(\mathbb{N}^+)$, then there are $y_1, \dots, y_n \in \mathbb{N}^+$ such that $|s| = 1 + q(y_1, \dots, y_n)$ and $|t| = 1 + q'(y_1, \dots, y_n)$ and s and t solely contain numbers whose prime divisors are all greater than 3.*

Proof of Claim 9.

1. Let $y_1, \dots, y_n \in \mathbb{N}^+$. Then according to Lemma 8 the inputs of the circuits (C_{P_j}, Q_{P_j}) can be chosen such that
   - $K = \{1\}$ for all $K \in Q_{P_j}$,
   - $|g^i_{P_j}| = y^i_j$ and $1 \in g^i_{P_j}$ for $i = 1, \dots, e_j$, and
   - all prime divisors of numbers in $g^i_{P_j}$ are in P_j and greater than 3.

   As the set of primes chosen for two different variables are disjoint and in step 2b we select primes not used before, the gate h_i associated with the monomial $a_i \cdot \prod_{j=1}^{n} x_j^{d_{i,j}}$ contains $a_i \cdot \prod_{j=1}^{n} y_j^{d_{i,j}}$ elements that only have prime divisors greater than 3. Furthermore, as $1 \in h_i$ for all i, we have $|h'_i| = 2 \cdot |h_i| - (|h_i| - 1) = |h_i| + 1$. Moreover, observe that $h'_i \cap h'_j = \{1\}$ for arbitrary $i \neq j$.
   For the node z_q choose the assignment $\bigcup_{i=1}^{m} h'_i$. Consequently, $1 \in z_q$ and

$$|z_q| = 1 + \sum_{i=1}^{m} \underbrace{(|h'_i| - 1)}_{=|h_i|} = 1 + \sum_{i=1}^{m} a_i \cdot \prod_{j=1}^{n} x_j^{d_{i,j}} = 1 + q(y_1, \dots, y_n).$$

   Since we do the same for the nodes associated with the polynomial q' we have $|z_{q'}| = 1 + q'(y_1, \dots, y_n)$ and $1 \in z_{q'}$. Observe that the prime divisors of numbers in z_q and $z_{q'}$ are greater than 3.
   It remains to observe that all nodes added into Q in step 2c compute the set $\{1\}$. This holds since z_q was chosen to be $\bigcup_{i=1}^{m} h'_i$.
2. Consider an assignment with $K = \{1\}$ for all $K \in Q$. Then according to Lemma 8 for each variable x_j we have $|g^i_{P_j}| = y^i_j$ for some $y_j \in \mathbb{N}^+$ and $i = 0, \dots, e_j$ and all numbers in these gates solely have prime divisors in P_j. As the P_j are pairwise disjoint and in step 2b we select primes not used before, we obtain $|h_i| = a_i \cdot \prod_{j=1}^{n} y_j^{d_{i,j}}$ and $|h'_i| = |h_i| + 1$. As $h'_i \cap h'_j = \{1\}$ for $i \neq j$ and each h'_i contains 1, it holds $|z_q| = 1 + \sum_{i=1}^{n} a_i \cdot \prod_{j=1}^{n} y_j^{d_{i,j}} = 1 + q(y_1, \dots, y_n)$. Similarly we obtain $|z_{q'}| = 1 + q'(y_1, \dots, y_n)$.
   It remains to argue that under the given assignment s and t do not contain any numbers with prime divisors ≤ 3. Obviously, the assigned inputs only compute sets whose elements solely have prime divisors greater than 3. By our construction and Lemma 8 the same

holds for all nodes $g^i_{P_j}$. As a consequence, all nodes h_i and h'_i have the same property and due to $z_q - \left(\bigcup_{i=1}^m h'_i - \{1\}\right) = \{1\}$ (cf. Step 2c) this also holds for $z_q = s$. An analogous argumentation shows that also t does not contain any numbers with prime divisors ≤ 3. ◀

As it has been argued above that $|q| \geq |q'|$, Claim 9 implies the following two statements: If $(q, q') \in \mathrm{DE}$, then $(C, Q, s, t) \in \mathrm{CC}$. If $(q, q') \notin \mathrm{DE}$, then $(C, Q, s, t) \in \overline{\mathrm{CC}} \cap \mathcal{C}$.

2. Now we show $(\mathrm{CC}, \overline{\mathrm{CC}} \cap \mathcal{C}) \leq_{\mathrm{m}} (\mathrm{BC}'(-,\cdot), \overline{\mathrm{BC}'(-,\cdot)})$. The following algorithm computes the reduction function. The italic comments are supposed to give some intuition.

1. Let a circuit (C, Q, s, t) be given. We construct a circuit (C', Q') by successively updating the given circuit.
2. Add new unassigned input gates X and X'. Insert the following nodes into Q':

$$\{1,2\} \cdot s - (X - \{1\}), \tag{1}$$
$$\{1,2\} \cdot t - (X - \{1\}), \tag{2}$$
$$\{1,2\} \cdot (X - s) - \big((X' \cup (X - s)) - \{1\}\big), \tag{3}$$
$$X' - \{2\} \cdot (X - s). \tag{4}$$

The basic idea is as follows: X is supposed to be an interval containing s and t and X' basically encodes the set $X - s$ where this set is made disjoint to t by multiplying it with $\{2\}$. As $|s| \geq |t|$, the set $X' \cup t$ is subbalanced. But if $|s| = |t|$, then $X' \cup t$ is almost balanced. Adding the element $\max(X')+1$ would make the set balanced. This element is generated in the next step.

3. Let $p_1 = 2$ and $p_2 = 3$. Add a circuit $(C_{\{p_1,p_2\}}, Q_{\{p_1,p_2\}})$ according to Lemma 7. Put all nodes of $Q_{\{p_1,p_2\}}$ into Q'. Add a node $g = \big(g^2_{\{p_1,p_2\}} \cdot \{1,3\}\big) - (g^2_{\{p_1,p_2\}} - \{1\})$.
4. Add a new unassigned input node O and the following nodes which are also added to Q':

$$O - \Big(\big(X' \cup t \cup g\big) - \{1\}\Big), \tag{5}$$
$$X' - (O - \{1\}), \tag{6}$$
$$t - (O - \{1\}), \tag{7}$$
$$g - (O - \{1\}). \tag{8}$$

Thus, roughly speaking, the output set O equals $X' \cup t \cup g$ and is only balanced if $|t| \geq |s|$.

5. Let O be the output node of the circuit (C', Q').

▶ **Claim 10.** *If $(C, Q, s, t) \in \mathrm{CC}$, then $(C', Q') \in \mathrm{BC}'(-,\cdot)$.*

Proof of Claim 10. Let $(C, Q, s, t) \in \mathrm{CC}$. Then there is some assignment with
- $|s| = |t|$,
- $1 \in s \cap t$,
- $K = \{1\}$ for all $K \in Q$, and
- s and t only contain numbers whose prime divisors are all greater than 3.

We now consider the circuit (C', Q') under an assignment satisfying the four conditions just mentioned. Moreover, we choose the input of $C_{\{p_1,p_2\}}$, X, X', and O such that
- $g = \{1, 3^m\}$ for m minimal with $4 \cdot (\max(s \cup t) + 1) < 3^m$ and $4 \mid 3^m - 1$ and all nodes in $Q_{\{p_1,p_2\}}$ compute $\{1\}$ (such an assignment exists by Lemma 7),
- $X = \{x \mid 1 \leq x \leq (3^m - 1)/2\}$,
- $X' = \{1\} \cup \{2\} \cdot (X - s)$, and
- $O = X' \cup t \cup g = \big(\big(\{2\} \cdot (X - s)\big)\big) \cup t \cup \{3^m\}$.

In order to see $K = \{1\}$ for all $K \in Q'$ it remains to consider the nodes added in the steps 2 and 4. Due to the choice of g and X it holds $\max(X) > 2{\cdot}\max(s \cup t)$ and thus the nodes defined in (1) and (2) compute $\{1\}$. The choice of X' immediately implies that the node defined in (4) computes $\{1\}$. Now we argue for the node defined in (3): As $X' = \{1\} \cup \{2\} \cdot (X - s)$ we have $\{1,2\} \cdot (X - s) - \big((X' \cup (X - s)) - \{1\}\big) = \{1,2\} \cdot (X - s) - \big((\{1,2\} \cdot (X - s)) - \{1\}\big) = \{1\}$. The nodes defined in (5), (6), (7), and (8) compute $\{1\}$ by the choice of g, X, X', and O.

As s and t only contain numbers whose prime divisors are > 3, the sets $\{2\} \cdot (X - s)$, t, and $\{3^m\}$ are disjoint. Hence, $|O| = \max(X) - |s| + |t| + 1 = \max(X) + 1 = \frac{\max(O)-1}{2} + 1 = \frac{\max(O)+1}{2}$ and thus O is balanced. ◀

▶ **Claim 11.** *If* $(C, Q, s, t) \in \overline{\mathrm{CC}} \cap \mathcal{C}$*, then* $(C', Q') \in \overline{\mathrm{BC}'(-,\cdot)}$.

Proof of Claim 11. For a contradiction, assume that $(C, Q, s, t) \in \overline{\mathrm{CC}} \cap \mathcal{C}$ and $(C', Q') \in \mathrm{BC}'(-,\cdot)$. As the second circuit is an extended version of the first circuit, both circuits can now be considered under the same assignment. Choose an assignment with values from $\mathcal{P}_{\mathrm{fin}}(\mathbb{N}^+)$ under which O is balanced and all $K \in Q'$ satisfy $K = \{1\}$. As by construction $Q \subseteq Q'$, we have $K = \{1\}$ for $K \in Q$.

As in particular the nodes defined in (1) and (2) compute $\{1\}$, we obtain $1 \in s \cap t$, $X \supseteq \{1,2\} \cdot s \cup \{1,2\} \cdot t$, and in particular $s \subseteq X$ and $\max(X) > \max(s) \geq 1$. As $\{1,2\} \cdot (X - s) - \big((X' \cup (X - s)) - \{1\}\big) = \{1\}$ (cf. (3)), it holds $2 \cdot \max(X) \in X'$. Since the node defined in (4) computes $\{1\}$, we obtain $X' \subseteq \{1\} \cup \{2\} \cdot (X - s)$. In particular, $\max(X') = 2 \cdot \max(X)$.

The fact that the nodes defined in (5), (6), (7), and (8) compute $\{1\}$ implies $1 \in O \cap X' \cap t \cap g$ and $O = X' \cup t \cup g$. Moreover, it follows from Lemma 7 that $g = \{1, 3^m\}$ for some $m \in \mathbb{N}^+$. Thus, as $1 \in t$,

$$O \subseteq \{1\} \cup \big(\{2\} \cdot (X - s)\big) \cup t \cup g = \big(\{2\} \cdot (X - s)\big) \cup t \cup \{3^m\}. \tag{9}$$

As O is balanced, $\max(O)$ is odd by Lemma 1. Since $X \supseteq t$ and $\max(X') = 2 \cdot \max(X)$ is even, $\max(O) = 3^m > \max(X')$. Due to $(C, Q, s, t) \in \mathcal{C}$, under the given assignment $|s| \geq |t|$ and s and t do not contain any numbers with prime divisors ≤ 3. Due to that, since we have seen $1 \in s \cap t$, and as by assumption $(C, Q, s, t) \notin \mathrm{CC}$, it even holds $|s| > |t|$.

Putting things together, as we have proven (9), $|s| > |t|$, $1 \in t$, $s \subseteq X$, $\max(X') = 2 \cdot \max(X)$, and $\max(O) > \max(X')$, we now obtain $|O| \leq \max(X) - |s| + |t| + 1 < \max(X) + 1 = \frac{\max(X')+2}{2} \leq \frac{\max(O)+1}{2}$, which contradicts the fact that O is balanced. ◀

This completes the proof of $(\mathrm{CC}, \overline{\mathrm{CC}} \cap \mathcal{C}) \leq_{\mathrm{m}} (\mathrm{BC}'(-,\cdot), \overline{\mathrm{BC}'(-,\cdot)})$ and thus $\mathrm{BC}'(-,\cdot)$ and $\mathrm{BC}(-,\cdot)$ are $\leq_{\mathrm{m}}$-complete for RE. ◀

4 Smaller Sets of Operations Lead to Problems in NP

In this section it is shown that all problems $\mathrm{BC}(\mathcal{O})$ for $\mathcal{O} \subsetneq \{-,\cdot\}$ are in NP. Each of these problems is proven to be $\leq_{\mathrm{m}}^{\log}$-complete for one of the classes L, NL, and NP.

4.1 The Complexity of the Problem Solely Admitting Multiplication

This section's purpose is to argue for the NL-completeness of $\mathrm{BC}(\cdot)$. Special cases of strong results from the literature [7, 8, 13] essentially yield the following theorem.

▶ **Theorem 12.** *There exists* $\mu \in \mathbb{N}$ *such that for all non-empty sets* $A, B \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ *with* $\max(A) \geq \mu$ *and* $\max(B) \geq \mu$ *the set* $A \cdot B$ *is subbalanced.*

▶ **Theorem 13.** BC(·) *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* NL.

Proof. In the following we present an NL-algorithm for BC(·). We make use of the fact that the graph accessibility problem for directed graphs and the modifications of this problem

$$\mathrm{GAP}_{\geq k} = \{(G,s,t) \mid G \text{ is an directed graph, there exist } k \text{ paths from } s \text{ to } t\}$$

and consequently

$$\mathrm{GAP}_{=k} = \{(G,s,t) \mid G \text{ is an directed graph, the number of paths from } s \text{ to } t \text{ is } k\}$$

for $k \in \mathbb{N}^+$ are in NL. We may assume the following for the input circuit C:

1. All gates in C are connected to the output gate g_C. Otherwise, delete all edges not connected to the output, which can be done by an NL-subroutine.
2. No assigned input computes the empty set or the set $\{0\}$. Otherwise, under the assumption of 1 we may reject immediately.
3. There is an assigned input gate a computing a set with maximum ≥ 2. Otherwise: under the assumption of 1 and 2,
   - we may accept if there is an unassigned input or no assigned input computes $\{0,1\}$
   - we may reject if there does not exist an unassigned input and there is an assigned input computing $\{0,1\}$.
4. No assigned input gate but possibly a computes a set containing 0. Otherwise, under the assumptions 1 and 3 we may delete 0 from all assigned inputs and insert 0 into a.
5. There is an assigned input node g_1 computing $\{1\}$.
6. For each set $M \subseteq \mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ there is at most one assigned input computing M. Otherwise, select one of the nodes computing M, let all outgoing edges of nodes computing M start in this node, and delete all other nodes computing M and their incident edges.

Assume there is an NL-algorithm P that accepts the set of those circuits C which satisfy the mentioned properties and whose unassigned inputs can be assigned with sets of *positive* naturals such that the output set is balanced. Then the following NL-algorithm accepts BC(·) (on input of a circuit C satisfying the properties listed above).

- If P accepts on C, accept.
- If there is an unassigned input, then add 0 into the set computed by the aforementioned node a and accept if P accepts the modified circuit.
- Reject.

Now we sketch P and argue that it is an NL-algorithm. Let $\mu \geq 2$ be the number mentioned in Theorem 12, i.e., for $A, B \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ with $\max(A) \geq \mu \leq \max(B)$ the set $A \cdot B$ is subbalanced. The algorithm will query the following constant-size problem

$$\Theta = \{(B,k_1,k_2) \mid B \subseteq \{(h,i_h) \mid h \subseteq \{0,1,\ldots,\mu\}, 1 \leq i_h \leq 2\}, |B| \leq \mu, k_1 \leq \mu, k_2 \leq \mu,$$
$$\exists_{E_1,\ldots,E_{k_1},F_1,\ldots,F_{k_2} \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N}^+)} \prod_{(h,i_h)\in B} h^{i_h} \cdot \prod_{i=1}^{k_1} E_i \cdot \prod_{i=1}^{k_2} F_i^2 \text{ is balanced}\}.$$

1. If there are two assigned input gates each containing an element $\geq \mu$, reject.
   If there is an assigned input gate with two paths to g_C containing an element $\geq \mu$, reject.
2. If there are at least μ assigned input gates computing a set with maximum ≥ 2, reject.
3. In case there is an assigned input gate computing a set with maximum ≥ 2 with at least three paths to the output, reject.
4. Let $v_1,\ldots,v_n$ be the nodes of the circuit in topological order. For $i = 1,\ldots,n$, if one of the conditions

- v_i is an unassigned input with at least three paths to g_C.
- v_i is an unassigned input with precisely one path to g_C, such that there are at least μ unassigned inputs $< v_i$ with precisely one path to g_C.
- v_i is an unassigned input with precisely two paths to g_C, such that there are at least μ unassigned inputs $< v_i$ with precisely two paths to g_C.
- g_1 is the only input with a path to v_i.

is satisfied, then delete v_i and let all outgoing edges of v_i start in g_1.
This step can be implemented as a non-deterministic logarithmic-space subroutine.

5. Let n_1 (resp., n_2) be the number of unassigned inputs with 1 path (resp., 2 paths) to g_C. Due to Step 4 we have $\max(n_1, n_2) \leq \mu$. Moreover, let A be a set consisting of all pairs (h, i_h) where h is a set computed by an assigned input with $1 < \max(h) \leq \mu$ and $i_h \in \{1, 2\}$ is the number of paths from h to g_C. Due to Step 2 it holds $|A| \leq \mu$. We have the following cases.

a. In case there is no assigned input gate with an element $\geq \mu$:
If $(A, n_1, n_2) \in \Theta$, then accept. Otherwise reject.
Computing the triple (A, n_1, n_2) is possible in non-deterministic logarithmic space whereas the subsequent test only requires constant time.

b. In case there is one assigned input gate g with an element $\geq \mu$:
Due to Step 1 the node g only has one path to the output.

i. For all $E_1, \ldots, E_{n_1}, F_1, \ldots, F_{n_2} \in \mathcal{P}(\{1, \ldots, \mu\})$ do the following

- Compute the constant-size set

$$M = \prod_{i=1}^{n_1} E_i \cdot \prod_{i=1}^{n_2} F_i^2 \cdot \prod_{(h,i_h)\in A, h\neq g} h^{i_h}.$$

- Test whether $g \in \mathrm{Bal}_M$ and accept in case the answer is "yes".

ii. Reject.

By Proposition 2 this step can be executed in logarithmic space.

In the following we observe that each step of the algorithm P accepts (resp., rejects) if and only if the circuit at the beginning of the execution of the respective step has a (resp., no) balancing assignment with values from $\mathcal{P}_{\mathrm{fin}}(\mathbb{N}^+)$. It suffices to argue for the following steps.

1. If the algorithm rejects in this step, then there are sets A and B with $\max(A) \geq \mu \leq \max(B)$ and a set M such that $g_C = A \cdot B \cdot M$. Then according to Theorem 12 it holds $|(A \cup \{0\}) \cdot B| \leq \max(A) \cdot \max(B)/2$. Hence for each set $M \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ the set $M \cdot A \cdot B \subseteq (A \cup \{0\}) \cdot B \cdot (M - \{0\})$ contains at most $\max(A) \cdot \max(B) \cdot \max(M)/2$ elements and its greatest element is $\max(A) \cdot \max(B) \cdot \max(M)$. Thus, the set is subbalanced.

2. If there are $\geq \mu$ sets with maximum greater 2 connected to the output, then we can interpret these sets as two sets with maxima $\geq \mu$ and argue in the same way as in the step before.

3. If the algorithm rejects in this step, then there are sets A and M with $\max(A) \geq 2$ and $g_C = A \cdot A \cdot A \cdot M$. If $\max(A) = 2$, then Lemma 1 states that the output set is not balanced. Otherwise, $\max(A) \geq 3$ and according to Statement 2 of Lemma 6 the set $A \cdot A \cdot A$ contains less than $\max(A)^3/2$ elements. Hence g_C contains less than $\max(A)^3 \cdot \max(M)/2$ elements and the maximum of this set is $\max(A)^3 \cdot \max(M)$. Thus g_C is subbalanced.

5. At the beginning of the execution of this step we have the following situation: Due to the steps 1, 2, and 3 and because of the assumption we made on the input circuit there

- is at most one assigned input containing an element $> \mu$ and this has at most one path to the output gate.
- are at most μ assigned inputs with maximum ≥ 2 and all these inputs have at most two paths to the output gate.
- is one assigned input with maximum < 2, namely $g_1 = \{1\}$.

Moreover, as observed above, because of Step 4 it holds $\max(n_1, n_2) \leq \mu$ and there are no unassigned inputs with more than 2 paths to the output.

Thus we have to consider two cases. Either there is no assigned input with maximum $> \mu$ or there is one. In the first case the circuit has a balancing assignment with values from $\mathcal{P}_{\text{fin}}(\mathbb{N}^+)$ if and only if there are $n_1 + n_2$ sets $E_1, \ldots, E_{n_1}, F_1, \ldots, F_{n_2} \in \mathcal{P}_{\text{fin}}(\mathbb{N}^+)$ such that $\prod_{(h,i_h)\in B} h^{i_h} \cdot \prod_{i=1}^{k_1} E_i \cdot \prod_{i=1}^{k_2} F_i^2$ is balanced. This is what the algorithm tests. In the second case, assigning one of the unassigned inputs with a set with maximum $> \mu$ would lead to a subbalanced output with the same argument as was used for Step 1. Thus, only assignments with values from $\mathcal{P}(\{1, \ldots, \mu\})$ have to be considered. Hence, there is a balancing assignment with values from $\mathcal{P}_{\text{fin}}(\mathbb{N}^+)$ if and only if there are sets $E_1, \ldots, E_{n_1}, F_1, \ldots, F_{n_2} \in \mathcal{P}(\{1, \ldots, \mu\})$ such that $\prod_{i=1}^{n_1} E_i \cdot \prod_{i=1}^{n_2} F_i^2 \cdot \left(\prod_{(h,i_h)\in A, h\neq g} h^{i_h}\right) \cdot g$ is balanced. This is what the algorithm tests.

It remains to observe that the circuit has a balancing assignment with values from $\mathcal{P}_{\text{fin}}(\mathbb{N}^+)$ before the execution of Step 4 if and only if it has afterwards:
In case there are more than μ unassigned inputs with one path (resp., two paths) to the output and more than μ of them are mapped to sets containing elements ≥ 2, then the same arguments as for Step 2 yield that the output is subbalanced. Therefore, all but μ of these nodes can be replaced with g_1.
Let g be an unassigned input with at least three paths to the output (if such a node exists). Assigning this node with a set with maximum ≥ 2 leads to a subbalanced output set with the same arguments as were used for Step 3. Therefore, g can be replaced with g_1.
For each node v_i there exists an input that has a path to v_i. Hence, if no input different from g_1 has a path to v_i, then v_i computes $\{1\}$ and can be replaced with g_1.

By a straightforward reduction from a problem investigated by McKenzie and Wagner [15] one receives the NL-hardness of BC(·). ◀

4.2 The Complexity of the Problems Not Admitting Multiplication

We consider the two remaining problems and prove that BC(−) is $\leq_{\text{m}}^{\log}$-complete for NP and BC(∅) is in L. The NP-hardness of BC(−) can be obtained by a straightforward reduction from CSAT. Hence, for the following theorem it suffices to argue for the membership in NP.

▶ **Theorem 14.** BC(−) *is* $\leq_{\text{m}}^{\log}$*-complete for* NP.

Proof. We sketch an NP-algorithm that accepts BC(−).

1. Input: a circuit C with output node g_C and labeling function α.
2. Go from g_C upwards always taking the left predecessor. Denote the input gate finally reached by g.
3. If g is assigned, then: guess an assignment with values from $\mathcal{P}(\alpha(g))$ and accept if the output set is balanced for this assignment, otherwise reject.
4. Here g is unassigned. Let M be the union of all sets computed by assigned inputs. Let $m = \max(M) + 1$. Guess an assignment such that $I(g) = \{m\}$ and each unassigned input either computes $\{m\}$ or $\emptyset$. If under this assignment g_C contains m, then accept.

5. Guess an assignment of the unassigned inputs such that each of them computes a subset of M. In case g_C is balanced, accept. Otherwise reject.

If the algorithm accepts, then $C \in \mathbf{BC}(-)$: It suffices to consider the 4-th step. If the algorithm accepts in this step, then there is an assignment that maps each unassigned input either to $\{m\}$ or to $\emptyset$ such that m is in the output set. Now change this assignment such that the sets mapped to $\{m\}$ are now mapped to $\{m+1, m+2, \ldots, 2m+1\}$. Then $I(C) = \{m+1, m+2, \ldots, 2m+1\}$ is balanced and $C \in \mathrm{BC}(-)$. Trivially, in case C is accepted in the 5-th step, $C \in \mathrm{BC}(-)$.

If the algorithm rejects, then $C \notin \mathbf{BC}(-)$: If the algorithm rejects, then this happens in step 3 or step 5. We argue for the first case. Here g is an assigned input gate. As the output set is a subset of $\alpha(g)$, it holds $g_c \subseteq \alpha(g)$ for any assignment and hence it suffices to consider assignments that map all unassigned inputs to subsets of $\alpha(g)$. As the algorithm rejects, g_C is not balanced under any of these assignments and thus $C \notin \mathrm{BC}(-)$.

It remains to argue for the case where the algorithm rejects in step 5. In this case, g is an unassigned input and as step 4 did not accept, there is no assignment putting elements outside of M into the circuit's output set. Hence, it is sufficient to consider assignments that solely map to subsets of M. As the algorithm rejects, none of these assignments yields a balanced output set and hence there is no assignment at all under which the output set is balanced. Therefore, $C \notin \mathrm{BC}(-)$. ◀

The following theorem basically follows from Proposition 2.

▶ **Theorem 15.** $\mathrm{BC}(\emptyset) \in \mathrm{L}$.

5 Conclusion and Open Questions

The following table summarizes our results, namely the lower and upper complexity bounds for the complexity of $\mathrm{BC}(\mathcal{O})$ with $\mathcal{O} \subseteq \{-, \cdot\}$.

$\mathrm{BC}(\mathcal{O})$ for $\mathcal{O} =$	$\leq_{\mathrm{m}}^{\log}$-hard for	contained in
$\emptyset$	L	L, Theorem 15
$\{-\}$	NP, Theorem 14	NP, Theorem 14
$\{\cdot\}$	NL, Theorem 13	NL, Theorem 13
$\{\cdot, -\}$	undecidable, Theorem 4	

To our knowledge, in contrast to all results from previous papers on complexty issues concerning decision problems for integer circuits (e.g., [15, 19, 2, 9, 11, 1]) or related constraint satisfaction problems ([10, 5]), a problem *admitting only one arithmetic operation* is shown to be undecidable. Beginning with this problem, namely $\mathrm{BC}(-, \cdot)$, the problems $\mathrm{BC}(\mathcal{O})$ for $\mathcal{O} \subseteq \{-, \cdot\}$ are systematically investigated and for each of these problems the complexity is precisely characterized. It turns out that decreasing the size of the set of allowed operations yields problems that are in NP. In particular, all these problems are $\leq_{\mathrm{m}}^{\log}$-complete for one of the classes L, NL, and NP.

Hence, in some sense the questions of this paper are completely answered. Nevertheless, there arise new questions from our results: Is there a set $\mathcal{O} \subseteq \{-, \cup, \cap\}$ such that $\mathrm{BC}(\mathcal{O} \cup \{+\})$ is undecidable? And if so, for which of the sets this is the case and for which it is not?

References

1 D. Barth, M. Beck, T. Dose, C. Glaßer, L. Michler, and M. Technau. Emptiness problems for integer circuits. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, pages 33:1–33:14, 2017. doi:10.4230/LIPIcs.MFCS.2017.33.

2 H.-G. Breunig. The complexity of membership problems for circuits over sets of positive numbers. In *Fundamentals of Computation Theory, 16th International Symposium, FCT 2007, Budapest, Hungary, August 27-30, 2007, Proceedings*, pages 125–136, 2007. doi:10.1007/978-3-540-74240-1_12.

3 S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158, 1971. doi:10.1145/800157.805047.

4 M. Davis, H. Putnam, and J. Robinson. The decision problem for exponential Diophantine equations. *Annals of Mathematics*, 74(2):425–436, 1961.

5 T. Dose. Complexity of constraint satisfaction problems over finite subsets of natural numbers. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, pages 32:1–32:13, 2016. doi:10.4230/LIPIcs.MFCS.2016.32.

6 T. Dose. Balance problems for integer circuits. Technical Report 18-055, Electronic Colloquium on Computational Complexity (ECCC), 2018. URL: https://eccc.weizmann.ac.il/report/2018/055.

7 K. Ford. integers with a divisor in $(y, 2y]$. In *Anatomy of integers*, volume 46 of *CRM Proc. and Lect. Notes*, pages 65–81. Amer. Math. Soc., Providence, RI, 2008.

8 K. Ford. the distribution of integers with a divisor in a given interval. *Annals of Math. (2)*, 168:367–433, 2008.

9 C. Glaßer, K. Herr, C. Reitwießner, S. D. Travers, and M. Waldherr. Equivalence problems for circuits over sets of natural numbers. *Theory Comput. Syst.*, 46(1):80–103, 2010. doi:10.1007/s00224-008-9144-8.

10 C. Glaßer, P. Jonsson, and B. Martin. Circuit satisfiability and constraint satisfaction around skolem arithmetic. *Theor. Comput. Sci.*, 703:18–36, 2017. doi:10.1016/j.tcs.2017.08.025.

11 C. Glaßer, C. Reitwießner, S. D. Travers, and M. Waldherr. Satisfiability of algebraic circuits over sets of natural numbers. *Discrete Applied Mathematics*, 158(13):1394–1403, 2010. doi:10.1016/j.dam.2010.04.001.

12 Thomas Gundermann, Nasser Ali Nasser, and Gerd Wechsung. A survey on counting classes. In *Proceedings: Fifth Annual Structure in Complexity Theory Conference, Universitat Politècnica de Catalunya, Barcelona, Spain, July 8-11, 1990*, pages 140–153, 1990. doi:10.1109/SCT.1990.113963.

13 D. Koukoulopoulos. On the number of integers in a generalized multiplication table. *Journal für die reine und angewandte Mathematik*, 689:33–99, 2014. doi:10.1515/crelle-2012-0064.

14 Y. V. Matiyasevich. Enumerable sets are Diophantine. *Doklady Akad. Nauk SSSR*, 191:279–282, 1970. Translation in Soviet Math. Doklady, 11:354–357, 1970.

15 P. McKenzie and K. W. Wagner. The complexity of membership problems for circuits over sets of natural numbers. *Computational Complexity*, 16(3):211–244, 2007. doi:10.1007/s00037-007-0229-6.

16 C. M. Papadimitriou. *Computational complexity.* Addison-Wesley, Reading, Massachusetts, 1994.

17 I. Pratt-Hartmann and I. Düntsch. Functions definable by arithmetic circuits. In *Mathematical Theory and Computational Practice, 5th Conference on Computability in Europe, CiE 2009, Heidelberg, Germany, July 19-24, 2009. Proceedings*, pages 409–418, 2009. doi:10.1007/978-3-642-03073-4_42.

18 L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, pages 1–9, 1973. doi:10.1145/800125.804029.

19 S. D. Travers. The complexity of membership problems for circuits over sets of integers. *Theor. Comput. Sci.*, 369(1-3):211–229, 2006. doi:10.1016/j.tcs.2006.08.017.

20 K. W. Wagner. The complexity of problems concerning graphs with regularities (extended abstract). In *Mathematical Foundations of Computer Science 1984, Praha, Czechoslovakia, September 3-7, 1984, Proceedings*, pages 544–552, 1984. doi:10.1007/BFb0030338.

21 K. Yang. Integer circuit evaluation is pspace-complete. *J. Comput. Syst. Sci.*, 63(2):288–303, 2001. doi:10.1006/jcss.2001.1768.

On Hadamard Series and Rotating $\mathbb{Q}$-Automata

Louis-Marie Dando
LaBRI UMR 5800, Université de Bordeaux, INP Bordeaux, CNRS
Bordeaux, FRANCE
louis-marie.dando@labri.fr

Sylvain Lombardy
LaBRI UMR 5800, Université de Bordeaux, INP Bordeaux, CNRS
Bordeaux, FRANCE
sylvain.lombardy@labri.fr

Abstract

In this paper, we study rotating $\mathbb{Q}$-automata, which are (memoryless) automata with weights in $\mathbb{Q}$, that can read the input tape from left to right several times. We show that the series realized by valid rotating $\mathbb{Q}$-automata are $\mathbb{Q}$-Hadamard series (which are the closure of $\mathbb{Q}$-rational series by pointwise inverse), and that every $\mathbb{Q}$-Hadamard series can be realized by such an automaton. We prove that, although validity of rotating $\mathbb{Q}$-automata is undecidable, the equivalence problem is decidable on rotating $\mathbb{Q}$-automata. Finally, we prove that every valid two-way $\mathbb{Q}$-automaton admits an equivalent rotating $\mathbb{Q}$-automaton. The conversion, which is effective, implies the decidability of equivalence of two-way $\mathbb{Q}$-automata.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory, Theory of computation → Quantitative automata, Theory of computation → Algebraic language theory

Keywords and phrases Rational series, Hadamard operations, Rotating automata, Two-way automata

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.6

1 Introduction

Rotating automata are a natural model of automata. They were first considered as a restriction of two-way automata [8, 11]. In this model, the automaton reads the input from the left to the right, but when the right end of the input is reached, either the automaton stops, or the tape is rewinded back to the left end of the input. In the Boolean case, rotating automata are as expressive as NFA, but they have been studied since their size can be much smaller [8]. In particular they can compute the intersection of two regular languages with a linear number of states.

In the framework of weighted automata or transducers, rotating automata are more expressive than one-way automata or transducers [9]. In particular, in *rationally additive semirings* [5], they realize Hadamard series, which are the closure of rational series by Hadamard product and Hadamard inverse. This is a sound class of series, and in this framework, algorithms have been defined to convert rotating automata to expressions describing Hadamard series and to synthesize rotating automata from such expressions [4].

Hadamard series over a field have been studied for a long time [12] and it is not surprising that rotating automata can realize them. Nevertheless, fields are not rationally additive semirings, and the potentially infinite number of runs for some input must be handled in a more subtle way to translate rotating automata to Hadamard series. In this paper we prove that the set of Hadamard series over $\mathbb{Q}$ is exactly the behaviour of rotating $\mathbb{Q}$-automata.

We also prove that two-way $\mathbb{Q}$-automata realize the same class of series. This is an extension of the work of Anselmo and Bertoni on probabilistic two-way automata [1]. In contrast, sweeping transducers with unary outputs are equivalent to rotating transducers and they are weaker than two-way transducers [7]. Notice that the algebraic characterization of series realized by weighted two-way automata is in general not achieved.

In Section 2, we briefly recall the definition of rational series and weighted automata. In Section 3, we define Hadamard series, which form a strictly larger class than rational series, and study different ways to describe them as well as the conversions between these descriptions. Finally we show that it is undecidable whether the description of a Hadamard series, called a Hadamard expression, is well defined, but in the case where two expressions actually define a Hadamard series, their equivalence is decidable.

In Section 4, we introduce rotating $\mathbb{Q}$-automata and we prove that they actually realize $\mathbb{Q}$-Hadamard series and that, conversely, a rotating $\mathbb{Q}$-automaton can be synthesized from any expression denoting a Hadamard series. In particular, rotating $\mathbb{Q}$-automata are strictly more expressive than one-way $\mathbb{Q}$-automata.

Finally, in Section 5 we formally define two-way $\mathbb{Q}$-automata and we show that they can be simulated by rotating $\mathbb{Q}$-automata. As a consequence, the equivalence of two-way $\mathbb{Q}$-automata is decidable.

2 Rational series and weighted automata

For every alphabet A, we denote A^* the free monoid generated by A. The set of formal power series over A^* with coefficients in $\mathbb{Q}$ is denoted $\mathbb{Q}\langle\langle A^*\rangle\rangle$. A *series* s in $\mathbb{Q}\langle\langle A^*\rangle\rangle$ is a mapping from A^* into $\mathbb{Q}$; the *coefficient* of a word w in s is denoted $\langle s, w\rangle$, the coefficient of the empty word is the *constant term* of the series, and s itself is denoted as a formal sum:

$$s = \sum_{w \in A^*} \langle s, w\rangle w. \tag{1}$$

The sum $s + t$ of two series is the pointwise sum. The *Cauchy product* is the extension of the usual polynomial product to series; the series 1 (where all the coefficients are 0 except the constant term equal to 1) is neutral for this product. The Cauchy product is associative and distributes over the sum. Hence $(\mathbb{Q}\langle\langle A^*\rangle\rangle, +, \cdot)$ is a semiring.

In a semiring, the *star* of an element is defined as the sum of the powers of the element, if it exists. Hence, in $(\mathbb{Q}\langle\langle A^*\rangle\rangle, +, \cdot)$, the star of s is defined if the constant term belongs to $]-1; 1[$; it is called the *Kleene star* of s and is denoted s^*. In contrast to the case of the Boolean semiring, the Kleene star is not idempotent in $\mathbb{Q}$, neither in series with coefficients in $\mathbb{Q}$.

The *support* of a series s is the set of words w such that $\langle s, w\rangle \neq 0$. A series with a finite support is a *polynomial*; if every word of A^* belongs to the support, we say that the series has a *full support*.

▶ **Definition 1.** The set $\mathbb{Q}\mathrm{Rat}A^*$ of *rational series* is the closure of polynomials in $\mathbb{Q}\langle\langle A^*\rangle\rangle$ under sum, Cauchy product and Kleene star.

Likewise, the set of $\mathbb{Q}_+$-rational series is the closure of polynomials with positive coefficients under sum, Cauchy product and Kleene star.

The *behaviour* of an automaton is the language or series that describes the result of the automaton on every input. For instance, the behaviour of an NFA is the language of accepted words. The behaviour of a $\mathbb{Q}$-automaton is the series which maps every word to the weight of this word in the automaton.

▶ **Definition 2.** A $\mathbb{Q}$-automaton over an alphabet A is a tuple $\mathcal{A} = (Q, E, I, T)$, where
- Q is a finite set of states;
- I and T are, respectively, the initial and the final weight functions from Q into $\mathbb{Q}$;
- E is the transition weight function from $Q \times A \times Q$ into $\mathbb{Q}$.

The set of initial (*resp.* final) states is the support of I (*resp.* T), *i.e.* the states p such that $I(p) \neq 0$ (*resp.* $T(p) \neq 0$). The set of transitions is the support of E.

As usual, a *run* with label $w = w_1 w_2 \dots w_k$ in A^* is a sequence of states $(p_i)_{i \in [\![0;k]\!]}$ such that p_0 is an initial state, p_k is a final state, and for every i in $[\![1;k]\!]$, (p_{i-1}, w_i, p_i) is a transition. The *weight* of the run is equal to $I(p_0).\prod_{i=1}^{k} E(p_{i-1}, x_i, p_i).T(p_k)$. The weight $\mathcal{A}(w)$ of a word w in $\mathcal{A}$ is the sum of all runs with label w. The behaviour of $\mathcal{A}$ is the series $\sum_{w \in A^*} \mathcal{A}(w)w$. We say that $\mathcal{A}$ *realizes* the series s if s is the behaviour of $\mathcal{A}$. Like regular languages are the behaviour of NFA, by the Kleene-Schützenberger Theorem [15], $\mathbb{Q}$-rational series are the behaviour of one-way $\mathbb{Q}$-automata.

In the sequel, it will be useful to consider automata with particular properties.

▶ **Lemma 3.** *Every $\mathbb{Q}$-automaton is equivalent to an automaton with a single initial state with initial weight 1 and such that the weight of every transition is positive.*

Proof. Let $\mathcal{A} = (Q, E, I, T)$ be a $\mathbb{Q}$-automaton. We first show that $\mathcal{A}$ is equivalent to an automaton with a single initial state with initial weight 1. An initial state i is added; for every state q and every letter a, we extend E with $E(i, a, q) = \sum_p I(p).E(p, a, q)$, and T with $T(i) = \sum_p I(p).T(p)$. The automaton $\mathcal{A}' = (Q \cup \{i\}, E, \chi_i, T)$, where χ_i is the characteristic function of i, is then equivalent to $\mathcal{A}$.
We assume now that $\mathcal{A} = (Q, E, \chi_i, T)$ has a single initial state with initial weight 1. We define $\mathcal{B} = (Q \times \{-1, 1\}, E', \chi_{(i,1)}, T')$ such that, for every p, q in Q, every letter a, and every i, j in $\{-1, 1\}$,

$$E'((p,i),a,(q,j)) = \begin{cases} i.j.E(p,a,q) & \text{if } i.j.E(p,a,q) > 0, \\ 0 & \text{otherwise;} \end{cases} \qquad T'(p,i) = iT(p). \tag{2}$$

There is a natural bijection between runs of $\mathcal{A}$ and runs of $\mathcal{B}$ such that corresponding runs have the same weights, hence $\mathcal{B}$ is equivalent to $\mathcal{A}$. ◀

3 Hadamard series

3.1 Hadamard operations

We consider now the Hadamard product of two series: if s and t are two series in $\mathbb{Q}\langle\langle A^* \rangle\rangle$, then for every word w in A^*, $\langle s \odot t, w\rangle = \langle s, w\rangle.\langle t, w\rangle$. This product is also called the pointwise product. The neutral for this product is the series where the coefficient of every word is 1; this series is the characteristic series of A^* and is denoted itself A^*. Like in any commutative semiring, the Hadamard product preserves the rationality of series.

▶ **Proposition 4** ([16])**.** *The Hadamard product of two $\mathbb{Q}$-rational series is a $\mathbb{Q}$-rational series.*

This result is constructive (*cf.* for instance [13]): if two rational series are realized by two $\mathbb{Q}$-automata, their Hadamard product is realized by the *direct product* of the $\mathbb{Q}$-automata. As usual, the Hadamard power of a series can be defined from the Hadamard product. The 0-th Hadamard power of every series is the neutral for the Hadamard product, that is A^*. If

s is a series with each coefficient in $]-1;1[$, then the sum of the Hadamard powers of s is defined; it is called the Hadamard iteration of s and denoted $s^{\circledast}$.

Likewise, if the support of s is full, the Hadamard inverse of s is defined, and for every series t, the Hadamard quotient of t by s is defined as

$$\forall w \in A^*,\ \langle \odot\frac{t}{s}, w\rangle = \frac{\langle t, w\rangle}{\langle s, w\rangle}. \tag{3}$$

With this notation, the *Hadamard inverse* of s is $\odot\frac{A^*}{s}$.

▶ **Definition 5.** The set $\mathbb{Q}\mathrm{Had}A^*$ of Hadamard series over $\mathbb{Q}$ is the set of series which are equal to the Hadamard quotient of two rational series.

The set of Hadamard series is closed under sum and Hadamard product:

$$\odot\frac{t}{s} \odot \odot\frac{t'}{s'} = \odot\frac{t \odot t'}{s \odot s'}, \qquad \odot\frac{t}{s} + \odot\frac{t'}{s'} = \odot\frac{t \odot s' + t' \odot s}{s \odot s'}. \tag{4}$$

If x is a rational number in $]-1;1[$, the sum of powers of x is a rational number $x^* = (1-x)^{-1}$. Since the Hadamard product is a pointwise operation, this extends to formal power series: if every coefficient of a series s is in $]-1;1[$, then

$$s^{\circledast} = \odot\frac{A^*}{A^* - s}. \tag{5}$$

Therefore, series generated from rational series using sum, Hadamard product, and Hadamard iteration are Hadamard series.
Conversely, the Hadamard inverse can also be computed from the Hadamard iteration. To this end, for every rational number λ, we define $\mathsf{Geom}(\lambda)$ as the series over A^* such that, for every word w, $\langle \mathsf{Geom}(\lambda), w\rangle = \lambda^{|w|+1}$.

▶ **Lemma 6.** *Let t be a $\mathbb{Q}$-rational series such that, for every word w, $\langle t, w\rangle > 0$.*[1] *There exists a positive rational number λ such that $t' = A^* - t \odot \mathsf{Geom}(\lambda)$ is a $\mathbb{Q}_+$-rational series where every coefficient is in $[0;1[$.*

Proof. By Lemma 3, there exists a $\mathbb{Q}$-automaton $\mathcal{A} = (Q, E, I, T)$ that realizes t, with a single initial state i with weight 1 and positive transitions.

Let $M = \max(\max_{p,a} \sum_q E(p,a,q), \max_p T(p))$ and $\lambda = 1/M$. We define $\mathcal{B} = (Q \cup \{\bot\}, E', I, T')$, where $\bot$ is a fresh state and, for every p, q in $Q \cup \{\bot\}$ and every letter a,

$$E'(p,a,q) = \begin{cases} E(p,a,q).\lambda & \text{if } p,q \in Q, \\ 1 - \sum_{r \in Q} E(p,a,r).\lambda & \text{if } p \in Q \text{ and } q = \bot, \\ 1 & \text{if } p = q = \bot, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Depending on T', the automaton $\mathcal{B}$ may realize different series:

[1] In particular, t has full support.

(a) $T'(p) = T(p).\lambda$ for every p in Q and $T'(\bot) = 0$; then there is a natural bijection between runs of $\mathcal{B}$ and runs of $\mathcal{A}$; the weight of each run with label w in $\mathcal{B}$ is equal to the weight of the corresponding run in $\mathcal{A}$ multiplied by $\lambda^{|w|+1}$. Hence $\langle \mathcal{B}, w\rangle = \langle t, w\rangle.\lambda^{|w|+1}$.

(b) $T'(p) = 1$ for every p in $Q \cup \{\bot\}$; for every letter a and every state p, the sum of weights of outgoing transitions from p with label a is equal to 1. As there is only one initial state with weight 1, the weight of every word in $\mathcal{B}$ is 1.

(c) $T'(p) = 1 - T(p).\lambda$ for every p in Q and $T'(\bot) = 1$. In this case, the behaviour is the difference between the two previous behaviours and $\langle \mathcal{B}, w\rangle = 1 - \langle t, w\rangle.\lambda^{|w|+1}$, which is in $[0; 1[$.

In the last case, every weight in $\mathcal{B}$ is positive; hence $\mathcal{B}$ is a $\mathbb{Q}_+$-automaton which realizes a $\mathbb{Q}_+$-rational series. ◀

▶ **Proposition 7.** *For every $\mathbb{Q}$-rational series s with full support, there exists a positive rational number λ such that the Hadamard iteration of $A^* - s \odot s \odot \mathsf{Geom}(\lambda)$ is defined and*

$$\odot\frac{A^*}{s} = (A^* - s \odot s \odot \mathsf{Geom}(\lambda))^{\circledast} \odot s \odot \mathsf{Geom}(\lambda). \tag{7}$$

Proof. If s is a $\mathbb{Q}$-rational series with full support, $t = s \odot s$ is also rational with positive coefficients. Hence, by Lemma 6, there exists λ such that $1 - \langle s, w\rangle^2.\lambda^{|w|+1}$ is in $[0; 1[$ for every word w, and

$$(A^* - t \odot \mathsf{Geom}(\lambda))^{\circledast} \odot s \odot \mathsf{Geom}(\lambda) = \odot\frac{A^*}{t \odot \mathsf{Geom}(\lambda)} \odot s \odot \mathsf{Geom}(\lambda) = \odot\frac{A^*}{s}. \tag{8}$$

◀

Finally, our definition of Hadamard series is consistent with the definition of [4]:

▶ **Corollary 8.** *$\mathbb{Q}\mathrm{Had}A^*$ is the closure of $\mathbb{Q}\mathrm{Rat}A^*$ under Hadamard product, sum, and Hadamard iteration.*

▶ **Proposition 9.** *There exist $\mathbb{Q}$-Hadamard series which are not rational.*

Proof. Let t be the series such that, for every k in $\mathbb{N}$, $\langle t, a^k\rangle = k + 1$. It is a rational series: $t = a^*.a^*$. Let s be the Hadamard inverse of t: for every k in $\mathbb{N}$, $\langle s, a^k\rangle = \frac{1}{k+1}$. The coefficients of a rational series over one variable satisfy a linear recurrence relation (*cf.* for instance [2]). Therefore s is not rational. ◀

▶ **Proposition 10.** *The set $\mathbb{Q}\mathrm{Had}A^*$ is not closed under Cauchy product.*

Proof. Let s be the series defined in the proof of Proposition 9. We consider the Cauchy product of s with itself; for large integers k, it holds:

$$\langle s.s, a^k\rangle = \frac{2}{k+2}\sum_{i=0}^{k}\frac{1}{i+1} \sim \frac{2\ln k}{k}. \tag{9}$$

If $s.s$ is a Hadamard series, it is the Hadamard quotient of two rational series x and y. If x (*resp.* y) is rational, its coefficients satisfy a linear recurrence relation. The ratio of $\langle x, a^k\rangle$ and $\langle y, a^k\rangle$ can not be equivalent to $\frac{2\ln k}{k}$. ◀

▶ Remark. By Lemma 6, if a series is the Hadamard quotient of two $\mathbb{Q}_+$-rational series, it is also in the closure of $\mathbb{Q}_+\mathrm{Rat}A^*$ under sum, Hadamard product and Hadamard iteration.

Nevertheless, it is unknown whether a series in this closure is always the Hadamard quotient of two $\mathbb{Q}_+$-rational series.

3.2 Validity and equivalence

Rational series over $\mathbb{Q}$ are represented by rational expressions. It is well known that such an expression may be non valid if the star operator is applied to a subexpression whose interpretation is a series s on which the star is not defined. Nevertheless, the star can be applied if and only if the constant term of s is in $]-1;1[$; this condition is decidable on rational expressions; thus it is decidable whether a rational expression is valid.

There are two ways to describe $\mathbb{Q}$-Hadamard series. First, a $\mathbb{Q}$-Hadamard series is a quotient of s by t where s and t are two $\mathbb{Q}$-rational series; hence, it can be described as a pair of rational expressions. Such a representation is valid if and only if both rational expressions are valid and t has full support.

It is undecidable whether a $\mathbb{Q}$-rational series has full support (*cf.* [2, 13]). Notice that it is decidable whether a $\mathbb{Q}_+$-rational series has full support.

▶ **Proposition 11.** *It is undecidable whether the representation of a $\mathbb{Q}$-Hadamard series as a pair of $\mathbb{Q}$-rational series is valid.*
If the denominator series is $\mathbb{Q}_+$-rational, the validity of the representation is decidable.

Another description of $\mathbb{Q}$-Hadamard series consists in the application of pointwise operators (sum, Hadamard product, Hadamard iteration) to $\mathbb{Q}$-rational expressions. This leads to $\mathbb{Q}$-Hadamard expressions as defined in [4]. The validity of $\mathbb{Q}$-Hadamard expressions is undecidable, as well as the validity of $\mathbb{Q}_+$-Hadamard expressions.

▶ **Proposition 12.** *It is undecidable whether a $\mathbb{Q}_+$-Hadamard expression is valid.*

Proof. It is undecidable whether a probabilistic automaton with weights in $\{0; \frac{1}{2}; 1\}$ accepts some word with a probability larger than or equal to $\frac{1}{2}$ [10]. Let s be the $\mathbb{Q}_+$-rational series which is the behaviour of this automaton. It is undecidable whether $(2s)^{\circledast}$ is defined. ◀

Assume now that $h_1 = (s_1, t_1)$ and $h_2 = (s_2, t_2)$ are two valid representations of $\mathbb{Q}$-Hadamard series as pairs of $\mathbb{Q}$-rational series. Then h_1 and h_2 represent the same series if and only if $s_1 \odot t_2 = s_2 \odot t_1$. The Hadamard product of two $\mathbb{Q}$-rational series is a computable $\mathbb{Q}$-rational series (Prop. 4), and the equivalence of $\mathbb{Q}$-rational series is decidable (*cf.* [2, 13]).

If $\mathbb{Q}$-Hadamard series are described by $\mathbb{Q}$-Hadamard expressions, these descriptions can be converted into pairs of $\mathbb{Q}$-rational series. Actually, using Equations (4) and (5), every $\mathbb{Q}$-Hadamard expression can be turned into a pair of expressions which are combinations of $\mathbb{Q}$-rational expressions connected with sum and Hadamard product operators. Each of these expressions denotes a $\mathbb{Q}$-rational series.

▶ **Theorem 13.** *The equivalence of valid descriptions of $\mathbb{Q}$-Hadamard series is decidable.*

3.3 Extension to real and complex numbers

All results presented in this section apply directly to series over $\mathbb{R}$, up to the calculability of operations with real numbers.

To apply them on series over $\mathbb{C}$, we need to consider the complex conjugacy. The conjugacy commutes with rational operations; hence, if s is a $\mathbb{C}$-rational series, its complex conjugacy $\overline{s}$ is also a $\mathbb{C}$-rational series. Thus, Proposition 7 can be extended.

▶ **Proposition 14.** *For every $\mathbb{C}$-rational series s with full support, there exists a positive real number λ such that the Hadamard iteration of $A^* - s \odot \overline{s} \odot \mathsf{Geom}(\lambda)$ is defined and*

$$\odot\frac{A^*}{s} = (A^* - s \odot \overline{s} \odot \mathsf{Geom}(\lambda))^{\circledast} \odot \overline{s} \odot \mathsf{Geom}(\lambda). \tag{10}$$

Actually, $s \odot \overline{s}$ is a $\mathbb{C}$-rational series whose coefficients are positive real numbers, hence, it is a $\mathbb{R}$-rational series [6] and Lemma 6 applies.

4 Weighted rotating automata

A rotating $\mathbb{Q}$-automaton is a $\mathbb{Q}$-automaton that can read its input from left to right several times. To this end, it is endowed with transitions with a special label $\mathbf{r}$, which is not in A.

A run of a rotating $\mathbb{Q}$-automaton is accepting for a word w in A^* if the label of the run is in $(w\mathbf{r})^*w$.

For every word w, if the sum of the weights of the accepting runs for w is defined, the weight $\mathcal{A}(w)$ of w in $\mathcal{A}$ is equal to this sum. The automaton is *valid* if the weight of every word in A^* is defined.

▶ Remark. Every one-way $\mathbb{Q}$-automaton over an alphabet A can be considered as a rotating $\mathbb{Q}$-automaton without any transition with label $\mathbf{r}$. In this case, every run accepting a word w has label w, hence, its behaviour as a rotating $\mathbb{Q}$-automaton is the same as its behaviour as a one-way $\mathbb{Q}$-automaton.

▶ **Proposition 15.** *The behaviour of a valid rotating $\mathbb{Q}$-automaton is a $\mathbb{Q}$-Hadamard series.*

Proof. $\mathbb{Q}_+ \cup \{\infty\}$ is a *rationally additive semiring*: the star of every element is defined. By [9], every rotating $(\mathbb{Q}_+ \cup \{\infty\})$-automaton realizes a Hadamard series. Thus every valid rotating $\mathbb{Q}_+$-automaton realizes a $\mathbb{Q}$-Hadamard series. If $\mathcal{A}$ is a valid rotating $\mathbb{Q}$-automaton, its behaviour s can be split into $s = s_+ - s_-$, where s_+ and s_- are realized by rotating $\mathbb{Q}_+$-automata. This construction is similar to the one described in the proof of Lemma 3. Since s_+ and s_- are $\mathbb{Q}$-Hadamard series, the behaviour of $\mathcal{A}$ is a $\mathbb{Q}$-Hadamard series. ◀

▶ Remark. In the definition of validity, it is not assumed that the potentially infinite sums of weights are in $\mathbb{Q}$ (they might be in $\mathbb{R}$); it appears that these sums can be computed through the rational operations (sum, product and star), hence, if they are defined, they belong to $\mathbb{Q}$.

▶ Remark. Clearly, Proposition 15 extends to $\mathbb{R}$. It also holds for rotating $\mathbb{C}$-automata. Using a construction similar to the construction in the proof of Lemma 3 (with $Q' = Q \times \{-1, 1, i, -i\}$), one shows that every series s with coefficients in $\mathbb{C}$ can be split into four series with positive real coefficients: $s = s_{\mathrm{re}+} - s_{\mathrm{re}-} + i.s_{\mathrm{im}+} - i.s_{\mathrm{im}-}$.

The following proposition is a corollary of Proposition 15 and Theorem 13, since every rotating $\mathbb{Q}$-automaton can be turned into a $\mathbb{Q}$-Hadamard expression.

▶ **Proposition 16.** *The equivalence of valid rotating $\mathbb{Q}$-automata is decidable.*

We prove now that every $\mathbb{Q}$-Hadamard series can be realized by a rotating $\mathbb{Q}$-automaton. If two automata $\mathcal{A}$ and $\mathcal{B}$ respectively realize series s and t, it is straightforward that the union of $\mathcal{A}$ and $\mathcal{B}$ realizes the series $s + t$. Likewise, $s \odot t$ is realized by the automaton $\mathcal{A} \odot \mathcal{B}$ based on the union of $\mathcal{A}$ and $\mathcal{B}$, where

- for every final state p of $\mathcal{A}$ and every initial state q of $\mathcal{B}$ there is a transition with label $\mathbf{r}$ and weight $T_{\mathcal{A}}(p)I_{\mathcal{B}}(q)$;
- the initial function is restricted to states of $\mathcal{A}$ and the final function to states of $\mathcal{B}$.

In order to realize $s^{\circledast}$, a similar construction could be applied to $\mathcal{A}$ by adding transitions with label $\mathbf{r}$ from final states to initial states. This construction may lead to an infinite number of runs accepting a given word w, and, even if the weight of w is in $]-1; 1[$, there is no guarantee that the sum of all these runs is defined. Therefore, we consider that $\mathbb{Q}$-Hadamard series are Hadamard quotients of $\mathbb{Q}$-rational series and we prove that such a quotient can be realized by a rotating $\mathbb{Q}$-automaton.

▶ **Proposition 17.** *Let s be a $\mathbb{Q}$-rational series with full support. There exists a valid rotating $\mathbb{Q}$-automaton $\mathcal{A}$ such that the behaviour of $\mathcal{A}$ is the Hadamard inverse of s.*

Proof. By Lemma 6, there exists a positive rational number λ such that $t = A^* - s \odot s \odot \mathsf{Geom}(\lambda)$ is a series with coefficients in $[0;1[$ that can be realized by a $\mathbb{Q}_+$-automaton $\mathcal{A} = (Q, E, I, T)$. Without loss of generality, we assume that $\mathcal{A}$ has a single initial state i with weight 1. Let $\mathcal{B}$ be the rotating $\mathbb{Q}$-automaton built from $\mathcal{A}$ by:

- adding a transition from each final state p to state i with label $\mathbf{r}$ and weight $T(p)$;
- adding a new state $\perp$, which is initial and final with weight 1, and such that there is a loop on the state with weight 1 for every label except $\mathbf{r}$.

Every run with label w in $\mathcal{B}$ is either a circuit in $\perp$ with weight 1, or a concatenation of runs in $\mathcal{A}$ (glued with $\mathbf{r}$-transitions). The weight of every run in $\mathcal{A}$ is positive and for every word w, the sum of the weights of all runs with label w in $\mathcal{A}$ is smaller than 1. Hence, the weight of w in $\mathcal{B}$ is defined: it is the star of the weight of w in $\mathcal{A}$. Therefore, $\mathcal{B}$ is valid and its behaviour is $(A^* - s \odot s \odot \mathsf{Geom}(\lambda))^{\circledast} = \odot\frac{A^*}{s \odot s \odot \mathsf{Geom}(\lambda)}$. It is then easy to build a rotating automaton that realizes $(A^* - s \odot s \odot \mathsf{Geom}(\lambda))^{\circledast} \odot s \odot \mathsf{Geom}(\lambda) = \odot\frac{A^*}{s}$. ◀

▶ **Theorem 18.** *The set of series which are behaviours of valid rotating $\mathbb{Q}$-automata is the set of $\mathbb{Q}$-Hadamard series.*

Rotating $\mathbb{Q}$-automata and $\mathbb{Q}$-Hadamard expressions are therefore equivalent. Proof of Proposition 11 applies on rotating $\mathbb{Q}$-automata and their validity is therefore undecidable.

▶ **Proposition 19.** *The validity of a rotating $\mathbb{Q}$-automaton is undecidable.*

A sweeping $\mathbb{Q}$-automaton is an automaton that can read its input from left to right and right to left, but can only change the direction of the reading head on one of the endmarkers. Every rotating $\mathbb{Q}$-automaton can be simulated by a sweeping $\mathbb{Q}$-automaton. Conversely, like in any commutative semiring, sweeping $\mathbb{Q}$-automata can be simulated by rotating $\mathbb{Q}$-automata.

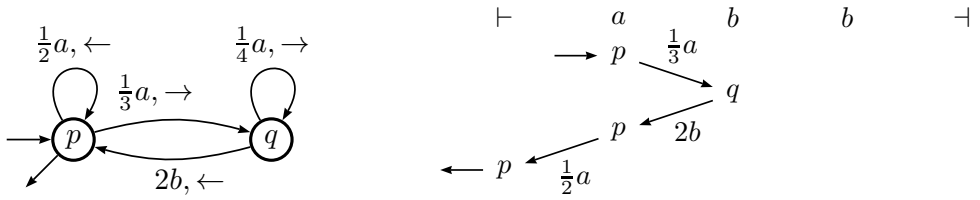
Hence, the validity of sweeping $\mathbb{Q}$-automata is undecidable, but the equivalence of valid sweeping $\mathbb{Q}$-automata is decidable.

5 From two-way to rotating automata

In this section, we formally define two-way $\mathbb{Q}$-automata and we show that they can be simulated by rotating $\mathbb{Q}$-automata. The proof presented here is inspired by the work in [1] on two-way probabilistic automata. Starting with a two-way $\mathbb{Q}$-automaton, for every word w, we define a matrix M_w such that the weight of w in the two-way $\mathbb{Q}$-automaton is an entry of M_w^*, the sum of iterated powers of M_w. We show that since M_w^* is the inverse of $\mathsf{Id} - M_w$, the weight of w can be computed as the ratio of an entry of the matrix of cofactors of $\mathsf{Id} - M_w$ by the determinant of $\mathsf{Id} - M_w$. We prove that both the entry of the matrix of cofactors and the determinant are rational power series (when w spans over A^*), therefore the behaviour of the two-way $\mathbb{Q}$-automaton is the Hadamard quotient of two rational series, *i.e.* a Hadamard series.

5.1 Weighted two-way automata

There are different models of two-way automata. If reading the left and right endmarks is allowed, they are all equivalent; depending on the model, the computation can start (*resp.* stop) at the beginning, the middle or the end of the word, and the move of the reading head can be performed in each state or during each transition traversal.



(a) The automaton $\mathcal{A}_1$. **(b)** A run over the word *abb*.

Figure 1 A two-way automaton and one run over the word *abb*.

For convenience in the conversion from two-way $\mathbb{Q}$-automata to rotating $\mathbb{Q}$-automata, we use in this paper two-way $\mathbb{Q}$-automata where computations start and stop at the left end of the word.

Formally, if A is an alphabet, a two-way $\mathbb{Q}$-automaton over A is a tuple $\mathcal{A} = (Q, E, I, T)$, where

- Q is the finite set of *states*;
- E is the *transition* function: $Q \times (A \cup \{\vdash, \dashv\} \times Q \longrightarrow \mathbb{Q}$;
- I and T are respectively the initial and final functions in $Q \longrightarrow \mathbb{Q}$.

A transition is an element t of $Q \times (A \cup \{\vdash, \dashv\} \times Q$ such that $E(t)$ is different from 0. Likewise a state p is initial if $I(p) \neq 0$; it is final if $T(p) \neq 0$.

The value in $\{-1, +1\}$ on each transition shows the direction of the move of the head of the automaton on the current letter. If it is equal to $+1$, the head moves forward and we can denote it by $\rightarrow$, if it is equal to -1, the head moves backward and it can be denoted $\leftarrow$. There is no transition with backward move and label $\vdash$; likewise, there is no transition with forward move and label $\dashv$.

A *path* compatible with a word $w = w_1 \dots w_k$ in A^* is a sequence of consecutive transitions $\rho = (p_{i-1}, x_i, m_i, p_i)_{i \in [\![1;\ell]\!]}$ such that there exists a function $\mathsf{pos} : [\![0;\ell]\!] \longrightarrow [\![0; k+1]\!]$ satisfying

- for every $i > 0$, $\mathsf{pos}(i) = \mathsf{pos}(i-1) + m_i$;
- for every $i \in [\![1;\ell]\!]$, if $\mathsf{pos}(i-1) = 0$, $x_i = \vdash$, and if $\mathsf{pos}(i-1) = k+1$, $x_i = \dashv$, otherwise $x_i = w_{\mathsf{pos}(i-1)}$.

If furthermore, p_0 is initial, $\mathsf{pos}(0) = 1$, p_ℓ is final, and $\mathsf{pos}(\ell) = 0$, then ρ is a *run* (over w).

The weight of such a run is $I(p_0).\left(\prod_{i=1}^{\ell} E(p_{i-1}, x_i, m_i, p_i)\right).T(p_\ell)$.

Notice that with this model, a run over the empty word w can exist; it contains at least one transition. For instance, if p is an initial state, q a final state and $(p, \dashv, \leftarrow, q)$ is a transition, there is a run over the empty word formed with this single transition.

The *weight* computed by $\mathcal{A}$ on a word w in A^* is the sum (if defined) of the weights of all runs over w. The automaton $\mathcal{A}$ is valid if for every word w, the sum of the weights of runs over w is defined.

▶ **Example 20.** Let $\mathcal{A}_1$ be the two-way automaton of Figure 1a. A run of this automaton over the word *abb* is described in Figure 1b; it is the path $(p, a, \rightarrow, q)(q, b, \leftarrow, p)(p, a, \leftarrow, p)$: it starts at position 1 in an initial state, and ends at position 0 in a final state.

We use a classical extension of the model of adjacency matrices for graphs. We suppose from now that $Q = [\![1; n]\!]$ where n is a positive integer. For every letter a in $A \cup \{\vdash, \dashv\}$ we define $F(a)$ (*resp.* $B(a)$) as the matrix of size $n \times n$ such that $F(a)_{p,q} = E(p, a, \rightarrow, q)$ (*resp.* $B(a)_{p,q} = E(p, a, \leftarrow, q)$).

These matrices represent paths of length 1. In graphs or one-way automata, paths of length k are represented by the k-th power of the adjacency matrix. In the case of a two-way automaton, the position of the head (given by the function pos) must be taken into account, and since there may exist runs with arbitrary large length for a given input, all the powers of the suitable matrix must be considered.

To this end, we define the star of a (square) matrix M as the (infinite) sum of its powers, if it is defined. It satisfies $M^* = \mathsf{Id} + M.M^*$, where Id is the identity matrix, and it can be inductively computed (cf [3]):

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^* = \begin{bmatrix} (A^*BD^*C)^*A^* & A^*(BD^*CA^*)^*BD^* \\ D^*C(A^*BD^*C)^*A^* & D^* + D^*CA^*(BD^*CA^*)^*BD^* \end{bmatrix}. \tag{11}$$

We fix from now w as a word with length k: $w = w_1 \dots w_k$. To study paths involved in the runs over w, we consider block matrices with dimension $(k+2) \times (k+2)$ where every entry is itself a $n \times n$ matrix. We assume that indices of blocks are integers in $[\![0; k+1]\!]$. If X is such a matrix, $X_{i,j}$ is a matrix and $(X_{i,j})_{p,q}$ represents some subpaths of runs over w which start in position i and state p, and end in position j and state q.

We first consider the matrix M_w that represents subpaths of runs over w with length 1:

$$M_w = \left[\begin{array}{c|c|ccc} 0 & F(\vdash) & 0 & \dots & 0 \\ \hline B(w_1) & 0 & F(w_1) & \dots & \vdots \\ \hline 0 & B(w_2) & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & 0 & F(w_k) \\ \hline 0 & \dots & 0 & B(\dashv) & 0 \end{array}\right]. \tag{12}$$

During the computation, if the automaton reads the letter w_i (in position i) and follows a forward transition, the head moves to position $i+1$. Hence, for every i, the matrix $F(w_i)$ is the block $(i, i+1)$ of M_w; likewise the matrix $B(w_i)$ is the block $(i, i-1)$, $F(\vdash)$ is the block $(0, 1)$, and $B(\dashv)$ is the block $(k+1, k)$.

We show that $(M_w^*)_{i,j}$ represent all the subpaths of runs over w which start in position i and end in position j.

For r in $[\![0; k+1]\!]$, let $C^{(r)}$ be the $(r+1) \times (r+1)$ block matrix where $C^{(r)}_{i,j}$ is a $n \times n$ matrix such that the entry at position (p, q) is the sum of the weights of the paths compatible with w from position i and state p to position j and state q with no position larger than r.

▶ **Lemma 21.** *With the notations above, $M_w^* = C^{(k+1)}$.*

The proof is by induction on k and on the star of the restriction of M_w to the $k+1$ first row blocks and the $k+1$ column blocks.

On a two-way automaton, the initial and final functions can respectively be seen as row and column vectors in $\mathbb{Q}^n$. We let L_i denote the $1 \times (k+1)$ block matrix where every block is null, except the i-th block, which is the identity matrix. Proposition 22 follows then from Lemma 21:

▶ **Proposition 22.** *The weight of w computed by $\mathcal{A}$ is equal to $I.L_2.M_w^*.{}^tL_1.T$.*

If $\mathcal{A}$ is valid, the star of M_w is defined for every word w, hence $\mathsf{Id} - M_w$ is invertible. Otherwise, there would exist a non zero vector v such that $M_w.v = v$; since M_w^* is defined, $M_w^*.v = (\mathsf{Id} + M_w^*.M_w).v = v + M_w^*.v$, and $v = 0$ which is a contradiction. Therefore, for every word w,

$$\langle \mathcal{A}, w\rangle = I.L_2.(\mathsf{Id} - M_w)^{-1}.{}^tL_1.T = \frac{1}{\det(\mathsf{Id} - M_w)} \sum_{p,q\in Q} I_p.(\operatorname{adj}(\mathsf{Id} - M_w))_{n+p,q}.T_q, \quad (13)$$

where $\operatorname{adj}(X)$ is the adjugate matrix of X and $\det(X)$ is the determinant of X. For every p, q in Q, let $\alpha_{p,q}$ be the series defined by $\langle \alpha_{p,q}, w\rangle = (\operatorname{adj}(\mathsf{Id} - M_w))_{n+p,q}$ and let δ be the series defined by $\langle \delta, w\rangle = \det(\mathsf{Id} - M_w)$. We show in the next part that all these series are $\mathbb{Q}$-rational.

5.2 Inductive computation of a determinant of a tridiagonal block matrix

Inductive computations of determinants of tridiagonal block matrices have already been studied [14]. We give here a new presentation of this computation in order to show that it can be realized by a (one-way) $\mathbb{Q}$-automaton.

Let n and k be two positive integers. We consider two families $(A_i)_{i\in[\![1;k]\!]}$ and $(A'_i)_{i\in[\![0;k]\!]}$ of matrices in $\mathbb{Q}^{n\times n}$.

For every r in $[\![0;k]\!]$, we consider the matrix $N^{(r)}$ in $\mathbb{Q}^{(r+1)n\times(r+2)n}$ defined as:

$$N^{(r)} = \begin{bmatrix} A'_r & \mathsf{Id}_n & A_r & & & 0 \\ 0 & A'_{r-1} & \mathsf{Id}_n & A_{r-1} & & \\ & & \ddots & \ddots & \ddots & \\ & & & A'_1 & \mathsf{Id}_n & A_1 \\ 0 & & & & A'_0 & \mathsf{Id}_n \end{bmatrix}, \quad (14)$$

where Id_n is the identity matrix with size n.

We introduce now some notations.

- For every set X and every positive integer i, $\mathcal{P}_iX$ denotes the set of subsets of X with i elements.
- If M is a matrix, the determinant of M is denoted $|M|$.
- If X is a set of indices, $\overline{X}$ is its complementary set, and ΣX is the sum of elements of X.
- If X and Y are two subsets of indices of a matrix M, $M_{X\times Y}$ is the restriction of M to rows in X and columns in Y.
- For every C in $\mathcal{P}_n[\![1;2n]\!]$, we let $G^{(r)}(C)$ denote the square matrix $N^{(r)}_{[\![1;(r+1)n]\!]\times\overline{C}}$.
- For every C, D in $\mathcal{P}_n[\![1;2n]\!]$,

$$K^{(r)}(C,D) = \begin{bmatrix} A'_r & \mathsf{Id}_n & A_r & 0 \\ 0 & A'_{r-1} & \mathsf{Id}_n & A_{r-1} \end{bmatrix}_{[\![1;2n]\!]\times(\overline{C}\cap[\![1;2n]\!])\cup(D+2n)}. \quad (15)$$

▶ **Lemma 23.** *For every $r \in [\![2;k]\!]$, for every C in $\mathcal{P}_n[\![1;2n]\!]$,*

$$\left|G^{(r)}(C)\right| = \sum_{D\in\mathcal{P}_n[\![1;2n]\!]} (-1)^{\mathsf{sig}(D+n)}.\left|K^{(r)}(C,D)\right|.\left|G^{(r-2)}(D)\right| \quad (16)$$

where $\mathsf{sig}(D+n) = \dfrac{n(n+1)}{2} + \Sigma D$.

The proof is an application of the Laplace expansion of the determinant:

$$\forall N \in \mathbb{Q}^{d\times d}, \forall X \subseteq [\![1;d]\!], \qquad |N| = \sum_{Y\in\mathcal{P}_{|X|}[\![1;d]\!]} (-1)^{\Sigma X+\Sigma Y} \, |N_{X\times Y}| \, . \left|N_{\overline{X}\times\overline{Y}}\right| . \tag{17}$$

We apply now Lemma 23 to the computation of the determinant of $\mathsf{Id} - M_w$. For every matrix X, we let ${}^{\circlearrowleft}X$ denote the rotation of X by half-turn; notice that $|X| = |{}^{\circlearrowleft}X|$. For every word $w = w_1 \dots w_k$, we set $A_0 = -{}^{\circlearrowleft}B(\dashv)$, $A'_{k+1} = -{}^{\circlearrowleft}F(\vdash)$, $A_i = -{}^{\circlearrowleft}B(w_i)$, and $A'_i = -{}^{\circlearrowleft}F(w_i)$, for every i in $[\![1;k]\!]$, then $G^{(k+1)}([\![1;n]\!])$ is equal to ${}^{\circlearrowleft}(\mathsf{Id} - M_w)$.

5.3 The transformation automaton

We describe the (one-way) automaton that computes $\det(\mathsf{Id} - M_w)$, based on the induction described in Lemma 23. The set of states is $\{i\} \cup \mathcal{P}_n[\![1;2n]\!] \cup A \times \mathcal{P}_n[\![1;2n]\!]$. The order of the induction is 2; there are two different kinds of initial states, depending on the parity of the length of the input.

- State i is initial with weight 1 and, for every D in $\mathcal{P}_n[\![1;2n]\!]$ and every a in A, there is a transition from i to D with label a and weight

$$\left|\begin{bmatrix} -{}^{\circlearrowleft}F(a) & \mathsf{Id}_n & -{}^{\circlearrowleft}B(a) \\ 0 & -{}^{\circlearrowleft}F(\vdash) & \mathsf{Id}_n \end{bmatrix}_{[\![1;2n]\!]\times\overline{D}}\right| . \tag{18}$$

- For every D in $\mathcal{P}_n[\![1;2n]\!]$, D is initial with weight $\left|\begin{bmatrix} -{}^{\circlearrowleft}F(\vdash) & \mathsf{Id}_n \end{bmatrix}_{[\![1;n]\!]\times\overline{D}}\right|$, and for every a in A, there is a transition from D to (a, D) with label a and weight $(-1)^{\mathsf{sig}(D+n)}$.
- For every C, D in $\mathcal{P}_n[\![1;2n]\!]$ and every a, b in A, there is a transition from (a, D) to C with label b and weight

$$\left|\begin{bmatrix} -{}^{\circlearrowleft}F(b) & I_n & -{}^{\circlearrowleft}B(b) & 0 \\ 0 & -{}^{\circlearrowleft}F(a) & \mathsf{Id}_n & -{}^{\circlearrowleft}B(a) \end{bmatrix}_{[\![1;2n]\!]\times(\overline{C}\cap[\![1;2n]\!])\cup(D+2n)}\right| . \tag{19}$$

- Every state (a, D) is final with weight $\left|\begin{bmatrix} \mathsf{Id}_n & -{}^{\circlearrowleft}B(\dashv) & 0 \\ -{}^{\circlearrowleft}F(a) & \mathsf{Id}_n & -{}^{\circlearrowleft}B(a) \end{bmatrix}_{[\![1;2n]\!]\times([\![1;n]\!]\cup D+n)}\right|$.

By Lemma 23, this automaton computes $\det(\mathsf{Id} - M_w)$ for every word w; it realizes the series δ which is thus $\mathbb{Q}$-rational.

Likewise, for every p, q in $[\![1;n]\!]$, $(\mathrm{adj}(\mathsf{Id} - M_w))_{p+n,q}$ is equal to the determinant of $C_w^{(q,p+n)}$, which is the matrix $\mathsf{Id} - M_w$ where every coefficient of the q-th row and every coefficient of the $p+n$-th column is replaced by 0, except the coefficient in $(q, p+n)$ which is replaced by 1. The determinant of $C_w^{(q,p+n)}$ can be computed with the same induction as for $\det(\mathsf{Id} - M_w)$ with different initial conditions.

Hence, for every p, q in Q, the series $\alpha_{p,q}$ is rational; so is the series $\alpha = \sum_{p,q\in Q} I_p.\alpha_{p,q}.T_p$. Finally, the series realized by the two-way $\mathbb{Q}$-automaton is the Hadamard quotient of two rational series. It is therefore a Hadamard series that can be realized by a rotating $\mathbb{Q}$-automaton.

▶ **Theorem 24.** *The set of series realized by two-way $\mathbb{Q}$-automata is exactly the set $\mathbb{Q}\mathrm{Had}A^*$ of series realized by rotating $\mathbb{Q}$-automata.*

Notice that the conversion of two-way $\mathbb{Q}$-automata to rotating $\mathbb{Q}$-automata is effective, hence the decidability of equivalence of rotating $\mathbb{Q}$-automata extends to two-way $\mathbb{Q}$-automata.

▶ **Theorem 25.** *The equivalence of valid two-way $\mathbb{Q}$-automata is decidable.*

6 Conclusion

The results presented in this paper can be extended to other fields. Section 5 actually shows that the behaviour of a two-way automaton over a field is the Hadamard quotient of two rational series. It extends to any field. Notice that the definition of the behaviour of a two-way automaton involves infinite sums; hence, a structure (topology for instance) is required to handle these sums, and this structure must transfer to matrices to define the star of a matrix.

To show that two-way automata are equivalent to rotating automata, it must be proved that every Hadamard quotient of two rational series can be realized by a rotating automaton (Theorem 18), and Proposition 7 is crucial in this proof. This proposition applies to $\mathbb{Q}$ and $\mathbb{R}$; it can also be used to prove Theorem 18 in $\mathbb{C}$. For other fields, dedicated proofs should be provided.

The conversion from two-way $\mathbb{Q}$-automata to rotating $\mathbb{Q}$-automata heavily relies on the inversion of a matrix describing the computations in the two-way $\mathbb{Q}$-automaton. This inversion is considered as the quotient between some coefficients of the adjugate matrix and the determinant and we exhibit one-way $\mathbb{Q}$-automata that realize this computation. It is still an open question to find a more combinatorial argument to the equivalence between the two models. It could help in characterizing semirings where two-way and rotating automata are equivalent. It could also lead to more efficient algorithms to convert two-way $\mathbb{Q}$-automata into rotating $\mathbb{Q}$-automata or to compute $\mathbb{Q}$-Hadamard expressions of the series they realize.

References

1 Marcella Anselmo and Alberto Bertoni. Two-way probabilistic automata and rational power series. In *Proc. IV Conv. It. Inform. Teor.*, pages 9–23. World Scientific, 1992.

2 Jean Berstel and Christophe Reutenauer. *Noncommutative Rational Series with Applications*, volume 137 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2010.

3 John H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, London, 1971.

4 Louis-Marie Dando and Sylvain Lombardy. From Hadamard Expressions to Weighted Rotating Automata and Back. In *CIAA'17*, volume 10328 of *LNCS*, pages 163–174. Springer, 2017. doi:10.1007/978-3-319-60134-2_14.

5 Zoltán Ésik and Werner Kuich. Rationally additive semirings. *J. UCS*, 8(2):173–183, 2002. doi:10.3217/jucs-008-02-0173.

6 Michel Fliess. Matrices de Hankel. *J. Math. Pures et Appl.*, 53:197–222, 1974.

7 Bruno Guillon. Input- or output-unary sweeping transducers are weaker than their 2-way counterparts. *RAIRO - Theor. Inf. and Applic.*, 50(4):275–294, 2016. doi:10.1051/ita/2016028.

8 Christos Kapoutsis, Richard Královič, and Tobias Mömke. Size complexity of rotating and sweeping automata. *Journal of Computer and System Sciences*, 78(2):537–558, 2012. doi:10.1016/j.jcss.2011.06.004.

9 Sylvain Lombardy. Two-way representations and weighted automata. *RAIRO - Theoretical Informatics and Applications*, 50(4):331–350, 2016. doi:10.1051/ita/2016026.

10 Azaria Paz. *Introduction to Probabilistic Automata (Computer Science and Applied Mathematics)*. Academic Press, Inc., Orlando, FL, USA, 1971.

11 Giovanni Pighizzini. Two-way finite automata: Old and recent results. *Fundam. Inform.*, 126(2-3):225–246, 2013. doi:10.3233/FI-2013-879.

12 Christophe Reutenauer. Sur les éléments inversibles de l'algèbre de Hadamard des séries rationnelles. *Bull. Soc. Math. France*, 110:225–232, 1982.

13 Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

14 Davod Khojasteh Salkuyeh. Comments on "a note on a three-term recurrence for a tridiagonal matrix". *Appl. Math. Comput.*, 176(2):442–444, 2006. doi:10.1016/j.amc.2005.09.033.

15 Marcel-Paul Schützenberger. On the definition of a family of automata. *Inform. and Control*, 4:245–270, 1961. doi:10.1016/S0019-9958(61)80020-X.

16 Marcel-Paul Schützenberger. On a theorem of R. Jungen. *Proc. Amer. Math. Soc.*, 13(6):885–890, 1962.

One-Sided Error Communication Complexity of Gap Hamming Distance

Egor Klenin
Lomonosov Moscow State University, Moscow, Russia
Moscow, 1 Leninskiye Gory, Russia
yegorklenin@gmail.com

Alexander Kozachinskiy[1]
National Research University Higher School of Economics, Moscow, Russia
Moscow, 3 Kochnovsky Proezd, Russia
akozachinskiy@hse.ru
https://orcid.org/0000-0002-9956-9023

Abstract

Assume that Alice has a binary string x and Bob a binary string y, both strings are of length n. Their goal is to output 0, if x and y are at least L-close in Hamming distance, and output 1, if x and y are at least U-far in Hamming distance, where $L < U$ are some integer parameters known to both parties. If the Hamming distance between x and y lies in the interval (L, U), they are allowed to output anything. This problem is called the Gap Hamming Distance. In this paper we study public-coin one-sided error communication complexity of this problem. The error with probability at most $1/2$ is allowed only for pairs at Hamming distance at least U. In this paper we determine this complexity up to factors logarithmic in L. The protocol we construct for the upper bound is simultaneous.

2012 ACM Subject Classification Theory of computation → Communication complexity

Keywords and phrases Communication Complexity, Gap Hamming Distance, one-sided error

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.7

1 Communication complexity of GHD

Given two strings $x = x_1 \dots x_n \in \{0,1\}^n$, $y = y_1 \dots y_n \in \{0,1\}^n$, Hamming distance between x and y is defined as the number of positions, where x and y differ:

$$d(x,y) = |\{i \in \{1,\dots,n\} \mid x_i \neq y_i\}|.$$

Let $L < U \leqslant n$ be integer numbers. In this paper we consider the following communication problem $\mathsf{GHD}_{L,U}$, called the Gap Hamming Distance problem:

▶ **Definition 1.** Let Alice receive an n-bit string x and Bob an n-bit string y such that either $d(x,y) \leqslant L$, or $d(x,y) \geqslant U$. They have to output 0, if the first inequality holds, and 1, if the second inequality holds. If the promise is not fulfilled, they may output anything.

[1] Supported in part by RFBR grapnt 16-01-00362 and by the Russian Academic Excellence Project "5-100".

1.1 Prior work

1.1.1 Two-sided error upper bounds

Let $R(\mathsf{GHD}_{L,U})$ denote randomized two-sided error public coin communication complexity of $\mathsf{GHD}_{L,U}$. It is known (see [11, 15]) that $R(\mathsf{GHD}_{L,U})) = O(L^2/(U-L)^2)$ (assuming the constant error probability less than 1/2).

The paper [9] established the upper bound $R(\mathsf{GHD}_{L,U}) = O(L \log L)$ in the case $U = L+1$, that is, there is no gap. This bound is much better than $O(L^2/(U-L)^2)$, which is $O(L^2)$ in this case.

It turns out that the protocols attaining these two upper bounds are *simultaneous*. That is, in these protocols Alice and Bob do not communicate at all, but rather send messages to the third party, Charlie, who then computes the output of the protocol. Charlie doesn't see inputs of Alice and Bob but sees public coins. The corresponding model, called simultaneous message passing (SMP) model, is even more restricted than one-way public-coin communication: every simultaneous protocol can be converted into one-way protocol without increasing communication cost.

1.1.2 One-sided error public coin communication protocols

The one-sided error public coin communication complexity will be denoted by R^0. The superscript 0 means that the protocol is allowed to err only for input pairs which are at least U-far in Hamming distance. Here we assume that the maximal probability of error[2] is 1/2 The superscript 1 will mean the opposite: protocols are allowed to err only for input pairs which are at least L-close in Hamming distance.

Let us first note that for all x, y we have

$$\mathsf{GHD}_{L,U}(x,y) = \neg\mathsf{GHD}_{n-U,n-L}(\neg x, y).$$

(Alice flips all bits of her input string.) Thus $\mathsf{GHD}_{L,U}(x,y)$ reduces to $\mathsf{GHD}_{n-U,n-L}$ and the other way around. This reduction maps 0-instances to 1-instances and vice verse. This observation implies that

$$R^1(\mathsf{GHD}_{L,U}) = R^0(\mathsf{GHD}_{n-U,n-L}).$$

Thus it suffices to study only one of these quantities and we will stick to R^0 (the error is allowed when the distance is at least U) .

The paper [8] noticed that for all $U > L$ it holds that $R^0(\mathsf{GHD}_{L,U}) = O(L \log n)$. Once again, there is a public coin SMP protocol attaining this bound (which is just a simple modification of the standard protocol for EQUALITY).

1.1.3 GHD and the lower bounds in data streams and property testing

Several works used GHD to obtain lower bounds for streaming algorithms and for property testing problems. As was discovered in [14] by Woodruff, there is a reduction from GHD to a number of fundamental data stream problems, including the problem of estimating frequency moments. More specifically, if there is a $\Omega(n)$ lower bound against any r-round two-sided error communication protocol for $\mathsf{GHD}_{n/2-\Theta(\sqrt{n}),\ n/2+\Theta(\sqrt{n})}$, then there is a $\Omega(1/(r\varepsilon^2))$ lower bound on the space complexity of any r-pass streaming algorithm estimating the frequency moments in a data stream within a factor of $(1+\varepsilon)$.

[2] By the standard amplification argument we could have any constant between 0 and 1 instead of 1/2.

In [14] Woodruff proved $\Omega(n)$ lower bound against 1-round protocols (see also [10] for more direct and simple proof). In a subsequent works ([2, 3]) this lower bound was extended to $O(1)$-round protocols. Finally, $\Omega(n)$ lower bound in the most general setting, when there is no restriction on the number of rounds at all, was obtained in [6, 13, 12].

As it turns out, lower bounds on the one-sided error version of GHD are also useful. In [5] Buhrman, Cleve and Wigderson proved that for any constant $c < 1$ it holds that $R^1(\mathsf{GHD}_{0,\,cn}) = \Omega(n)$. Moreover, they showed that $\Omega(n)$ lower bound holds also for a weaker version of $\mathsf{GHD}_{0,\,cn}$ problem, in which Hamming distance between the inputs is either 0 or *exactly* cn (provided that cn is an even integer).

Blais, Brody and Matulef in [1] used this result to obtain lower bounds on testing decision trees and signed majorities with one-sided error.

Further, Brody and Woodruff ([4]) used lower bound on one-sided error GHD from [5] to obtain lower bounds for streaming algorithms with *one-sided approximation*, i.e., for algorithms which either always return an overestimate or always return an underestimate on the objective function. Their results include lower bounds for the problem of over(under)-estimating the number of non-zero rows in a matrix and the Earth Mover Distance between two multisets.

1.2 This work

In this paper we study public-coin one-sided error communication complexity R^0 of $\mathsf{GHD}_{L,U}$. Once again, the error is allowed only for pairs at Hamming distance at least U.

1.2.1 The upper bound

Our main result is a one-sided error public-coin simultaneous protocol for $\mathsf{GHD}_{L,U}$ on n-bit strings with communication complexity $O((L^2/U)\log L)$. It is constructed in the following 4 steps (description of the protocol in this section is a bit informal, and the precise bounds can be found below in the paper). Let us stress that steps 1 and 2 are enough to obtain $O((L^2/U)\log n)$ solution; the purpose of steps 3 and 4 is to replace $O(\log n)$-factor by $O(\log L)$-factor. Importance of eliminating dependency on n in the upper bounds was also acknowledged in previous works ([15, 9]).

Step 1. On this step we construct our main novel protocol, called *the Triangle Inequality Protocol*. This protocol communicates $O((L^2/U)\log n)$ bits (which is a bit more than required, since $\log L$ is replaced by $\log n$) and solves the $\mathsf{GHD}_{L,U}$ problem when the ratio U/L is larger than a certain constant.

The protocol works as follows. It randomly splits x and y in $b = O(L^2/U)$ blocks $x^1, \ldots, x^b$ and $y^1, \ldots, y^b$. The ith bit x_i of x goes in the block x^j where j is chosen at random with uniform probability distribution over $\{1, \ldots, b\}$, and decisions for different i's are independent. Each bit y_i of y goes in the block y^j with the same index as x_i goes in. This partition is made using the shared random source (so that the parties have the same partition). Both parties also read random strings $r^1, \ldots, r^b$ from the shared random source and Alice communicates $d(x^j, r^j)$ to Charlie for all $j = 1, \ldots, b$. Bob does the same with $d(y^j, r^j)$. Thus the communication is $b \log n = O((L^2/U)\log n)$. Charlie outputs 0 if the sum

$$\sum_{j=1}^{b} |d(x^j, r^j) - d(y^j, r^j)|$$

is at most L and 1 otherwise. By the triangle inequality each term in this sum is at most $d(x^j, y^j)$ and thus the sum is at most $d(x, y)$. Therefore this protocol does not err if $d(x, y) \leqslant L$.

On the other hand, if $d(x, y) \geqslant U \geqslant C'L$ for a certain constant C', then for any fixed j the average value of $d(x^j, y^j)$ is at least 2. From the properties of binomial distributions it follows that we have $d(x^j, y^j) \geqslant d(x, y)/10b$ with probability at least $1/3$. The value $d(x^j, r^j) - d(y^j, r^j)$ is distributed as the distance from the origin in a random walk with $d(x^j, y^j)$ steps along a line (each step has length 1 and is directed to the left or to the right with equal probabilities). From the properties of random walks it follows that for every j we have $|d(x^j, r^j) - d(y^j, r^j)| > \sqrt{d(x^j, y^j)}$ with constant positive probability. These two facts imply that with constant probability the sum $\sum_{j=1}^{b} |d(x^j, r^j) - d(y^j, r^j)|$ is $\Omega(b\sqrt{d(x, y)/10b}) = \Omega(\sqrt{bd(x, y)})$. Recall that $b = O(L^2/U)$ and we assume that $d(x, y) \geqslant U$. If the constant hidden in O-notation is large enough then the lower bound $\Omega(\sqrt{bd(x, y)})$ for the sum $\sum_{j=1}^{b} |d(x^j, r^j) - d(y^j, r^j)|$ is larger than L.

Step 2. In [8] it was noticed that for all $L < U$ there is one-sided error public-coin simultaneous protocol for $\mathsf{GHD}_{L,U}$ with communication $O(L \log n)$. This protocol is just a modification of the standard protocol for equality and it never errs for inputs at distance at most L.

Our second protocol runs the Triangle Inequality Protocol if $U > C'L$ and the protocol from [8] otherwise. Notice that in the latter case $L = O(L^2/U)$, and thus we obtain a protocol with communication $O((L^2/U) \log n)$ for all L, U.

Step 3. On this step we use the techniques from [9] to replace the $\log n$ factor by a $\log L$ factor. More specifically, we run the protocol from the second step for the strings u, v of length $O(L^8)$ obtained from the original strings x, y by the following transformation. As in the Triangle Inequality Protocol, we split x, y into $b = O(L^8)$ blocks and then replace each block by the parity of its bits. Obviously, $d(u, v) \leqslant d(x, y)$. We then show that $d(u, v) = d(x, y)$ with constant probability provided $d(x, y) \leqslant L^4$. Therefore this protocol has constant one-sided error probability for all input pairs with $d(x, y) \leqslant L^4$. By construction this protocol communicates $O((L^2/U) \log L)$ bits.

Step 4. Finally, to handle the case $d(x, y) > L^4$, we consider the following protocol. We run the protocol from step 3 and then a simplified version of the Triangle Inequality Protocol. If any of these two protocols output 1, we output 1 and otherwise 0. The simplified version of the Triangle Inequality Protocol works as follows. Alice and Bob read a random n-bit string r from the shared random source. They compute distance from their inputs to r. Observe that due to triangle inequality $|d(x, r) - d(y, r)| \leqslant d(x, y)$. Hence $d(x, y) \leqslant L$ implies $|d(x, r) - d(y, r)| \leqslant L$. On the other hand, if $d(x, y) > L^4$, then due to the properties of random walks with constant positive probability it holds that $|d(x, r) - d(y, r)| > L^2$.

Thus step 4 is reduced to the following communication problem. Alice holds a number $a \in \{0, 1, \ldots, n\}$, Bob holds a number $b \in \{0, 1, \ldots, n\}$ and it is known that either $|a - b| \leqslant L$ or $|a - b| > L^2$. The goal is to find out whether the first or the second inequality is true. We construct a public-coin simultaneous protocol with communication $O(\log L)$ which always outputs 0 when $|a - b| \leqslant L$ and which with some constant positive probability outputs 1 when $|a - b| > L^2$.

There is a simple SMP protocol communicating $O(\log L + \log \log n)$ bits to solve even a gapless (L vs $L + 1$) version of this problem. Let $p_1, \ldots, p_k$ be the first $k = (4L + 2) \cdot \log_2(n + L)$

primes. Parties read a random $p \in \{p_1, \ldots, p_k\}$ from the shared random source. Alice and Bob communicate $a \pmod p$ and $b \pmod p$ to Charlie. He checks, whether there is $i \in [-L, L]$ such that $a + i \pmod p = b \pmod p$. If there is such i, he outputs 0, otherwise 1. A straightforward analysis shows that this protocol has an error at most $1/2$ only in the case when $|a - b| \geqslant L + 1$.

The problem with this protocol is that a and b range from 0 to n. To get rid of $O(\log \log n)$ term we do the following. Instead of taking remainders modulo $p_1, \ldots, p_k$ we just hash our $O(n)$-size universe into $O(L)$-size universe simply by taking remainder modulo $4L + 2$. Of course this may lead to a collision when two number which were far from each other become L-close. We resolve this issue by considering $Z_0 + \ldots + Z_a$ and $Z_0 + \ldots + Z_b$ instead of a and b, where $Z_0, \ldots, Z_n$ are independent symmetric Bernoulli random variables. It can be shown that provided $|a - b| > L^2$, the difference $Z_{a+1} + \ldots + Z_b$ is distributed almost uniformly modulo $4L + 2$. This guaranties that the collision probability is by a constant bounded away from 1.

1.2.2 Lower bounds

As it turns out, a very simple argument proves an almost matching $\Omega(L^2/U)$ lower bound. We include this argument for completeness.

As we mentioned, provided that U is even and $U = (1 - \Omega(1))n$, the paper [5] proves $\Omega(n)$ lower bound on one-sided error communication complexity R^1 of an easier version of $\mathsf{GHD}_{0,U}$, in which the distance between inputs is either 0 or exactly U. However, we need a lower bound in the regime when U is very close to n. We observe that a simple modification of a proof from [5] works as well in such regime when one switches to a harder problem, in which the distance between inputs can be greater than U. Namely, we show that $R^1(\mathsf{GHD}_{0,U}) = \Omega((n - U)^2/n)$ for $\mathsf{GHD}_{0\,U}$ on n-bit strings.

As a corollary we obtain the lower bound $\Omega(L^2/U)$ for one-sided error complexity R^0 of $\mathsf{GHD}_{L,U}$ (the error is allowed when the distance is at least U). As we explained earlier, R^1 of $\mathsf{GHD}_{0,U-L}$ on U-bit strings equals R^0 of $\mathsf{GHD}_{L,U}$ on U-bit strings. As the former is $\Omega((U - (U - L))^2/U)$ we obtain the lower bound $\Omega(L^2/U)$ for the latter. On the other hand, the problem $\mathsf{GHD}_{L,U}$ on U-bit strings reduces to the problem $\mathsf{GHD}_{L,U}$ on n-bit strings (Alice and Bob append $n - U$ zeros to their strings), hence the one-sided complexity R^0 of the latter is also $\Omega(L^2/U)$.

1.2.3 The summary

Let us summarize our results.

▶ **Theorem 2.** *The one-sided error public-coin communication complexity R^0 of $\mathsf{GHD}_{L,U}$ on n-bit strings is at most*

$$O\left(\left(\frac{L^2}{U} + 1\right)\log(L + 2)\right)$$

(The error is allowed only when the distance is at least U.) There is a public-coin simultaneous protocol attaining this bound.

▶ **Theorem 3.** *The one-sided error public-coin communication complexity R^1 of $\mathsf{GHD}_{0,U}$ on n-bit strings is at least*

$$\Omega\left(\frac{(n - U)^2}{n} + 1\right).$$

(The error is allowed only when the distance is 0.)

▶ **Corollary 4.** *The one-sided error public-coin communication complexity R^0 of $\mathsf{GHD}_{L,U}$ on n-bit strings is at least*

$$\Omega\left(\frac{L^2}{U}+1\right).$$

(The error is allowed only when the distance is at least U.)

Thus our results determine the one-sided public-coin communication complexity of $\mathsf{GHD}_{L,U}$ (up to a factor $O(\log L)$) in the case when the parties are allowed to err only for input pairs at distance at least U. If the parties are allowed to err only for input pairs at distance at most L, then the one-sided public-coin communication complexity of $\mathsf{GHD}_{L,U}$ is $(n-U)^2/(n-L)$ up to a factor of $O(\log(n-U))$.

2 Preliminaries

2.1 Communication Complexity

Let $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ be a Boolean function.

▶ **Definition 5.** A deterministic communication protocol is a rooted binary tree, in which each non-leaf vertex is associated either with Alice or with Bob and each leaf is labeled by 0 or 1. Each non-leaf vertex v, associated with Alice, is assigned a function $f_v : \mathcal{X} \to \{0,1\}$ and each non-leaf vertex u, associated with Bob, is assigned a function $g_u : \mathcal{Y} \to \{0,1\}$. For each non-leaf vertex one of its out-going edges is labeled by 0 and other one is labeled by 1.

A computation according to a deterministic protocol runs as follows. Alice is given $x \in \mathcal{X}$, Bob is given $y \in \mathcal{Y}$. They start at the root of tree. If they are in a non-leaf vertex v, associated with Alice, Alice sends $f_v(x)$ to Bob and they move to the son of v by the edge labeled by $f_v(x)$. If they are in a non-leaf vertex, associated with Bob, they act in a similar same way, however this time it is Bob who sends a bit to Alice. When they reach a leaf, they output the bit which labels this leaf.

▶ **Definition 6.** Communication complexity of a deterministic protocol π, denoted by $CC(\pi)$, is defined as the depth of the corresponding binary tree.

Randomized protocols with shared randomness (aka public-coin protocols) can be defined as follows:

▶ **Definition 7.** A public-coin communication protocol is a probability distribution over deterministic protocols. Communication complexity of a public-coin protocol τ, denoted by $CC(\tau)$, is defined as $\max_{\pi} CC(\pi)$, where π is taken over the deterministic protocols from the support of τ (recall that τ is a distribution).

Given a public-coin protocol τ, Alice and Bob choose the deterministic protocol to be executed according to the distribution, defined by τ.

▶ **Definition 8.** We say that a public-coin protocol computes a partial function f with error probability ε, if for every pair of inputs (x,y) in the domain of f with probability at least $1-\varepsilon$ that protocol outputs $f(x,y)$. Randomized communication complexity of f is defined as

$$R_\varepsilon(f) = \min_{\pi} CC(\pi),$$

where minimum is over all protocols that compute f with error probability ε.

A deterministic *simultaneous* protocol τ is a triple $\langle \phi, \psi, \theta \rangle$ where

$$\phi : \mathcal{X} \to \{0,1\}^{c_1}, \qquad \psi : \mathcal{Y} \to \{0,1\}^{c_2},$$

$$\theta : \{0,1\}^{c_1} \times \{0,1\}^{c_2} \to \{0,1\}.$$

The communication cost of τ is $c_1 + c_2$. A public-coin simultaneous protocol π is a probability distribution over deterministic simultaneous protocols. Communication cost of π is the maximal possible communication cost of τ, where τ is a deterministic simultaneous protocol taken from the support of π.

Assume that Alice is given $x \in \mathcal{X}$ and Bob is given $y \in \mathcal{Y}$. The output of a public-coin simultaneous protocol π on (x, y) is a random variable defined as follows. Sample a deterministic simultaneous protocol $\tau = \langle \phi, \psi, \theta \rangle$ according to π. Output $\theta(\phi(x), \psi(y))$.

If for $i \in \{0,1\}$ we require that the protocol never errs on inputs from $f^{-1}(i)$, then the corresponding notion is called "randomized one-sided error communication complexity" and is denoted by $R^i_\varepsilon(f)$.

The Gap Hamming Distance problem is the problem of computing the following partial function:

$$\mathsf{GHD}^n_{L,U}(x,y) = \begin{cases} 0 & d(x,y) \leqslant L, \\ 1 & d(x,y) \geqslant U, \\ \text{undefined} & L < d(x,y) < U, \end{cases} \qquad \text{for } x, y \in \{0,1\}^n.$$

2.2 Hamming Space

▶ **Definition 9.** The function

$$h(x) = x \log_2 \frac{1}{x} + (1-x) \log_2 \frac{1}{1-x}$$

is called the *Shannon function*.

For any $B \subset \{0,1\}^n$ define $\mathsf{diam}(B) = \max\limits_{x,y \in B} d(x,y)$. Let $V_2(n,r)$ denote the size of Hamming ball of radius r, that is $V_2(n,r) = \binom{n}{0} + \ldots + \binom{n}{r}$.

We will use the following well-known facts about the size of Hamming balls.

▶ **Proposition 10** ([7]). *If $r \leqslant \frac{n}{2}$, then $V_2(n,r) \leqslant 2^{h\left(\frac{r}{n}\right)n}$.*

▶ **Proposition 11** ([7]). *If $B \subset \{0,1\}^n$, r is natural, $\mathsf{diam}(B) \leqslant 2r$ and $n \geqslant 2r+1$, then*

$$|B| \leqslant V_2(n,r).$$

Propositions 11, 10 and the fact that $h'(1/2) = 0, h''(1/2) < 0$ easily imply the following

▶ **Lemma 12.** *Assume that $r < n/2$. Then the cardinality of every set $B \subset \{0,1\}^n$ with $\mathsf{diam}(B) \leqslant 2r$ is at most $2^{n(1-c(1-(2r/n))^2)}$ for some absolute positive constant c.*

2.3 Probability Theory

▶ **Definition 13** (Probability distributions). Let $B(n,p)$ denote the binomial distribution with parameters $n \in \mathbb{N}$ and $p \in (0,1)$. For every natural n let S_n denote the one-dimensional random walk with n steps. More specifically, let S_n be equal to

$$S_n = X_1 + \ldots + X_n,$$

where $X_1, \ldots, X_n$ are independent random variables taking values in $\{-1, 1\}$, such that for each i the following holds: $\Pr[X_i = 1] = \Pr[X_i = -1] = \frac{1}{2}$.

3 The upper bound

The protocol for Theorem 2 is a combination of three different protocols. The most important of them solves $\mathsf{GHD}_{L,U}$ with one sided error in the case when U/L exceeds some constant. Its communication length is $O((L^2/U+1)\log n)$. We call that protocol the "Triangle Inequality Protocol", because it uses the triangle inequality for Hamming distance.

3.1 The Triangle Inequality Protocol

The following Lemma is the standard fact of Probability Theory:

▶ **Lemma 14.** *There exists a positive constant $\alpha > 0$ such that for every m it holds that*

$$\Pr[S_m \geqslant \sqrt{m}] \geqslant \alpha,$$

where S_m denotes one-dimensional random walk with m steps, i.e, S_m is equal to the sum of m independent random variables, each taking the values 1 and -1 with probabilities 1/2.

Everywhere below α stands for the constant from Lemma 14.

Let $x, y \in \{0,1\}^n$ denote Alice's and Bob's input strings, respectively. The parties set $b = \lceil CL^2/U + 1 \rceil$, where $C = 360/\alpha^2$. Then they use public coins to sample a function $\chi : \{1,2,\ldots,n\} \to \{1,2,\ldots,b\}$ uniformly at random. They use χ to divide x and y into b blocks

$$x^1,\ldots,x^b, \qquad y^1,\ldots,y^b.$$

The block x^j consists of all bits x_i of x such that $\chi(i) = j$. Similarly, y^j consists of all bits y_i with $\chi(i) = j$. The order in which bits of jth block are arranged is not important, the parties care only that they use the same order.

Then they use public coins to sample b random strings $r^1,\ldots,r^b$ of the same lengths, as $x^1,\ldots,x^b$ and $y^1,\ldots,y^b$. Alice then sends b numbers to Charlie:

$$d(x^1,r^1),\ldots,d(x^b,r^b).$$

In turn, Bob sends

$$d(y^1,r^1),\ldots,d(y^b,r^b).$$

Then Charlie computes the sum

$$T = \sum_{j=1}^{b} \left| d(x^j,r^j) - d(y^j,r^j) \right|.$$

If $T \leqslant L$, Charlie outputs 0. Otherwise he outputs 1.

If $d(x,y) \leqslant L$, then the protocol always outputs 0. Indeed, since Hamming distance satisfies the triangle inequality, we have that

$$T = \sum_{j=1}^{b} \left| d(x^j,r^j) - d(y^j,r^j) \right| \leqslant \sum_{j=1}^{b} d(x^j,y^j) = d(x,y) \leqslant L.$$

Thus this protocol has a one-sided error: it can err only if $d(x,y) \geqslant U$. Now we will estimate the probability of error in the case when $d(x,y) \geqslant U$.

▶ **Lemma 15.** *Assume that $U \geqslant 2b$. Then the protocol for the input pair x, y at distance at least U outputs 1 with some positive constant probability (more specifically, with probability at least $\alpha/6$).*

Proof. Assume that U, L, x, y satisfy the assumption of the lemma. Fix $j = 1, \ldots, b$. First we have to understand what is the distribution of the random variable $|d(x^j, r^j) - d(y^j, r^j)|$. By construction Alice and Bob choose a random function χ that governs the partition of x, y into blocks. For each i such that $x_i \neq y_i$ the probability that x_i, y_i land into the block with number j is $1/b$. Hence the random variable $d(x^j, y^j)$ has binomial distribution $B(d(x, y), 1/b)$ with parameters $d(x, y)$ and $1/b$, i.e., the probability of the event $d(x^j, y^j) = k$ equals

$$\binom{d(x,y)}{k}(1/b)^k(1-1/b)^{d(x,y)-k}.$$

The average value of $d(x^j, y^j)$ is thus equal to $d(x, y)/b$.

Once x^j, y^j are determined, Alice and Bob sample r^j. The value $d(x^j, r^j) - d(y^j, r^j)$ can be represented as the sum of $|x^j| = |y^j|$ terms where each term corresponds to a number i with $\chi(i) = j$. If $x_i = y_i$ then the term is 0. Otherwise it is either -1 or 1 depending on whether the respective bit of r^j is equal to x_i or to y_i. Thus for every fixed partition into blocks the value $|d(x^j, r^j) - d(y^j, r^j)|$ is distributed as the distance from origin in the random walk along the line with $d(x^j, y^j)$ independent steps where each step is 1 with probability $1/2$ and -1 with the same probability.

To finish the proof we will use the following fact about binomial distribution.

▶ **Lemma 16.** *If X is distributed according to the binomial distribution $B(n, p)$ and $pn \geqslant 2$, then*

$$\Pr\left[X > \frac{pn}{10}\right] \geqslant \frac{1}{3}.$$

Proof of Lemma 16. The expectation and variation of X are given by:

$$\mathrm{E}X = pn, \qquad \mathrm{Var}X = p(1-p)n \leqslant pn.$$

Hence by Chebyshev inequality we get

$$\Pr\left[X \leqslant \frac{pn}{10}\right] \leqslant \frac{\mathrm{Var}X}{\left(pn \cdot \left(1 - \frac{1}{10}\right)\right)^2} \leqslant \frac{\frac{100}{81}}{pn} \leqslant \frac{100}{162} \leqslant \frac{2}{3}.$$ ◀

Recall that the random variable $d(x^j, y^j)$ has binomial distribution $B(d(x, y), 1/b)$ and we assume that $d(x, y)/b \geqslant U/b \geqslant 2$. Hence by Lemma 16 with probability at least $1/3$ we have $d(x^j, y^j) \geqslant d(x, y)/10b$.

Fix any partition into blocks such that $d(x^j, y^j) \geqslant d(x, y)/10b$. By Lemma 14 with probability at least α we have

$$|d(x^j, r^j) - d(y^j, r^j)| \geqslant \sqrt{d(x^j, y^j)} \geqslant \sqrt{d(x, y)/10b}.$$

We have proved that for every fixed j with probability at least $\alpha/3$ we have $|d(x^j, r^j) - d(y^j, r^j)| \geqslant \sqrt{d(x, y)/10b}$. A simple averaging argument shows that with probability at least $\alpha/6$ the fraction of j that satisfy this inequality is bigger than $\alpha/6$. Indeed, let the random variable θ denote the fraction of j that satisfy this inequality. Its average is at least $\alpha/3$. On the other hand, we can upperbound its average by the sum

$$\Pr[\theta > \alpha/6] \cdot 1 + \Pr[\theta \leqslant \alpha/6] \cdot (\alpha/6) \leqslant \Pr[\theta > \alpha/6] + \alpha/6.$$

Thus with probability $\alpha/6$ we have

$$\sum_{j=1}^{b} |d(x^j, r^j) - d(y^j, r^j)| > (\alpha/6)b\sqrt{d(x,y)/10b} = (\alpha/6)\sqrt{b \cdot d(x,y)/10}.$$

Recall that $b = \lceil CL^2/U + 1 \rceil$, where $C = 360/\alpha^2$, and $d(x,y) \geqslant U$. So the right hand side of the last displayed inequality is strictly larger than L. ◀

If the ratio U/L is larger than a certain constant then the protocol solves $\mathsf{GHD}_{L,U}$ with constant one-sided error-probability. One can verify that the assumption $U \geqslant 2b$ of Lemma 15 is met for all $U \geqslant 2CL + 4$.

Recall that the communication length of the protocol is $O((L^2/U + 1)\log n)$. Now we need a protocol with the same communication length for L, U such that $U \leqslant 2CL + 3$. Notice that in this case the upper bound $O((L^2/U)\log n)$ for communication boils down to $O(L \log n)$. A protocol with such performance was constructed in [8].

3.2 The protocol of [8]

For the reader's convenience and to stress that the protocol from [8] has one-sided error we give here its full description.

Here $\oplus$ stands for the bit-wise XOR over n-bit vectors and $\langle \cdot, \cdot \rangle : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ denotes the inner product over $\mathbb{F}_2$:

$$\langle a, b \rangle = \sum_{i=1}^{n} a_i b_i \pmod 2.$$

Let $x, y \in \{0,1\}^n$ denote Alice's and Bob's input strings, respectively. They use public coins to sample N vectors

$$R_1, \ldots, R_N \in \{0,1\}^n$$

independently uniformly at random. Alice sends $\langle x, R_1 \rangle, \ldots, \langle x, R_N \rangle$ to Charlie. Bob does the same with y. If there is $f \in \{0,1\}^n$ of Hamming weight at most L such that:

$$\langle x \oplus f, R_1 \rangle = \langle y, R_1 \rangle, \ldots, \langle x \oplus f, R_N \rangle = \langle y, R_N \rangle, \tag{1}$$

then Charlie outputs 0. Otherwise Charlie outputs 1.

Such protocol costs $O(N)$ bits. If $d(x,y) \leqslant L$, then the protocol outputs 0 with probability 1. Indeed, $f = x \oplus y$ (which is of Hamming weight at most L in this case) satisfies (1).

Now assume that $d(x,y) > L$. Then any $f \in \{0,1\}^n$ of Hamming weight at most L satisfies (1) only with probability at most 2^{-N} (because $x + f \neq y$). Hence the error probability of the protocol is at most $V_2(n,L) \cdot 2^{-N}$ in this case. Here $V_2(n,L)$ is the size of Hamming ball of radius L. As $V_2(n,L) \leqslant (n+1)^L$, it is enough to take $N = O(L \log n)$.

3.3 The simplified version of the Triangle Inequality Protocol

Thus for all L, U we have a public-coin simultaneous protocol with communication length $O((L^2/U + 1)\log n)$ to solve $\mathsf{GHD}_{L,U}$ with constant one-sided error probability. To replace $\log n$ factor by $\log L$ factor we will need the following public-coin simultaneous protocol with communication length $O(\log L)$ to solve $\mathsf{GHD}_{L,\,(4L+2+N_0)^4}$ with constant one-sided error probability. Here N_0 is a constant from the following Lemma.

▶ **Lemma 17.** *There is a positive integer N_0 and a positive real c such that the following holds. Assume that m and N are positive integers and $N \geqslant \max\{N_0, m^2\}$. Consider N independent random variables $Z_1, \ldots, Z_N$, where each variable takes the values 0 and 1 with probabilities 1/2. Then for every $i \in \{0, 1, \ldots, m-1\}$ it holds that:*

$$\Pr[Z_1 + \ldots + Z_N = i \pmod m] \geqslant \frac{c}{m}.$$

(The proof of this Lemma will be given in the end of this subsection). Notice that $O((L^2/U + 1)\log L)$ becomes just $O(\log L)$ for $U = (4L + 2 + N_0)^4$.

The protocol. The parties use public coins to sample a vector $r \in \{0,1\}^n$ uniformly at random. Alice and Bob compute the distance from r to their input strings. If $d(x, y) \leqslant L$, then by Triangle Inequality we have $|d(x, r) - d(y, r)| \leqslant d(x, y) \leqslant L$. On the other hand, assume that $d(x, y) \geqslant (4L + 2 + N_0)^4$. From Lemma 14 it follows that in this case with constant positive probability we have $|d(x, r) - d(y, r)| \geqslant \sqrt{d(x, y)} > (4L + 2)^2 + N_0^2$.

Consider the following auxiliary problem. Alice holds a number $a \in \{0, 1, \ldots, n\}$, Bob holds a number $b \in \{0, 1, \ldots n\}$ and it is promised that either $|a - b| \leqslant L$ or $|a - b| > (4L + 2)^2 + N_0^2$. They want to know whether the first or the second inequality is true. As the previous paragraph shows, if there is a public-coin SMP protocol with communication length $O(\log L)$, which always outputs 0 when $|a - b| \leqslant L$ and which with constant positive probability outputs 1 when $|a - b| > (4L + 2)^2 + N_0^2$, then we are done.

Define $m = 4L + 2$. Use public coins to sample $n + 1$ independent random variables

$$Z_0, Z_1, Z_2, \ldots, Z_n,$$

where each variable takes the values 0 and 1 with probabilities 1/2.

Alice sends $\sum_{i=0}^{a} Z_i \pmod m$ to Charlie, Bob sends $\sum_{i=0}^{b} Z_i \pmod m$ to Charlie. This takes only $O(\log m) = O(\log L)$ bits. Let (s, t) be any pair of integers satisfying the following three conditions:

$$s \equiv \sum_{i=0}^{a} Z_i \pmod m \tag{2}$$

$$t \equiv \sum_{i=0}^{b} Z_i \pmod m \tag{3}$$

$$|s - t| = \min\left\{|s' - t'| : s' \equiv \sum_{i=0}^{a} Z_i \pmod m, t' \equiv \sum_{i=0}^{b} Z_i \pmod m\right\}. \tag{4}$$

Obviously, knowing $\sum_{i=0}^{a} Z_i \pmod m$, $\sum_{i=0}^{b} Z_i \pmod m$, Charlie is able to find (s, t) satisfying these tree conditions. He then simply checks whether $|s - t| \leqslant L$. If this is the case, he outputs 0. Otherwise he outputs 1.

Once again, the protocol communicates only $O(\log L)$ bits, as required. Further, it is easy to see that the protocol has one-sided error. Indeed, assume that $|a - b| \leqslant L$. Note that a pair $(\sum_{i=0}^{a} Z_i, \sum_{i=0}^{b} Z_i)$ satisfies (2) and (3) . Hence $|s - t| \leqslant \left|\sum_{i=0}^{a} Z_i - \sum_{i=0}^{b} Z_i\right| \leqslant |a - b| \leqslant L$.

Now, let's consider the case when $|a-b| > (4L+2)^2+N_0^2$. Assume without loss of generality that $a < b$. Let E be the event that there is no $r \in [-L, L]$ such that $Z_{a+1} + \ldots + Z_b \equiv r$

(mod m). Let us verify that E implies that $|s-t| > L$ (which means that Charlie outputs 1). Indeed, observe that

$$t - s \equiv \sum_{i=0}^{b} Z_i - \sum_{i=0}^{a} Z_i \equiv Z_{a+1} + \ldots + Z_b \pmod{m},$$

but if $|s-t| \leqslant L$, this contradicts E.

It only remains to show that E happens with constant positive probability. This follows from Lemma 17. Namely, this lemma implies that $\Pr[E] \geqslant \frac{c(m-2L-1)}{m} = c/2$. Parameters are chosen in such a way that restrictions of Lemma 17 are satisfied:

$$b - a > (4L+2)^2 + N_0^2 \geqslant (\max\{N_0, 4L+2\})^2 \geqslant \max\{N_0, m^2\}.$$

Proof of Lemma 17. Take N_0 to be the first natural satisfying the following condition: there exists $d > 0$ such that for all $N \geqslant N_0$ and for every k between $N/2 - \sqrt{N}$ and $N/2 + \sqrt{N}$ the following holds:

$$\Pr[Z_1 + \ldots + Z_N = k] = \binom{N}{k} 2^{-N} \geqslant \frac{d}{\sqrt{N}}.$$

The existence of such N_0, d is just a standard corollary of the Stirling formula, applied to $\binom{N}{k}$.

Now let us show that for all $m > 0$, $N \geqslant m^2$ and $i \in \{0, 1, \ldots, m-1\}$ the number of k between $N/2 - \sqrt{N}$ and $N/2 + \sqrt{N}$ such that $k \equiv i \pmod{m}$ is at least $\frac{\sqrt{N}}{m}$. The number of such k is equal to the number of $r \in \mathbb{Z}$ satisfying:

$$N/2 - \sqrt{N} \leqslant mr + i \leqslant N/2 + \sqrt{N},$$

This number is at least

$$\left\lfloor \frac{N/2 + \sqrt{N} - i}{m} \right\rfloor - \left\lceil \frac{N/2 - \sqrt{N} - i}{m} \right\rceil + 1 \geqslant \frac{2\sqrt{N}}{m} - 1.$$

Provided $N \geqslant m^2$, the last expression is at least $\frac{\sqrt{N}}{m}$.

Set $c = d$ and observe that for all m, N such that $m > 0$ and $N \geqslant \max\{N_0, m^2\}$ and for every $i \in \{0, 1, \ldots, m-1\}$ it holds that

$$\Pr[Z_1 + \ldots + Z_N \equiv i \pmod{m}] \geqslant \frac{\sqrt{N}}{m} \cdot \frac{d}{\sqrt{N}} = \frac{c}{m}.$$

◀

3.4 The final protocol for Theorem 2

The protocol. *Step 1.* Alice and Bob first run the Simplified Triangle Inequality Protocol from the previous subsection. If that protocol outputs 1 they output 1 and halt. Otherwise they proceed to Step 2.

Step 2. They divide x and y into $w = 2(4L + 2 + N_0)^8$ blocks randomly (as in the construction of the Triangle Inequality Protocol). Let

$$x^1, \ldots, x^w, \qquad y^1, \ldots, y^w$$

denote the resulting blocks. Let u_i be the XOR of all bits from x^i and let v_i be the XOR of all bits from y^i. Alice privately computes $u_1, \dots, u_w$ and sets $u = u_1 \dots u_w$. Bob privately computes $v_1, \dots, v_w$ and sets $v = v_1 \dots v_w$.

Recall that we have a protocol (a combination of the Triangle Inequality Protocol and the protocol of [8]) with communication length $O((L^2/U + 1) \log w) = O((L^2/U + 1) \log L)$ to solve $\mathsf{GHD}_{L,U}$ on w-bit strings with constant positive one-sided error probability.

Alice and Bob run this protocol for input pair (u, v) (and not (x, y)). They output the result of this run.

The communication length of the constructed protocol is $O((L^2/U + 1) \log L)$. We have to show that it has one-sided constant error probability.

If $d(x, y) \leqslant L$ then the run of the Simplified Triangle Inequality Protocol will output 0 with probability 1. Thus they proceed to Step 2. The distance between u and v does not exceed the distance between x and y and hence is at most L. Thus the run of the second protocol also outputs 0 with probability 1.

Assume that $d(x, y) \geqslant U$. If $d(x, y) \geqslant (4L + 2 + N_0)^4$, then the Simplified Triangle Inequality Protocol outputs 1 with positive constant probability, they output 1 and halt.

Assume that $U \leqslant d(x, y) < (4L + 2 + N_0)^4$. We claim that in this case with constant positive probability we have $d(u, v) = d(x, y)$. Indeed, consider any two positions in which x and y differ. Those positions land into the same block with probability $\frac{1}{w}$. By union bound, with probability at least

$$1 - \frac{d(x, y)^2}{w} \geqslant 1 - \frac{(4L + 2 + N_0)^8}{2(4L + 2 + N_0)^8} = 0.5$$

all the positions in which x and y differ land in different blocks. The latter means that for all i the blocks x^i and y^i differ in at most 1 position and hence $d(u, v) = d(x, y)$. Thus with probability at least $1/2$ we have $d(u, v) \geqslant U$ and Alice and Bob output 1 with positive constant probability on the second step.

4 The lower bound

In this section we prove Theorem 3.

Proof of Theorem 3. Let τ be a protocol witnessing $R^1_{\frac{1}{2}}(\mathsf{GHD}_{0,U})$. Then the following hold:

- for each $x \in \{0, 1\}^n$ the protocol τ for input (x, x) outputs 0 with probability at least $\frac{1}{2}$;
- for all $x, y \in \{0, 1\}^n$ with $d(x, y) \geqslant U$ the protocol τ always outputs 1.

By the standard averaging argument due to von Neumann there is a deterministic protocol π such that

- the communication complexity of π is at most $R^1_{\frac{1}{2}}(\mathsf{GHD}_{0,U})$;
- π outputs 0 for at least half of diagonal input pairs (x, x);
- π outputs 1 for all inputs pairs at Hamming distance at least U.

Consider any 0-leaf of π and the corresponding rectangle $R = A \times B \subset \{0, 1\}^n \times \{0, 1\}^n$. The number of diagonal pairs from R is equal to $|A \cap B|$. Diameter of $A \cap B$ must be less than U. Indeed, if there are $x, y \in A \cap B$ such that $d(x, y) \geqslant U$, then π outputs 0 for input pair (x, y).

It turns out that the largest set of diameter $2r < n$ is the Hamming ball of radius r and the diameter of the latter is at most $2^{n(1-c(1-2r/n)^2)}$ for some positive constant c (Lemma 12).

Let $r = \lfloor U/2 \rfloor$. For $U = n$ the lower bound in Theorem 3 is constant and thus the statement is obvious. Therefore we may assume that $U < n$ and hence $r < n/2$. The diameter of $A \cap B$ is at most $2r$ (recall that the diameter of $A \cap B$ is strictly less than U). By Lemma 12 we have

$$|A \cap B| \leqslant 2^{n(1-c(1-2r/n)^2)} \leqslant 2^{n(1-c(1-U/n)^2)}.$$

We have shown that if R is the rectangle corresponding to a 0-leaf of π, then R covers at most $2^{n(1-c(1-U/n)^2)}$ diagonal pairs. As the total number of diagonal pairs covered by 0-leaves of π is at least 2^{n-1}, the number of 0-leaves in π is at least $2^{cn(1-U/n)^2-1}$. Thus we have

$$R^1_{\frac{1}{2}}(\mathsf{GHD}_{0,U}) \geqslant c \cdot \frac{(n-U)^2}{n} - 1. \tag{5}$$

Obviously we also have

$$R^1_{\frac{1}{2}}(\mathsf{GHD}_{0,U}) \geqslant 1. \tag{6}$$

From inequalities (5) and (6) we can easily deduce that

$$R^1_{\frac{1}{2}}(\mathsf{GHD}_{0,U}) \geqslant \Omega\left(\frac{(n-U)^2}{n} + 1\right)$$

(for example, we can add these inequalities with appropriate positive weights). ◀

References

1. Eric Blais, Joshua Brody, and Kevin Matulef. Property testing lower bounds via communication complexity. *Computational Complexity*, 21(2):311–358, 2012.
2. Joshua Brody and Amit Chakrabarti. A multi-round communication lower bound for gap hamming and some consequences. In *Computational Complexity, 2009. CCC'09. 24th Annual IEEE Conference on*, pages 358–368. IEEE, 2009.
3. Joshua Brody, Amit Chakrabarti, Oded Regev, Thomas Vidick, and Ronald De Wolf. Better gap-hamming lower bounds via better round elimination. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 476–489. Springer, 2010.
4. Joshua Brody and David P Woodruff. Streaming algorithms with one-sided estimation. In *APPROX-RANDOM*, pages 436–447. Springer, 2011.
5. Harry Buhrman, Richard Cleve, and Avi Wigderson. Quantum vs. classical communication and computation. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 63–68. ACM, 1998.
6. Amit Chakrabarti and Oded Regev. An optimal lower bound on the communication complexity of gap-hamming-distance. *SIAM Journal on Computing*, 41(5):1299–1317, 2012.
7. Gérard Cohen, Iiro Honkala, Simon Litsyn, and Antoine Lobstein. *Covering codes*, volume 54. Elsevier, 1997.
8. Dmitry Gavinsky, Julia Kempe, and Ronald de Wolf. Quantum communication cannot simulate a public coin. *arXiv preprint quant-ph/0411051*, 2004.
9. Wei Huang, Yaoyun Shi, Shengyu Zhang, and Yufan Zhu. The communication complexity of the hamming distance problem. *Information Processing Letters*, 99(4):149–153, 2006.
10. Thathachar S Jayram, Ravi Kumar, and D Sivakumar. The one-way communication complexity of hamming distance. *Theory of Computing*, 4(1):129–135, 2008.

11 Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30(2):457–474, 2000.

12 Alexander A Sherstov. The communication complexity of gap hamming distance. *Theory of Computing*, 8(1):197–208, 2012.

13 Thomas Vidick. A concentration inequality for the overlap of a vector on a large set, with application to the communication complexity of the gap-hamming-distance problem. *Chicago Journal of Theoretical Computer Science*, 1, 2012.

14 David Woodruff. Optimal space lower bounds for all frequency moments. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 167–175. Society for Industrial and Applied Mathematics, 2004.

15 Andrew Chi-Chih Yao. On the power of quantum fingerprinting. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 77–81. ACM, 2003.

Online Maximum Matching with Recourse

Spyros Angelopoulos

Sorbonne Université, CNRS, Laboratoire d'informatique de Paris 6, LIP6, F-75252 Paris, France
Spyros.Angelopoulos@lip6.fr
https://orcid.org/0000-0001-9819-9158

Christoph Dürr

Sorbonne Université, CNRS, Laboratoire d'informatique de Paris 6, LIP6, F-75252 Paris, France
Christoph.Durr@lip6.fr
https://orcid.org/0000-0001-8103-5333

Shendan Jin

Sorbonne Université, CNRS, Laboratoire d'informatique de Paris 6, LIP6, F-75252 Paris, France
Shendan.Jin@lip6.fr
https://orcid.org/0000-0003-1218-9085

Abstract

We study the online maximum matching problem in a model in which the edges are associated with a known recourse parameter k. An online algorithm for this problem has to maintain a valid matching while edges of the underlying graph are presented one after the other. At any moment the algorithm can decide to include an edge into the matching or to exclude it, under the restriction that at most k such actions per edge take place, where k is typically a small constant. This problem was introduced and studied in the context of general online packing problems with recourse by Avitabile *et al.* [1], whereas the special case $k = 2$ was studied by Boyar *et al.* [3].

In the first part of this paper, we consider the *edge arrival* model, in which an arriving edge never disappears from the graph. Here, we first show an improved analysis on the performance of the algorithm AMP given in [1], by exploiting the structure of the matching problem. In addition, we extend the result of [3] and show that the greedy algorithm has competitive ratio $3/2$ for every even k and ratio 2 for every odd k. Moreover, we present and analyze an improvement of the greedy algorithm which we call L-Greedy, and we show that for small values of k it outperforms the algorithm of [1]. In terms of lower bounds, we show that no deterministic algorithm better than $1 + 1/(k-1)$ exists, improving upon the lower bound of $1 + 1/k$ shown in [1].

The second part of the paper is devoted to the *edge arrival/departure model*, which is the fully dynamic variant of online matching with recourse. The analysis of L-Greedy and AMP carry through in this model; moreover we show a lower bound of $\frac{k^2-3k+6}{k^2-4k+7}$ for all even $k \geq 4$. For $k \in \{2, 3\}$, the competitive ratio is $3/2$.

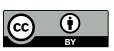
2012 ACM Subject Classification Mathematics of computing → Graph algorithms, Theory of computation → Online algorithms

Keywords and phrases Competitive ratio, maximum cardinality matching, recourse

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.8

Related Version A full version of this paper is available at `https://arxiv.org/abs/1801.03462`.

Funding Supported by ANR OATA, DIM RFSI DACM and Labex Mathématique Hadamard.



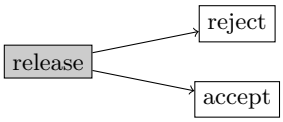
1 Introduction

In the standard framework of online computation, the input to the algorithm is revealed incrementally, i.e., request by request. For each such requested input item, the online algorithm must make a decision that is typically *irrevocable*, in the sense that the algorithm commits, in a permanent manner, to the decision associated with the request. More precisely, the algorithm may not alter any previously made decisions while considering later requests. This rather stringent constraint is meant to capture what informally can be described as "the past cannot be undone"; equally significantly, it is at the heart of adversarial arguments that can be used to argue that the competitive ratio of a given online problem cannot be improved beyond a certain bound.

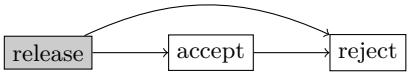
Nevertheless, there are real-life applications in which some (limited) rearrangement of the online solution during the execution of the algorithm may be doable, or even requisite. For instance, online call admission protocols may sporadically reconfigure the virtual paths assigned in the network. For a different example, in online scheduling (or resource allocation) problems, it may be permissible for a job to be transfered to a processor other than the one specified by the original decision associated with the job. Clearly, a trade-off is to be found between the guaranteed competitive ratio and the cost of re-optimizing the current solution. Different approaches to this objective have been considered. One such approach has studied the minimum total re-optimization cost required in order to maintain an optimal solution, see Bernstein *et al.* [2]. Another approach has focused on the best achievable competitive ratio when there is some bound on the allowed re-optimization, which has been first studied by Avitabile *et al.* [1], and is the main model we consider in this paper.

More specifically, we study the *online maximum cardinality matching* problem, in which the goal is to maintain a vertex disjoint edge set of maximum cardinality for a given graph. Two different online models have been studied in the past. In the *vertex arrival model*, vertices arrive in online fashion, revealing, at the same time, the edges incident to previously arrived vertices. This model has mainly been considered for bipartite graphs, with left side vertices arriving online, and right side vertices being initially known (see the survey [16]). In the *edge arrival model* the edges arrive online in arbitrary order, revealing at the same time incident vertices. We emphasize that in this work we consider the maximum cardinality matching problem; some previous work (with or without recourse) has considered the generalized *weighted matching* problem, in which each edge has a weight and the objective is to maximize the weight of matched edges.

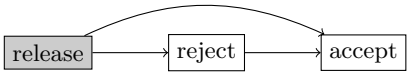
In the standard model, every edge constitutes a request and has to be immediately either accepted in the matching, or rejected. To quantify the impact of recourse, several models have been proposed that relax the irrevocable nature of a decision. In the *late reject* model [3], which is also called the *preemptive model* [5], an edge can be accepted only upon its arrival, but can be later rejected. In the *edge-bounded recourse* model, introduced in [1], the algorithm can switch between accepting and rejecting an edge that has already appeared, but is allowed up to k such modifications per edge. We emphasize that the initial default state of an edge is *rejected*, and therefore rejecting a newly arriving edge does not count as decision modification. However, accepting an edge or rejecting a previously accepted edge does count as a decision modification. For concreteness, we call this problem the *online maximum cardinality matching problem with edge-bounded recourse*. Boyar *et al.* [3] refer to this model for $k = 1$ as the *late accept* model, and for $k = 2$ as the *late accept/reject* model. Clearly, the competitive ratio is monotone in k, and our objective is to quantify this dependency. Figure 1 illustrates the algorithm's actions under the different models.



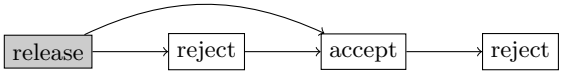
Standard model.



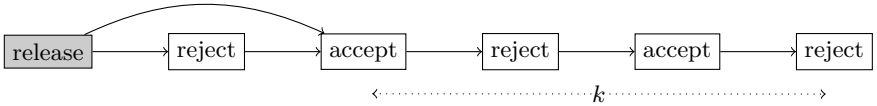
Late reject model, also called free disposal or preemptive model.



Late accept model, also called edge 1-bounded recourse model.



Late accept/reject model, also called edge 2-bounded recourse model.



Edge-bounded recourse model with general parameter k.

Figure 1 Illustration of the actions of an online matching algorithm under the different edge-arrival models with recourse.

In this work we also introduce and study the setting in which edges may undergo both arrivals and departures. In other words, edges may not only arrive (in the form of a request), but may also disappear adversarially (subsequently to their appearance). This setting is motivated by similar models that have been studied in the context of the online Steiner tree problem [9]. We call this problem the online maximum matching problem under the *edge arrival/departure model with edge-bounded recourse*. For this problem, we further distinguish two models concerning the edge departures. In the *full departure* model, the adversary is allowed to delete any edge in the graph, and thus also any edge that may have been provisionally accepted by the online algorithm. We show that this model is quite restrictive, since it yields excessive power to the adversary. We thus also study the *limited departure* model, in which the adversary may delete only edges not currently accepted by the online algorithm.

The limited departure setting can also model some natural applications related to resource allocation. For instance, consider a bipartite graph representing compatibility between tasks and workers, and that we seek to maximize the number of tasks assigned to workers. The compatibility of tasks and worker skills can change over time. Then the limited departure model stipulates that if worker w remains assigned to task t by the online algorithm, then w does not lose his qualification for t, in the sense that the worker has a continual occupation with the said task and maintains the required skills for the task. However, once the online algorithm decides to remove worker w from task t (i.e., the online algorithm provisionally rejects edge (w, t)), then the worker might lose his qualification for the task over time.

An online algorithm ALG for a maximization problem is said to be c-competitive if there is a constant d such that $\mathrm{ALG}(\sigma) \geq \mathrm{OPT}(\sigma)/c - d$, for all request sequences σ. Here $\mathrm{ALG}(\sigma)$ stands for the objective value of the solution produced by ALG on σ, while $\mathrm{OPT}(\sigma)$ stands for the value of the optimal solution. If $d = 0$, the algorithm is called *strictly* c-competitive. Note that some previous work on the matching problem has used the reciprocal ratio. The smallest c for which an online algorithm ALG is c-competitive is called the *competitive ratio* of ALG. The strict competitive ratio is defined similarly. If it so happens that this minimum value does not exist, the competitive ratio is actually defined by the corresponding infimum. In this setting an *upper bound* on the (strict or not) competitive ratio establishes the performance guarantee of an online algorithm, whereas a *lower bound* is a negative result.

Both upper and lower bounds in this work are shown for the strict competitive ratio. This implies that the upper bounds carry over to the more general definition, but this generalization does not necessarily hold for the lower bounds. We emphasize, however, that the known lower bounds for edge-bounded recourse problems in [1, 3] are likewise expressed in terms of the strict competitive ratio. This is due, perhaps, to difficulties in applying techniques that extend the lower bounds to the standard definition of the competitive ratio that are inherent to the recourse setting, and which do not arise in the traditional online framework of irrevocable decisions. Specifically, it is not obvious how to use techniques based on multiple copies of an adversarial instance in order to lower-bound the performance of any online algorithm, although this may be possible for specific online algorithms. For convenience, we will henceforth refer to the strict competitive ratio as simply the "competitive ratio".

Related work

Several online optimization problems have been studied under the recourse setting. The broad objective is to quantify the trade-off between the competitive ratio and a measure on the modifications allowed on the solution. Some representative examples include online problems such as minimum spanning trees and TSP [15], Steiner trees [9, 8], knapsack problems [11, 12], assignment problems in bipartite unweighted graphs [10], and general packing problems [1]. In the remainder of this section we review work related to online maximum matching.

Online matching in the standard model. For online weighted matching in the standard model (without recourse), it is easy to see that no algorithm can achieve a bounded competitive ratio. This holds for both the vertex and the edge arrival models. For unweighted matching, the seminal work of Karp *et al.* [13, 7] gave a randomized online algorithm with competitive ratio $\mathrm{e}/(\mathrm{e}-1)$ in the vertex arrival model together with a matching lower bound on any online algorithm. For the edge arrival model and the randomized competitive ratio, [4] showed a lower bound of $(3+1/\varphi^2)/2$, where φ is the golden ratio, as well as an upper bound of 1.8 for the special case of forests.

It is well known that any inclusion-wise maximal matching has cardinality at least half of the optimal maximum cardinality matching. From this it follows that the greedy online algorithm has competitive ratio at most 2, which in the standard model is optimal among all deterministic online algorithms.

Late reject. In the vertex arrival model, the greedy algorithm achieves trivially the competitive ratio of 2, which is optimal for all deterministic online algorithms. The situation differs in the edge arrival model. Epstein *et al.* [6] showed that for online weighted matching,

the deterministic competitive ratio is exactly $3 + 2\sqrt{2} \approx 5.828$, as the upper bound of [14] matches the lower bound of [18]. The same paper [6] shows that the randomized competitive ratio is between $1 + \log 2 \approx 1.693$ and 5.356. Chiplunkar *et al.* [5] presented a randomized 28/15-competitive algorithm for trees and a 4/3-competitive algorithm for paths.

Edge k-bounded recourse. This model was introduced and studied by Avitabile *et al.* [1] for the edge arrival setting, in the context of a much broader class of online packing problems. They gave an algorithm, which we call AMP, that combines doubling techniques with optimal solutions to offline instances of the problem, which has competitive ratio $1 + O\left(\frac{\log k}{k}\right)$ (see Section 2.1 for an analysis of AMP). On the negative side, they showed that no randomized algorithm can be better than $1 + 1/(9k - 1)$-competitive; we note also that their construction implies a lower bound of $1 + 1/k$ for all deterministic algorithms.

Boyar *et al.* [3] showed that the deterministic competitive ratio is 2 for $k = 1$ and 3/2 for $k = 2$, and these optimal ratios are achieved by the greedy algorithm. Moreover, [3] studied several other problems for a value of the recourse parameter equal to 2, such as independent set, vertex cover and minimum spanning forest.

Minimizing recourse. Bernstein *et al.* [2] studied a different recourse model in which the algorithm has to maintain an optimal matching, while minimizing a recourse measure, namely the number of times edges enter or leave the matching maintained by the algorithm. They considered the setting of a bipartite graph and the vertex arrival model and showed that a simple greedy algorithm achieves optimality using $O(n \log^2 n)$ replacements, where n is the number of nodes in the arriving bipartition, whereas the corresponding lower bound for any replacement strategy is $\Omega(n \log n)$.

The results of Avitabile *et al.* [1] were originally formulated in a similar dual setting. More precisely, [1] asks the question: how big should the edge budget k be such that there is a $1 + \varepsilon$ competitive online algorithm that makes at most k changes per edge? They showed that $k = O(\log(1/\varepsilon)/\varepsilon)$ suffices.

Contribution of this work

In the first part of this work, we study the online matching problem with edge k-bounded recourse under the edge arrival model. For this problem, we provide improvements on both upper and lower bounds. First, we revisit the doubling algorithm of [1] that was originally analyzed in the general context of online packing problems. We give a better analysis, specifically for the problem at hand, that uses concepts and ideas related to the matching problem; we also show that the AMP algorithm has competitive ratio $1 + O(\frac{\log k}{k})$. On the negative side, we show that no deterministic algorithm is better than $1 + 1/(k - 1)$ competitive, improving upon the known bound of $1 + 1/k$ of [1].

At first sight these improvements may seem marginal; however one should take into consideration that k is typically a small parameter, and thus the improvements are by no means negligible. In this spirit, we propose and analyze a variant of the greedy algorithm which we call L-Greedy. This algorithm applies, at any step, augmenting paths as long as their length is at most $2L + 1$. We show that for a suitable choice of L, this algorithm is $(1 + O(1/\sqrt{k}))$-competitive. While this algorithm is thus not superior to AMP for large k (and more specifically, to its improved analysis in the context of the matching problem), for small k (and in particular, for $k \leq 20$) it does achieve an improved competitive ratio. Boyar *et al.* [3] showed that the greedy algorithm is 3/2-competitive for $k = 2$. We extended this result to all even k, while for odd k, the competitive ratio is 2.

In terms of techniques, we analyze both AMP and L-Greedy using amortization arguments in which the profit of the algorithms is expressed in terms of weights appropriately distributed over nodes in the graph. We achieve these improvements by exploiting properties of augmenting paths in matching algorithms.

The second part of the paper is devoted to the *edge arrival/departure model*, which is the fully dynamic variant of the online matching problem. First, we observe that the analysis of L-Greedy and AMP carries through in this model as well. On the negative side, we show a lower bound of $(k^2-3k+6)/(k^2-4k+7)$ for all even $k \geq 4$. For $k \in \{2,3\}$, the competitive ratio is $3/2$. We obtain the lower bounds by modeling the game between the algorithm and the adversary as a game played over *strings* of numbers 0 up to k.

We note that, for the analysis of AMP and of L-Greedy, we assume that k is even. This assumption is borrowed from [1] and is required for the analysis. Of course for odd $k \geq 3$ these algorithms can be run with budget $k-1$, providing a valid upper bound on the competitive ratio. Note that our lower bound in the arrival model holds for all values of k.

Due to space limitations some of the proofs are omitted in this paper.

1.1 Preliminaries

A *matching* in a graph $G = (V, E)$ is a set of edges $M \subseteq E$ with disjoint endpoints. A vertex $v \in V$ is said to be *matched* by M if there is an edge $e \in M$ incident to v, and is *unmatched* otherwise. A key concept in maximum matching algorithms is the notion of an *augmenting alternating path*, or simply *augmenting path*. A path P in G is a sequence of vertices $v_0, v_1, \ldots, v_\ell$ for some length $\ell \geq 2$, such that $(v_i, v_{i+1}) \in E$ for all $i = 0, \ldots, \ell-1$. It is said to be *alternating with respect to* M if every other edge of P belongs to M. Moreover it is said to be *augmenting* if the first and the last vertex is unmatched by M. Applying such a path P to M consists in removing from M the edges in $M \cap P$ and adding the edges in $P \setminus M$. The resulting matching has cardinality $M+1$, and every previously matched vertex remains matched.

We define some concepts that will be useful in the analysis of algorithms throughout the paper. We will associate each edge with a *type* which is an integer in $[0, k]$. An edge is of type i if it has undergone i decision flips by the algorithm. Hence, for an edge of type k, where k is the recourse budget, its decision has been finalized, and cannot change further; we call such an edge *blocked*. The type of a path P is defined by the sequence of the types of its edges, and to make this concept unambiguous, we choose between the two orientations of the path the one that results in the lexicographically minimal such sequence. Note that when the algorithm applies some augmenting path P to its current matching M, then the type of every edge in P is increased by 1. Moreover, the two extreme edges of an augmenting path are of type 0, because the endpoints of P are unmatched. We will call a path *blocked* if it contains a blocked edge.

2 The edge arrival model

2.1 The algorithm AMP

In the more general *online set packing problem* sets arrive online and the goal is to maintain a collection of disjoint sets, maximizing their number. Avitabile *et al.* [1] proposed the *doubling algorithm* which is defined for even k only. It has a parameter $r > 1$ and there is a decision variable for every set which can be changed at most k times. The algorithm works in phases, sequentially numbered by an integer p. Initially $p = 0$, and $\mathrm{ALG}_0 = \emptyset$. Let ℓ be the largest

integer such that the optimal solution has value at least r^ℓ, and let ℓ be $-\infty$ if the optimal solution is empty. Whenever this value increases, the algorithms starts a new phase. We define $\ell(p)$ as the value ℓ during phase p. We have $\ell(p) + i \leq \ell(p + i)$ for every positive integer i. At the beginning of a new phase, all decision variables that have been changed fewer than k times are set as in OPT, resulting in the current solution ALG_p (note that the algorithm crucially depends on k being even in order to produce a feasible solution).

Avitabile *et al.* show that the competitive ratio of the algorithm is at most

$$\min_{r>r_0} \frac{r^k(r-1)}{r^{k-1}(r-1)-r}, \tag{1}$$

where r_0 is the solution to the equation $r^{k-1}(r-1) - r = 0$ in $[1, +\infty)$.

We will show how to obtain an improved analysis of the algorithm in the context of the matching problem. Since we know optimal algorithms for $k = 1, 2$ [3] and for $k = 3$ (see Section 2.4), for the analysis we assume $k \geq 4$. We begin by a restatement of the update phase that will help us exploit the structure of solutions obtained via augmenting paths. More specifically, on every edge arrival the algorithm updates a current optimal solution OPT.

At the beginning of a new phase, the algorithm produces a matching ALG_p obtained from ALG_{p-1} as follows: every edge $e \in \mathrm{ALG}_{p-1} \setminus \mathrm{OPT}$ is removed from the current matching, and every edge $e \in \mathrm{OPT} \setminus \mathrm{ALG}_{p-1}$ which is of type strictly smaller than k is added to the current matching. Note that edges adjacent to e have been removed, hence ALG_p is indeed a matching. Also note that all edges added or removed by the algorithm have their type increased by one.

Since ALG_{p-1} and OPT are matchings, their symmetric difference, excluding type k edges, consists of alternating cycles and alternating paths which can be of even or odd length. This means that the algorithm simply applies at the beginning of every phase all those alternating paths and cycles.

We start the analysis of the competitive ratio by bounding OPT during phase p as

$$r^{\ell(p)} \leq \mathrm{OPT} < r^{\ell(p)+1}, \tag{2}$$

which follows by the definition of phases.

During the phase $p \geq 1$ the competitive ratio is $\mathrm{OPT}/\mathrm{ALG}_p$. The type of an edge increases by 1 at most with each phase. Hence in the beginning of the k first phases the algorithm synchronizes with OPT as there are no blocked edges yet, and as a result during these phases the ratio is 1 at the beginning and does not exceed r by the upper bound in (2).

For the remaining phases we need the following argument.

▶ **Proposition 1.** *For even k and any phase $p \geq k + 1$, AMP maintains a matching ALG_p of size at least $r^{\ell(p)} - r^{\ell(p-k+1)+1}$.*

Proof. We denote by the *type of a vertex v* the maximum type of the edges adjacent to v, and by $n_{i,p}$ the number of vertices of type i in phase p. In addition, we denote by OPT_p the value of OPT at the beginning of phase p. With every new phase the type of a vertex can increase at most by 1. Hence every vertex of type k in phase p had positive type in phase $p - k + 1$. Thus

$$n_{k,p} \leq \sum_{i=1}^{k} n_{i,p-k+1} \leq 2 \cdot \mathrm{OPT}_{p-k+1},$$

where the last inequality uses the fact that the left hand side counts the number of vertices matched by the algorithm. In phase p, the difference between the optimal matching and the matching of the algorithm is at most the number of blocked augmenting paths, and each of them contains at least two type k vertices. Hence

$$\begin{aligned} ALG_p &\geq \mathrm{OPT}_p - \frac{1}{2} \cdot n_{k,p} \\ &\geq \mathrm{OPT}_p - \mathrm{OPT}_{p-k+1} \\ &> r^{\ell(p)} - r^{\ell(p-k+1)+1}. \end{aligned}$$

The last inequality holds since $r^{\ell(p)} \leq \mathrm{OPT}_p < r^{\ell(p)+1}$. ◀

Combining this proposition with the bounds (2) we obtain the following bound.

▶ **Proposition 2.** *The competitive ratio of AMP for $k \geq 4$ is upper bounded by the expression*

$$\min_{r>1} \frac{r^k}{r^{k-1} - r}. \tag{3}$$

In addition we can show that this is a better upper bound.

▶ **Proposition 3.** *For all even $k \geq 4$, Expression* (1) *upper bounds Expression* (3)*.*

The following theorem concludes the asymptotic analysis of the performance of AMP.

▶ **Theorem 1.** *For all even k, AMP has competitive ratio $1 + O(\frac{\log k}{k})$.*

Proof sketch. We first sketch a simple argument based on the Puiseux series expansion [17]: this is a type of power series that allows fractional powers, as opposed to only integer ones (e.g., Taylor series). In the full version we provide a second proof that relies only on standard calculus. Let r denote the optimal choice of the parameter, namely the one that minimizes (3). By analyzing the derivative, it follows that $r = (k-1)^{1/(k-2)}$, hence the competitive ratio is at most $\frac{(k-1)^{\frac{k-1}{k-2}}}{k-2}$, whose Puiseux series expansion at $k = \infty$ is $1 + \frac{\log k + 1}{k} + O(\frac{1}{k^2})$. ◀

2.2 The algorithm Greedy

We consider the algorithm Greedy, which repeatedly applies an arbitrary augmenting path whenever possible. This algorithm achieves an upper bound of 3/2 for $k = 2$ as has been shown in [3]. We show that the same guarantee holds for all even k.In what concerns the lower bound, the idea is to force the algorithm to augment an arbitrarily long path in order to create a configuration with an arbitrarily large number of blocked augmenting paths of lengths 5, which locally have ratio 3/2.

▶ **Proposition 4.** *The competitive ratio of* Greedy *is* 3/2 *for every even k and* 2 *for every odd k.*

2.3 The algorithm L-Greedy

The greedy algorithm has inferior performance because it augments along arbitrarily long augmenting paths, therefore sometimes sacrificing edge budget for only a marginal increase in the matching size. A natural idea towards an improvement would be to apply only short augmenting paths, as they are more budget efficient. For technical reasons, we restrict the choice of augmenting paths even further.

We define the algorithm L-Greedy for some given parameter L, which applies any non-blocked augmenting path of length at most $2L+1$ that is in the symmetric difference between the current matching and some particular optimal matching OPT. The latter is updated after each edge arrival by applying an augmenting path for OPT. Note that L-Greedy may not change its solution even if there is a short augmenting path in the current graph if it contains edges which are not in this particular optimal matching OPT. We will later optimize the parameter L as function of k.

2.3.1 Analysis of L-Greedy

We begin by observing that for $L = 0$ the algorithm collects greedily vertex disjoint edges without any recourse, which is precisely the behavior of Greedy for $k = 1$ and has competitive ratio 2. For $L = 1$ the algorithm L-Greedy applies only augmenting paths of length at most 3. In this case, the same argument as in the proof of Proposition 4 shows that the competitive ratio of L-Greedy is 3/2.

In what follows we analyze the general case $L \geq 2$. To this end, we assign weights to vertices in such a way that the total vertex weight equals the size of the current matching. Therefore, whenever the size of the matching is increased by 1, a total weight of 1 is distributed on the vertices along the augmenting path. Vertices in this path that were already matched receive a weight α, where $\alpha \geq 0$ is some constant that we specify later. Finally, the two vertices on the endpoints of the augmenting path receive the remaining weight, that is $1/2 - \ell\alpha$, where $2\ell + 1$ is the length of the path. It follows, from this weight assignment, that every unmatched vertex has weight 0, that every matched vertex has weight at least $1/2 - L\alpha$, and that every endpoint of a type k edge has weight at least $1/2 - L\alpha + (k-1)\alpha$.

Suppose that the online algorithm reaches a configuration in which it cannot apply any augmenting path, as specified in its statement. We consider the symmetric difference between the matching produced by the algorithm and the optimal matching maintained internally by the algorithm. This symmetric difference consists of alternating paths and/or alternating cycles, and we will upper bound for each such component the ratio between the number of edges in the optimal matching and the total vertex weight, which we call the *local ratio*. In particular, a component in the symmetric difference falls in one of the following cases: Either it is an augmenting path of length $2\ell + 1 \leq 2L + 1$, or an augmenting path of length $2\ell + 1 > 2L + 1$, or an alternating cycle or alternating path of even length.

Case 1: Augmenting path of length $2\ell + 1 \leq 2L + 1$. Such a path contains at least one edge of type k. It follows that $\ell \geq 2$, since an augmenting path of length 1 is a single type 0 edge, and an augmenting path of length 3 has edge types respectively $0, t, 0$ for some odd t (and k is assumed to be even). The path contains 2ℓ matched vertices, and at least 2 of them are adjacent to a type k edge. Hence the total vertex weight is at least $2\ell\left(\frac{1}{2} - L\alpha\right) + 2(k-1)\alpha$, and the local ratio of this component is at most

$$\frac{\ell + 1}{\ell - 2\ell L\alpha + 2(k-1)\alpha}. \tag{4}$$

Case 2: Augmenting path of length $2\ell + 1 > 2L + 1$. Such a path contains 2ℓ matched vertices and therefore the local ratio is at most

$$\frac{\ell + 1}{\ell - 2\ell L\alpha}. \tag{5}$$

Case 3: Alternating cycle or path of even length. Such a component contains 2ℓ matched vertices and therefore the local ratio is at most

$$\frac{\ell}{\ell - 2\ell L\alpha}, \tag{6}$$

which is dominated by (5). We choose α so as to minimize the maximum of the local ratios, as defined by (4) and (5). Then for this choice of α we optimize L; namely we choose the value $L = \lfloor\sqrt{k-1}\rfloor$. Note that for $k = 4$, this leads to the choice $L = 1$, which we analyzed in the beginning of the section. We obtain the following performance guarantee.

▶ **Theorem 2.** *The competitive ratio of* L-Greedy *with* $L = \lfloor\sqrt{k-1}\rfloor$ *is at most*

$$\frac{k(L+2)-2}{(L+1)(k-1)} = 1 + O\left(\frac{1}{\sqrt{k}}\right),$$

for even $k \geq 6$ *and at most* $3/2$ *for* $k = 4$.

One can show that this analysis of L-Greedy is essentially tight.

2.4 Lower bound for deterministic algorithms

Boyar *et al.* [3] show that the deterministic competitive ratio of the problem is 2 for $k = 1$ and $3/2$ for $k = 2$. We complete this picture by showing a lower bound of $1 + \frac{1}{k-1}$ for all $k \geq 3$. Note that the lower bound is tight for $k = 3$, as the algorithm Greedy, which works by assuming that k is only 2, has competitive ratio $3/2$.

▶ **Theorem 3.** *The deterministic competitive ratio of the online matching problem with edge-bounded recourse is at least* $1 + \frac{1}{k-1}$ *for all* $k \geq 3$.

Proof sketch. We consider three cases, namely the cases $k = 3$, k is even and at least 4, and finally k is odd and at least 5. For each case we present an appropriate adversarial argument. Due to space limitations we only present the case $k = 3$. Suppose, by way of contradiction, that for $k = 3$ some algorithm claims a competitive ratio strictly smaller than $(3n+2)/(2n+2)$ for some arbitrary $n \geq 1$. The adversary releases a single edge, creating an augmenting path of length 1. Then the algorithm applies the augmenting path, which the adversary extends by appending one edge on each side, creating an augmenting path of type 0,1,0, as shown in Figure 2(a). Since the current ratio is 2, the algorithm needs to apply this path, which the adversary again extends by appending an edge on each side, creating an augmenting path of type 0,1,2,1,0, as shown in Figure 2(b). Since the current ratio is $3/2$, the algorithm applies this path. In response the adversary appends an edge at each endpoint of the type 3 edge, and at each endpoint of one of the type 1 edges, as shown in Figure 2(c). The resulting graph has a blocked augmenting path of type 0,3,0, and an augmenting path of type 0,1,0, as shown in Figure 2(c). The algorithm needs to apply the latter one as the ratio is currently $5/3 > 3/2$.

At this point, the adversary repeats this construction $n - 1$ times, by identifying the shaded part of Figure 2(c) as the graph of Figure 2(a), and reapplying the above construction. The final graph consists of n blocked augmenting paths of type 0,3,0 and $n + 1$ edges of type 1 that belong both to the optimal and the algorithm's matchings. Figure 2(d) illustrates this final graph for $n = 4$. Hence the competitive ratio is $(3n+1)/(2n+1)$, contradicting the claimed ratio and showing a lower bound of $3/2$. ◀

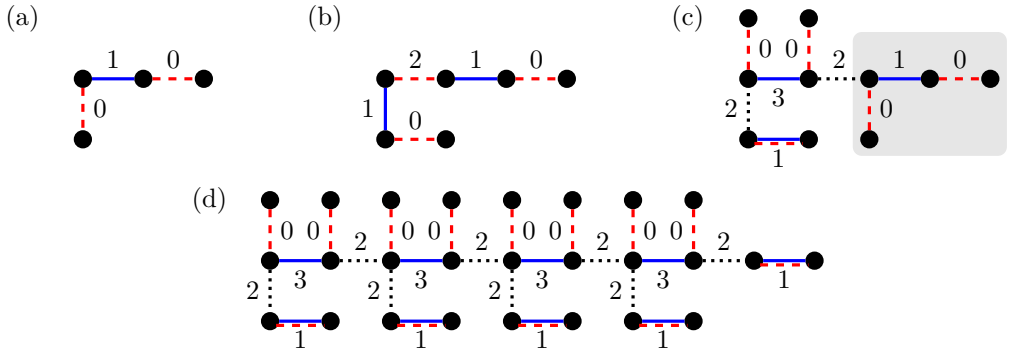


Figure 2 Lower bound construction for the case $k = 3$.

Table 1 Summary of lower bounds (LB) and upper bounds on the competitive ratio for the problem, for all even $4 \leq k \leq 22$. The lower bounds for the arrival/departure model are discussed in Section 3. The analysis of L-Greedy and AMP carry through to the (limited) edge arrival/departure model. For $k \geq 22$, the upper bound of AMP is superior to the upper bound of L-Greedy.

k	LB (arr.)	LB (arr./dep.)	L-Greedy	AMP
4	1.333333	1.428571	1.5	2.598076
6	1.2	1.263158	1.466667	1.869186
8	1.142857	1.179487	1.428571	1.613602
10	1.111111	1.134328	1.333333	1.480583
12	1.090909	1.106796	1.318182	1.398080
14	1.076923	1.088435	1.307692	1.341500
16	1.066666	1.075377	1.300000	1.300080
18	1.058823	1.065637	1.247059	1.268330
20	1.052631	1.058104	1.242105	1.243150
22	1.047619	1.052109	1.238095	1.222640

2.5 Comparing the algorithms L-Greedy and AMP

We have analyzed two deterministic online algorithms: the algorithm AMP, which has competitive ratio $1+O\left(\log(k)/k\right)$, and the algorithm L-Greedy, which has competitive ratio $1+\Theta(1/\sqrt{k})$. Since the analysis of L-Greedy is tight, it follows that AMP is asymptotically (i.e., for large k) superior to L-Greedy. However, for small values of k, namely $k \leq 20$, we observe that L-Greedy performs better, in comparison to the performance bound we have shown for AMP. These findings are summarized in Table 1 and Figure 3.

3 The edge arrival/departure model

In this section we consider the online matching problem in the setting in which edges may arrive but also *depart* online. In this context one can distinguish two models. In the *limited departure model* an edge cannot be removed from the instance while it is matched by the online algorithm, while in the stronger *full departure model* any edge can be removed.

It turns out that the latter model is quite restrictive. This is because it is possible for the adversary to force an online algorithm to augment some augmenting path and then to remove one of the edges in its matching. Eventually the algorithm can end up with blocked edges (type k), without having the chance to augment its matching. This intuition is formalized in the following lemma.

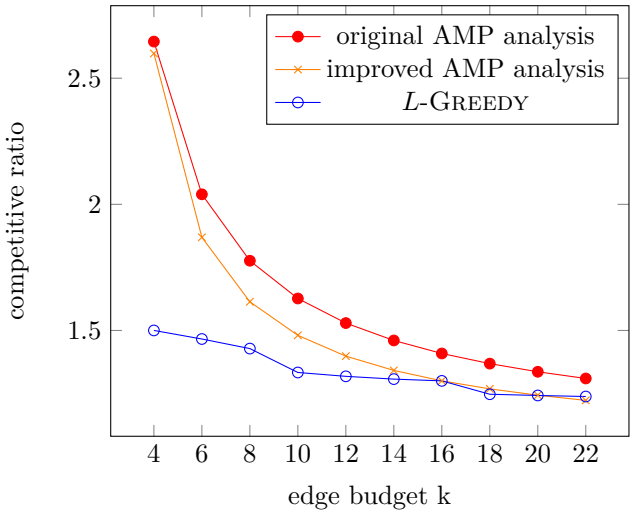


Figure 3 Comparison of the competitive ratios of the algorithm AMP and the algorithm L-Greedy.

▶ **Lemma 4.** *The competitive ratio in the full departure model is* 2.

Since the full departure model is very restrictive for the algorithm, as shown in Lemma 4, we will concentrate on the limited departure model, as defined in the introduction. For this model, we observe that the algorithms L-Greedy and AMP have the same performance guarantee as in the edge arrival model. This is because the analysis of L-Greedy uses weights on vertices which are not affected by edge departures, and the analysis of AMP is based on an upper bound over the number of type k edges, which still holds under edge departures. We thus focus on obtaining stronger lower bounds in this model (also included in Table 1). We begin by observing that the bound of 3/2 of the competitive ratio in the edge arrival model for $k \in \{2, 3\}$ still holds for the limited departure model, where the adversary is stronger. Hence, the smallest interesting value for k in this model is $k = 4$, for which we provide the following specific lower bound. The proof will also provide some intuition about the adversarial argument for general k.

▶ **Theorem 5.** *The competitive ratio in the* limited departure model *is at least* $10/7$ *for* $k = 4$.

Proof. We specify a particular adversary that maintains a graph such that the symmetric difference between the matching produced by the algorithm and the optimal matching consists only of augmenting paths and has no alternating cycles or alternating paths of even length.

The edge types along any path form a string over the integers $\{0, 1, \ldots, k\}$ with alternating parity and starting and ending with 0. We call these strings *alternating*. Thus, rather than a game played between the algorithm and the adversary on a *graph*, we will consider the game played on a *collection of alternating strings*.

Whenever the algorithm applies an augmenting path, this translates into the increment of each integer of the corresponding string, for example $01210 \to 12321$. The adversary responds to this action by three types of actions. First, the adversary may possibly split the string into smaller strings, for example $12321 \to 123, 1$ or $12321 \to 1, 3, 1$. This corresponds to deleting some edges which are currently not matched by the algorithm. Second, the adversary may possibly merge some of the resulting strings by concatenating them on both

sides of a 0, for example $1,1 \to 101$. Finally the adversary may append 0's to the ends of the strings, where needed to make them alternating, for example $101 \to 01010$. The last two actions correspond to the insertion of (type 0) edges by the adversary.

The main idea is as follows. Consider an algorithm that claims a competitive ratio at most $(10-\epsilon)/7$ for some sufficiently small $\epsilon > 0$. We will then show that the adversary can force the algorithm into a final configuration of ratio at least $10/7$, a contradiction.

We describe the current configuration by variables v, w, x, y, z which count the number of the specific strings that are described in Figure 4. In particular, z is the total number of strings 030 and 01410, which we call *bad strings*. The name is motivated by the fact that 01410 has worst local ratio among all blocked strings and is created from 030 strings.

The adversary starts by presenting the string 0, which the algorithm has to augment, resulting in a single string 010. This unique adversarial action has been omitted for simplification from Figure 4. From this moment onwards, we distinguish 3 different phases during the game, decided by the adversary, and depicted in Figure 4. For example, if the algorithm augments the string 01010, then in phase 1 the adversary will replace it with the strings 010 and 01210, while in phases 2 and 3 he will replace it with the string 0121210.

At a high level, the objective in phase 1 is to create a large enough number of bad strings, while phase 2 creates a large enough proportion of bad strings. Last, phase 3 leads the algorithm to a configuration which consists only of blocked strings.

The game starts with phase 1. During this phase the competitive ratio is at least $3/2$, thus forcing the algorithm to continue augmenting strings. After a finite number of steps, the condition $7z+3 > 4/\epsilon$ holds and the game moves to phase 2. During the second phase we have the invariants (1) $7z+3 > 4/\epsilon$ and (2) $2x+y+2w \le z+1$. Invariant (1) holds because the left-hand side will not decrease throughout the phase. Moreover, Invariant (2) follows from the definitions of the involved parameters and the transitions between strings, as defined by the statement of the phase. Together both invariants imply that the competitive ratio is strictly larger than $(10-\epsilon)/7$, forcing the algorithm to augment strings. This is because the competitive ratio can be lower bounded by

$$\frac{3z+3y+4(x+w)}{2z+2y+3(x+w)} \ge \frac{10z+2y+4}{7z+y+3} \ge \frac{10z+4}{7z+3} \ge \frac{10-\epsilon/2}{7},$$

where the first inequality follows from Invariant (2) and the last one from (1).

In phase 2, at some moment eventually the condition $z \ge 8(x+y+v+w)$ will hold since $x+y+v+w$ does not change, but any sequence of $x+y+1$ steps increases z. At that moment, the game starts phase 3. We note that the condition is preserved during phase 3 and that it implies a ratio of at least $10/7$, forcing the algorithm to a configuration consisting only of the blocked strings 01410 and 012343210. This is because $z \ge 8(x+y+v+w)$ implies $z+y+x \ge 2w+8v$, which in turn implies

$$\frac{3(x+y+z)+4w+6v}{2(x+y+z)+3w+5v} \ge \frac{10}{7},$$

where the left hand side lower bounds the competitive ratio in phase 3. ◀

We can generalize the ideas in the proof of Theorem 5 so as to obtain a non-trivial lower bound for general even $k \ge 4$, see Figure 5.

▶ **Theorem 6.** *The competitive ratio in the limited departure model is at least* $\frac{k^2-3k+6}{k^2-4k+7}$*, for all even* $k \ge 4$*.*

Since $\frac{k^2-3k+6}{k^2-4k+7} > 1 + \frac{1}{k-1}$ for all $k \ge 4$, Theorem 6 shows a stronger lower bound for even k than Theorem 3 under the limited departure model.

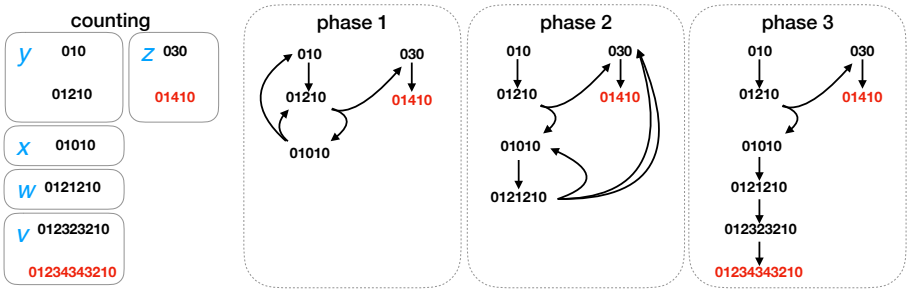


Figure 4 The lower bound construction in the arrival/departure model and $k = 4$.

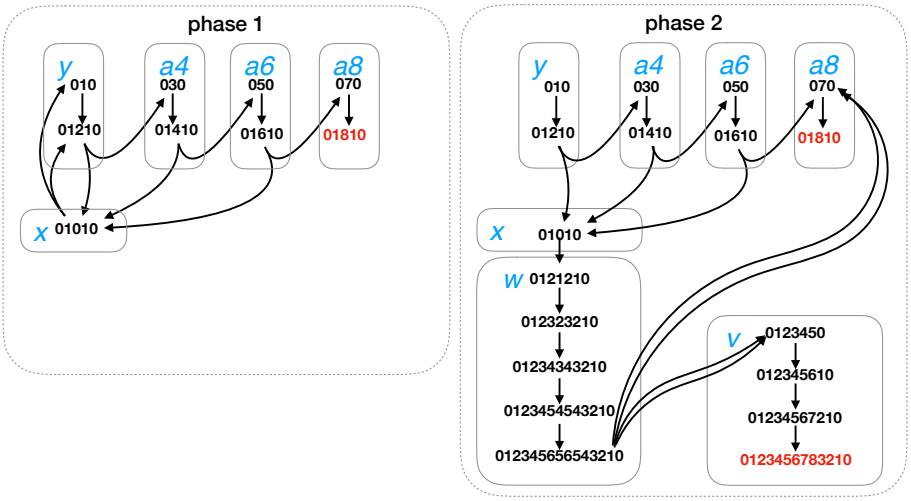


Figure 5 The lower bound construction in the arrival/departure model illustrated for $k = 8$. Blocked strings are depicted in red. The arcs illustrate the adversarial strategy. For example, if the algorithm augments the string 012345656543210, the adversary replaces the resulting string by two strings 070 and two strings 0123450.

References

1 Tess Avitabile, Claire Mathieu, and Laura H. Parkinson. Online constrained optimization with recourse. *Information Processing Letters*, 113(3):81–86, 2013.

2 Aaron Bernstein, Jacob Holm, and Eva Rotenberg. Online bipartite matching with amortized replacements. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 947–959. SIAM, 2018.

3 Joan Boyar, Lene M. Favrholdt, Michal Kotrbčík, and Kim S. Larsen. Relaxing the irrevocability requirement for online graph algorithms. In *Proceedings of the 15th Workshop on Algorithms and Data Structures, (WADS)*, pages 217–228. Springer, 2017.

4 Niv Buchbinder, Danny Segev, and Yevgeny Tkach. Online algorithms for maximum cardinality matching with edge arrivals. In *Proceedings of the 25th Annual European Symposium on Algorithms, (ESA), Vienna, Austria*, pages 22:1–22:14, 2017.

5 Ashish Chiplunkar, Sumedh Tirodkar, and Sundar Vishwanathan. On randomized algorithms for matching in the online preemptive model. In *Proceedings of the 23rd Annual European Symposium on Algorithms, (ESA)*, pages 325–336. Springer, 2015.

6 Leah Epstein, Asaf Levin, Danny Segev, and Oren Weimann. Improved bounds for online preemptive matching. In *Proceedings of the 30th International Symposium on Theoretical*

Aspects of Computer Science, (STACS), pages 389–399, 2013.

7 Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 982–991. Society for Industrial and Applied Mathematics, 2008.

8 Albert Gu, Anupam Gupta, and Amit Kumar. The power of deferral: maintaining a constant-competitive steiner tree online. *SIAM Journal on Computing*, 45(1):1–28, 2016.

9 Anupam Gupta and Amit Kumar. Online steiner tree with deletions. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA), Portland, Oregon, USA*, pages 455–467, 2014.

10 Anupam Gupta, Amit Kumar, and Cliff Stein. Maintaining assignments online: Matching, scheduling, and flows. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 468–479, 2014.

11 Xin Han and Kazuhisa Makino. Online minimization knapsack problem. In *Proceedings of the 7th International Workshop on Approximation and Online Algorithms, (WAOA)*, pages 182–193. Springer, 2009.

12 Kazuo Iwama and Shiro Taketomi. Removable online knapsack problems. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming, (ICALP)*, pages 293–305, 2002.

13 Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, (STOC)*, pages 352–358. ACM, 1990.

14 Andrew McGregor. Finding graph matchings in data streams. In *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, (APPROX-RANDOM)*, pages 170–181. Springer, 2005.

15 Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. The power of recourse for online MST and TSP. *SIAM Journal on Computing*, 45(3):859–880, 2016.

16 Aranyak Mehta. Online matching and ad allocation. *Foundations and Trends in Theoretical Computer Science*, 8(4):265–368, 2013.

17 C.L. Siegel. *Topics in Complex Function Theory, Vol. 1: Elliptic Functions and Uniformization Theory.* New York: Wiley, 1988.

18 Ashwinkumar Badanidiyuru Varadaraja. Buyback problem-approximate matroid intersection with cancellation costs. In *Proceedings of the 38th International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 379–390. Springer, 2011.

Linking Focusing and Resolution with Selection

Guillaume Burel
ENSIIE and Samovar, Télécom SudParis and CNRS, Université Paris-Saclay, Évry, France
Inria and LSV, CNRS and ENS Paris-Saclay, Université Paris-Saclay, Cachan, France
guillaume.burel@ensiie.fr

Abstract

Focusing and selection are techniques that shrink the proof search space for respectively sequent calculi and resolution. To bring out a link between them, we generalize them both: we introduce a sequent calculus where each *occurrence* of an atom can have a positive or a negative polarity; and a resolution method where each literal, whatever its sign, can be selected in input clauses. We prove the equivalence between cut-free proofs in this sequent calculus and derivations of the empty clause in that resolution method. Such a generalization is not semi-complete in general, which allows us to consider complete instances that correspond to theories of any logical strength. We present three complete instances: first, our framework allows us to show that ordinary focusing corresponds to hyperresolution and semantic resolution; the second instance is deduction modulo theory; and a new setting, not captured by any existing framework, extends deduction modulo theory with rewriting rules having several left-hand sides, which restricts even more the proof search space.

2012 ACM Subject Classification Theory of computation → Proof theory, Theory of computation → Automated reasoning

Keywords and phrases logic in computer science, automated deduction, proof theory, sequent calculus, refinements of resolution, deduction modulo theory, polarization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.9

Related Version A full version of the paper can be found at `https://hal.inria.fr/hal-01670476`.

1 Introduction

In addition to clever implementation techniques and data structures, a key point that explains the success of state-of-the-art automated theorem provers is the use of calculi that dramatically reduce proof search space. In the last decades, the independent developments of two families of techniques can be highlighted. First, in the kind of methods based on resolution, proof search space can be shrunk using ordering and selection techniques. The intuition is to restrict the application of the resolution rule to only some literals in a clause. If equality is considered, this leads to the superposition calculus [2] which is the base calculus of the currently most efficient automated provers for first-order classical logic. Second, in sequent calculi, Andreoli [1] introduced a technique called focusing to reduce non-determinism in the application of sequent-calculus rules. It works by first applying all invertible rules (those whose conclusion is logically equivalent to their premises) and second by chaining the application of non-invertible rules. Originally developed for linear logic, focusing has been extended to intuitionistic and classical first-order logic [26]. Focusing is mostly used in fields where sequent calculi, and related inverse and tableaux methods, are the most accurate proving method. For instance, there exists tools for first-order linear logic [12], for intuitionistic logic [27] and for modal logic [28]. Focusing is also the key ingredient in Miller's ProofCert project aiming at building a universal framework for proof certification [15].

Despite their apparent lack of relation, we show in this paper that selection in refinements of the resolution calculus and focusing in sequent calculus are in fact strongly related, so that ordinary focusing in classical first-order logic corresponds actually to hyperresolution, where all negative literals are selected in a clause and are resolved at once. This connection is obtained by relaxing both techniques: concerning resolution, we allow any literal of the input clauses to be selected, whatever its sign; for the focusing part, we allow polarization not only of connectives, but also of all occurrences of literals. The main theorem of this paper, Theorem 3, shows that the sets of clauses whose insatisfiability can be proved by the resolution method with arbitrary input selection are exactly the sequents that have a cut-free proof in the generalized focusing setting.

This generalization allows us to cover a wider spectrum of proof systems. In particular, this permits to consider systems that search for proofs modulo some theory. Indeed, in real world applications, proof obligations are often verified within one or several theories. This explains the interest in and the success of Satisfiability Modulo Theory tools in recent years. Embedding a theory in our framework amounts to giving an axiomatic presentation of it where some literals are selected.

By relaxing the conditions for selecting literals, our framework is not always refutationaly complete. However, this should not be considered as a drawback, but as an essential point to be able to represent efficiently all kinds of theories. Indeed, let us consider a proof search method $\mathbb{P}(\mathcal{T})$ parameterized by a theory $\mathcal{T}$. Ideally, $\mathbb{P}(\mathcal{T})$ should be as efficient as a generic proof search method if it is fed with a formula that is not related to the theory $\mathcal{T}$. In particular, if it tries to refute the true formula $\top$, it should terminate, and with the answer "NO". Let us say that $\mathbb{P}(\mathcal{T})$ is relatively consistent if it is the case. As we pointed out with Dowek [8], we cannot have a generic proof of the completeness of a relatively consistent method $\mathbb{P}(\mathcal{T})$ that would work for all $\mathcal{T}$. Indeed, such a proof would imply the consistency of the theory $\mathcal{T}$, and, according to Gödel, this cannot be performed in $\mathcal{T}$ itself. So either the completeness of the proof system is proved once and for all, but it cannot represent theories that are logically at least as strong as that proof of completeness; or it is not complete in general but it can be proved to be complete for particular theories of some arbitrary logical strength. What is interesting therefore is to give proofs of completeness of $\mathbb{P}(\mathcal{T})$ for particular theories $\mathcal{T}$.

Therefore, we give three instances of our framework, where we can have proofs of completeness. First, as stated above, we link ordinary focusing with hyperresolution, and, in the ground case, with semantic resolution. Second, we show that Deduction Modulo Theory [20] is also a particular instance of this framework, knowing that there exists numerous proof techniques to prove the completeness of Deduction Modulo a particular theory, for instance [24, 21, 18, 7]. Third, we show how completeness in our framework can be reduced to completeness of several instances of Deduction Modulo Theory. To give an intuition about this last part, and to illustrate how much the proof search space can be constrained without losing completeness, let us consider for example the theory defining the powerset:

$$\forall X,\ \forall Y,\ (X \in \mathcal{P}(Y)) \Leftrightarrow (\forall Z,\ (Z \in X) \Rightarrow (Z \in Y))$$

This theory can be put in clausal normal form, using d as a Skolem symbol, and we select (by underlining them) some literals in these clauses[1]:

[1] We use the associative-commutative-idempotent symbol $\curlyvee$ in clauses to distinguish it from the symbol $\vee$ that is used in formulas.

$$\underline{\neg X \in \mathcal{P}(Y)} \curlyvee \underline{\neg Z \in X} \curlyvee Z \in Y \tag{1}$$

$$\underline{X \in \mathcal{P}(Y)} \curlyvee d(X,Y) \in X \tag{2}$$

$$\underline{X \in \mathcal{P}(Y)} \curlyvee \underline{\neg d(X,Y) \in Y} \tag{3}$$

Using focusing in general, and in our framework in particular, the decomposition of connectives is so restricted that, given an axiom, a proof derivation decomposing this axiom would necessarily have certain shapes. Thus, the axiom can be replaced by new inference rules, called synthetic rules, that are used instead of the derivation of those shapes. See end of Section 2, page 6, for more details. In our framework, this would lead to the following three synthetic rules, that can be used in place of the axioms (the explanation how these rules are obtained is given in Section 5.3):

$$(1)\vdash \frac{\Delta, u \in \mathcal{P}(v), t \in u, t \in v \vdash}{\Delta, u \in \mathcal{P}(v), t \in u \vdash} \qquad (2)\vdash \frac{\Delta, \neg u \in \mathcal{P}(v), d(u,v) \in u \vdash}{\Delta, \neg u \in \mathcal{P}(v) \vdash}$$

$$(3)\vdash \frac{}{\Delta, \neg u \in \mathcal{P}(v), d(u,v) \in v \vdash}$$

The only proof of transitivity of the membership in the powerset is then

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a \in \mathcal{P}(b), b \in \mathcal{P}(c), \neg a \in \mathcal{P}(c), d(a,c) \in a, d(a,c) \in b, d(a,c) \in c \vdash}(3)}{a \in \mathcal{P}(b), b \in \mathcal{P}(c), \neg a \in \mathcal{P}(c), d(a,c) \in a, d(a,c) \in b \vdash}(1)}{a \in \mathcal{P}(b), b \in \mathcal{P}(c), \neg a \in \mathcal{P}(c), d(a,c) \in a \vdash}(1)}{a \in \mathcal{P}(b), b \in \mathcal{P}(c), \neg a \in \mathcal{P}(c) \vdash}(2)}{a \in \mathcal{P}(b) \wedge b \in \mathcal{P}(c) \wedge \neg a \in \mathcal{P}(c) \vdash}\wedge\vdash}{\exists A.\ \exists B.\ \exists C.\ A \in \mathcal{P}(B) \wedge B \in \mathcal{P}(C) \wedge \neg A \in \mathcal{P}(C) \vdash}\exists\vdash$$

where the active formulas in a sequent are underwaved, and double lines indicate potentially several applications of an inference rule.

On the resolution side, clauses (1) to (3) lead to the following ground derived rules (see also Section 5.3):

$$(1)\ \frac{u \in \mathcal{P}(v) \curlyvee C \qquad t \in u \curlyvee D}{t \in v \curlyvee C \curlyvee D} \qquad (2)\ \frac{\neg u \in \mathcal{P}(v) \curlyvee C}{d(u,v) \in u \curlyvee C}$$

$$(3)\ \frac{\neg u \in \mathcal{P}(v) \curlyvee C \qquad d(u,v) \in v \curlyvee D}{C \curlyvee D}$$

Once again, there is only one proof of transitivity, i.e. starting from the set of clauses $\{a \in \mathcal{P}(b); b \in \mathcal{P}(c); \neg a \in \mathcal{P}(c)\}$:

$$(3)\ \frac{\neg a \in \mathcal{P}(c) \qquad (1)\ \dfrac{b \in \mathcal{P}(c) \qquad (1)\ \dfrac{a \in \mathcal{P}(b) \qquad (2)\ \dfrac{\neg a \in \mathcal{P}(c)}{d(a,c) \in a}}{d(a,c) \in b}}{d(a,c) \in c}}{\square}$$

and we cannot even infer other clauses than those. We let the reader compare with what happens if we used clauses (1) to (3) in resolution, even using the ordered resolution with selection refinement.

Related work. Chaudhuri et al. [13] show that hyperresolution for Horn clauses can be explained as an instance of a sequent calculus for intuitionistic linear logic with focusing where atoms are given a negative polarity.

Farooque et al. [23] developed a sequent calculus, based on focusing, that is able to simulate DPLL($\mathcal{T}$), the most common calculus used in SMT provers. The main difference with our framework is that in [23], the theory is considered as a black box which is called as an oracle. Here, the theory is considered as a first-class citizen.

Within the ProofCert project, resolution proofs can be checked by a kernel built upon a sequent calculus with focusing [15]. Based on this, the tool `Checkers` [14] is able to verify proofs coming from automated theorem provers based on resolution such as E-prover. Different from here, they translate resolution derivations using cuts to get smaller proofs.

Hermant [25] proves the correspondance between the cut-free fragment of a sequent calculus and a resolution method, in the setting of Deduction Modulo Theory. Since Deduction Modulo Theory is subsumed by our framework, Theorem 3 is a generalization of Hermant's work. Proving it is simpler in our setting because focusing restrains the shape of possible sequent calculus proofs, whereas Hermant had to prove technical lemmas to give proofs a canonical shape.

Notations and conventions. We use standard definitions for terms, predicates, formulas (with connectives $\bot, \top, \neg, \wedge, \vee$ and quantifiers $\forall, \exists$), sequents and substitutions. A literal is an atom or its negation. A clause is a set of literals. We will identify a literal with the unit clause containing it. Unless stated otherwise, letters $P, Q, R, P', P_1, \dots$ denote atoms, $L, K, L', L_1, \dots$ denote literals, $A, B, A', A_1, \dots$ denote formulas, $C, D, C', C_1, \dots$ denote clauses, Γ, Δ denote set of clauses or set of formulas (depending on the context). $A^{\perp}$ denotes the negation normal form of $\neg A$.

2 Focusing with Polarized Occurrences of Atoms

Focusing was introduced by Andreoli [1] to restrict the non-determinism in some sequent calculus for linear logic. It relies on the alternation of two phases: During the asynchronous phase (sequents with $\Uparrow$), all invertible rules are applied on the formulas of the sequent. Recall that a rule is said invertible if its conclusion implies the conjunction of its premises. During the synchronous phase (sequents with $\Downarrow$), a particular formula is selected – the focus is on it – and all possible non-invertible rules are successively applied on it. This idea has been extended to intuitionistic and classical first-order logic [26]. In these, connectives may have invertible and non-invertible versions of their sequent calculus rules. Therefore, one considers in that case two versions of a connective, one called positive when the right introduction rule is non-invertible, and one called negative when it is invertible. Some connectives, i.e. $\exists$ in classical logic, only have a positive version, and dually, others, such as $\forall$ in classical logic, only have a negative version. Given a usual formula, one can decide which version of a connective one wants to use at a particular occurrence, which is called a polarization of the formula.[2] Note that the polarity of a connective does not affect its semantics, it only alters the shape of the sequent calculus proofs. Similarly, one can decide the polarity of each literal. If a literal with negative polarity L is focused on in a branch, then this branch

[2] Let us note that this notion of polarity is a standard denomination when dealing with focusing, and should not be confused with the more usual but unrelated notion defined by the parity of the negation-depth of a position in a formula.

Asynchronous phase:

$$\widehat{\Uparrow}\vdash \frac{}{\Gamma, L, L^\perp \Uparrow \vdash} \qquad \Uparrow\exists\vdash \frac{\Gamma \Uparrow \Delta, A \vdash}{\Gamma \Uparrow \Delta, \exists x.\ A \vdash}\ x \text{ not free in } \Gamma, \Delta$$

$$\Uparrow\vee\vdash \frac{\Gamma \Uparrow \Delta, A \vdash \qquad \Gamma \Uparrow \Delta, B \vdash}{\Gamma \Uparrow \Delta, A \vee^+ B \vdash} \qquad \Uparrow\wedge\vdash \frac{\Gamma \Uparrow \Delta, A, B \vdash}{\Gamma \Uparrow \Delta, A \wedge^+ B \vdash} \qquad \Uparrow\top\vdash \frac{\Gamma \Uparrow \Delta \vdash}{\Gamma \Uparrow \Delta, \top \vdash}$$

Synchronous phase:

$$\widehat{\Downarrow}\vdash \frac{}{\Gamma, L^\perp \Downarrow \underline{L} \vdash} \qquad \Downarrow\forall\vdash \frac{\Gamma \Downarrow \{t/x\}A \vdash}{\Gamma \Downarrow \forall x.\ A \vdash} \qquad \Downarrow\perp\vdash \frac{}{\Gamma \Downarrow \perp \vdash}$$

$$\Downarrow\vee\vdash \frac{\Gamma \Downarrow A \vdash \qquad \Gamma \Downarrow B \vdash}{\Gamma \Downarrow A \vee^- B \vdash} \qquad \Downarrow\wedge_1\vdash \frac{\Gamma \Downarrow A \vdash}{\Gamma \Downarrow A \wedge^- B \vdash} \qquad \Downarrow\wedge_2\vdash \frac{\Gamma \Downarrow B \vdash}{\Gamma \Downarrow A \wedge^- B \vdash}$$

$$\textsf{Focus}\ \frac{\Gamma, A \Downarrow A \vdash}{\Gamma, A \Uparrow \vdash}\ A \text{ negative} \qquad \textsf{Release}\ \frac{\Gamma \Uparrow A \vdash}{\Gamma \Downarrow A \vdash}\ A \text{ positive} \qquad \textsf{Store}\ \frac{\Gamma, A \Uparrow \Delta \vdash}{\Gamma \Uparrow A, \Delta \vdash}\ A \text{ negative or literal}$$

Figure 1 The sequent calculus LKF$^\perp$.

must necessarily be closed, with $L^\perp$ in the same context. (See rule $\widehat{\Downarrow}\vdash$ in Figure 1.) In the ordinary presentation of focusing, this polarity is chosen globally for all occurrences of each atom, and the polarity of $\neg P$ is defined as the inverse of that of P. In our setting, the polarity is attached to the position of the literal in the formula. In particular, if a substitution is applied to the formula, the polarities of the resulting literals do not change. The polarity of a formula is defined as the polarity of its top connective. Besides, note that to switch the polarity of a formula, e.g. to impose a change of phase, one can prefix it by so-called delays: $\delta^- A$ is negative whatever the polarity of A. Delays can be defined for instance by $\delta^- A = \forall x.\ A$ where x is not free in A, so we do not need them in the syntax and the rules.

Liang and Miller [26] introduce the sequent calculus LKF, and prove it to be complete for classical first-order logic. In Figure 1, we present the calculus LKF$^\perp$, which is almost the same with the following differences:

- All formulas are put on the left-hand side of the sequent, instead of the right-hand side. Therefore, one does not try to prove a disjunction of formulas, but one tries to refute a conjunction of formulas. This is the same thanks to the dual nature of classical first-order logic, and this helps to be closer to the resolution derivations. Note that, consequently, the focus is on negative formulas, and invertible rules are applied on positive formulas.
- The polarity of atoms is not chosen globally, but each *occurrence* of a literal can have a positive or a negative polarity. In particular, we can have two literals L and $L^\perp$ which are both negative, or both positive. We denote by $\underline{L}$ the fact that the literal L has a negative polarity. To be able to close branches on which we have two positive opposed literals, we add a rule $\widehat{\Uparrow}\vdash$.

We denote by $\Gamma \Uparrow \Delta \vdash$ (with Γ or Δ, possibly empty, containing polarized formulas) the fact that there exists a proof of the sequent $\Gamma \Uparrow \Delta \vdash$ in LKF$^\perp$, that is, a derivation starting from this sequent and whose branches are all closed (by $\widehat{\Downarrow}\vdash$, $\widehat{\Uparrow}\vdash$ or $\Downarrow\perp\vdash$). Thanks to focusing, such a proof has the following shape :

- Since one starts in an asynchronous ($\Uparrow$) phase, invertible rules are successively applied to the positive formulas of Δ, until one obtains negative formulas or literals that are put on the left of $\Uparrow$ using Store.
- When no formula appears on the right of $\Uparrow$, then either the branch is closed by $\widehat{\Uparrow}\vdash$; or the focus is put on a negative formula using Focus.
- In the latter case, one is now in synchronous ($\Downarrow$) phase where non-invertible rules are

successively applied to the formula upon which the focus is, until either the branch is closed using $\overset{\frown}{\Downarrow}\vdash$ or $\Downarrow\bot\vdash$; or one obtains a positive formula and the synchronous phase ends using Release.

- In the latter case, one starts again in the asynchronous phase.

Focusing therefore strongly constraints the shape of possible proofs, and therefore reduces the proof search space. The $\overset{\frown}{\Downarrow}\vdash$ in particular imposes to close branches immediately when the focus is on a negative literal, and thus rules out many derivations.

Note that proofs can be closed when the polarities of an atom and its negation are both positive (rule $\overset{\frown}{\Uparrow}\vdash$), or when one is positive and the other negative (rule $\overset{\frown}{\Downarrow}\vdash$), but not when they are both negative. Therefore, this restricts how formulas that contains literals with negative polarities can interact one with the others, and this is the main point of $\mathrm{LKF}^{\bot}$ to reduce the proof search space.

Restricting proof search using focusing leads to what are called synthetic rules (see for instance [13, pp.148–150] where they are called derived rules). The idea is to replace some formula A in the context of the sequent by new inference rules. Instead of proving the sequent $A, \Delta \vdash$ in $\mathrm{LKF}^{\bot}$, one proves $\Delta \vdash$ in ($\mathrm{LKF}^{\bot}$+ the synthetic rules obtained from A). Indeed, a proof focusing on A can only have certain shapes, and thus instead of having A in the context, it can be replaced by new rules synthesizing those shapes. For instance, the formula $\underline{P} \vee^{-} (Q \wedge^{+} \underline{R})$ in a context Γ can only lead to the following derivations when the focus is put on it:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \Downarrow P \vdash}\,\overset{\frown}{\Downarrow}\vdash \qquad \dfrac{\dfrac{\dfrac{\Gamma, Q \Uparrow \vdash}{\Gamma \Uparrow Q \vdash}\,\mathsf{Store}}{\Gamma \Downarrow Q \vdash}\,\mathsf{Release}}{\Gamma \Downarrow Q \wedge^{-} \underline{R} \vdash}\,\Downarrow\wedge_1\vdash}{\Gamma \Downarrow P \vee^{-} (Q \wedge^{-} \underline{R}) \vdash}\,\Downarrow\vee\vdash}{\Gamma \Uparrow \vdash}\,\mathsf{Focus}$$

and

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \Downarrow P \vdash}\,\overset{\frown}{\Downarrow}\vdash \qquad \dfrac{\dfrac{}{\Gamma \Downarrow \underline{R} \vdash}\,\overset{\frown}{\Downarrow}\vdash}{\Gamma \Downarrow Q \wedge^{-} \underline{R} \vdash}\,\Downarrow\wedge_2\vdash}{\Gamma \Downarrow P \vee^{-} (Q \wedge^{-} \underline{R}) \vdash}\,\Downarrow\vee\vdash}{\Gamma \Uparrow \vdash}\,\mathsf{Focus}$$

.

In the left derivation, $P^{\bot}$ must be in Γ to be able to close the left branch, so Γ is in fact of the form $\underline{P} \vee^{-} (Q \wedge^{+} \underline{R}), \Delta, P^{\bot}$. In the right one, Γ must be of the form $\underline{P} \vee^{-} (Q \wedge^{+} \underline{R}), \Delta, P^{\bot}, R^{\bot}$. Instead of searching for a proof with $\underline{P} \vee^{-} (Q \wedge^{+} \underline{R})$ in the context, the following two synthetic rules can therefore be used:

$$\mathsf{Syn1}\ \dfrac{\Delta, P^{\bot}, Q \Uparrow \vdash}{\Delta, P^{\bot} \Uparrow \vdash} \qquad\qquad \mathsf{Syn2}\ \dfrac{}{\Delta, P^{\bot}, R^{\bot} \Uparrow \vdash}$$

Provability is the same because each application of a synthetic rule can be replaced by applying Focus on $\underline{P} \vee^{-} (Q \wedge^{+} \underline{R})$ and following the derivation leading the synthetic rule, and vice versa. This is used for instance in provers based on the inverse method and focusing [27].

The sequent calculus $\mathrm{LKF}^{\bot}$ is not complete in general. One of the simplest examples of incompleteness is the sequent $\underline{P} \vee^{-} Q, \underline{\neg P} \vee^{-} Q, \neg Q \Uparrow \vdash$ which has no proof although $P \vee Q, \neg P \vee Q, \neg Q$ is not satisfiable.

3 Resolution with Input Selection

Two approaches can be used to reduce the proof search space of the resolution calculus: first, one can restrict on which pairs of clauses the resolution rule can be applied; this leads for instance to the set-of-support strategy [32], in which clauses are split into two sets, called the theory and the set of support; at least one of the clauses involved in a resolution step must be in the set of support. Second, one can restrict which literals in the clauses can be resolved

Resolution $\dfrac{L \curlyvee C \qquad L'^{\perp} \curlyvee D}{\sigma(C \curlyvee D)}$

- $\mathcal{S}(L \curlyvee C) = \emptyset$
- $\mathcal{S}(L'^{\perp} \curlyvee D) = \emptyset$
- σ is the most general unifier of $L =^? L'$

Factoring $\dfrac{L \curlyvee L' \curlyvee C}{\sigma(L \curlyvee C)}$

- $\mathcal{S}(L \curlyvee L' \curlyvee C) = \emptyset$
- σ is the most general unifier of $L =^? L'$

Resolution with Selection $\dfrac{\underline{K_1} \curlyvee \ldots \curlyvee \underline{K_n} \curlyvee C \qquad K_1'^{\perp} \curlyvee D_1 \qquad \ldots \qquad K_n'^{\perp} \curlyvee D_n}{\sigma(C \curlyvee D_1 \curlyvee \ldots \curlyvee D_n)}$

- $\mathcal{S}(\underline{K_1} \curlyvee \ldots \curlyvee \underline{K_n} \curlyvee C) = \{K_1; \ldots; K_n\}$
- $\mathcal{S}(K_i'^{\perp} \curlyvee D_i) = \emptyset$
- σ is the mgu of the simultaneous unification problem $K_1 =^? K_1', \ldots, K_n =^? K_n'$

Figure 2 Resolution with Input Selection.

upon; those literals are said to be selected in the clause. Resolution with free selection is complete for Horn clauses, but incomplete in general. Selecting a subset of the negative literals (if no literal is selected, then any literal of the clause can be used in resolution) is however complete, and combining this with an ordering restriction on clauses with no selected literals leads to Ordered Resolution with Selection, which was introduced by Bachmair and Ganzinger [2] (see also [3]) as a complete refinement of resolution.

Resolution with Input Selection combines these two approaches. It is parameterized by a selection function $\mathcal{S}$ that associate to each input clause a subset of its literals. If the selection function selects at least one literal, only those can be used in Resolution. Otherwise, any of them can be used. Note that for generated clauses, we impose that $\mathcal{S}(C) = \emptyset$. We also allow to have the same input clause several times with different selections. (That is, we actually work with couples composed of a clause and its selected literals.) The inference rules of Resolution with Input Selection are presented in Fig. 2. Literals that are selected in a clause are underlined. We will see that they indeed correspond to the literals that have a negative polarization in $\mathrm{LKF}^{\perp}$. As usual, variables are renamed in the clauses to avoid that premises of the inference rules share variables. We have two flavors of the resolution rule: the usual binary resolution, that is applied on two premises that do not select any literal; and Resolution with Selection that is applied on a clause in which n literals are selected and n clauses is which no literal is selected. Consequently, clauses with a non-empty selection cannot be resolved one with the others. By considering them as the theory part, and the clauses with an empty selection as the set of support, it is easy to see that Resolution with Input Selection is a generalization of the set-of-support strategy. Notwithstanding, note that neither Resolution with Input Selection is a generalization of Ordered Resolution with Selection nor the converse.

▶ **Definition 1** (Resolution derivation)**.** We write $\Gamma \rightsquigarrow C$ if C can be derived from some clauses in Γ using the inference rules Resolution with Selection, Resolution, or Factoring presented in Figure 2. We write $\Gamma \rightsquigarrow^* C$ if

- $C \in \Gamma$ or if
- there exists D such that $\Gamma \rightsquigarrow D$ and $\Gamma, D \rightsquigarrow^* C$.

As usual in resolution methods, the goal is to produce the empty clause $\square$ starting from a set of clauses Γ to show, since all rules are sound, that Γ is unsatisfiable. Here again, the calculus is not complete in general: from the set of clauses $\underline{P} \curlyvee Q, \underline{\neg P} \curlyvee Q, \neg Q$, no inference rule can be applied: to apply Resolution with Selection, we would need a clause where P, or $\neg P$, is not selected, and Resolution needs two clauses without selection.

4 LKF$^\perp$ is a Conservative Extension of Resolution with Input Selection

To link LKF$^\perp$ with Resolution with Input Selection, we need to indicate how clauses are related to polarized formulas.

▶ **Definition 2.** Given a clause $C = \underline{L_1} \curlyvee \cdots \curlyvee \underline{L_n} \curlyvee K_1 \curlyvee \ldots \curlyvee K_m$ whose free variables are $x_1, \ldots, x_l$ and such that $\mathcal{S}(C) = \{L_1; \ldots; L_n\}$, we define the associated formula $\ulcorner C \urcorner = \forall x_1, \ldots, x_l.\ \underline{L_1} \vee^- \cdots \vee^- \underline{L_n} \vee^- \delta^-(K_1 \vee^+ \cdots \vee^+ K_m)$. $\ulcorner C \urcorner$ is said to be in clausal form. By extension, $\ulcorner \Gamma \urcorner$ is the set of the formulas associated to the clauses of the set Γ.

The main theorem of this article relates LKF$^\perp$ with Resolution with Input Selection:

▶ **Theorem 3.** *Let Γ be a set of clauses. We have $\ulcorner \Gamma \urcorner \Uparrow \vdash$ iff $\Gamma \rightsquigarrow^* \square$.*

The proof can be found in the long version of the paper (`https://hal.inria.fr/hal-01670476`). To prove the right-to-left direction, we prove that all inference rules of Resolution with Input Selection are admissible in LKF$^\perp$, in the sense that if $\Gamma \rightsquigarrow C$ then LKF$^\perp$ proofs of $\ulcorner \Gamma \urcorner, \ulcorner C \urcorner \Uparrow \vdash$ can be turned into proofs of $\ulcorner \Gamma \urcorner \Uparrow \vdash$. Note that they are admissible, but they are not derivable. In particular, the size of the proof in LKF$^\perp$ can be much larger than the resolution derivation, as expected in a cut-free sequent calculus. Using cuts would lead to a closer correspondence between resolution derivations and sequent-calculus proofs, as in [15]. However, we chose to stay in the cut-free fragment to prove that, even in the incomplete case, resolution coincides with cut-free proofs, as in [25].

5 Complete Instances

5.1 Ordinary Focusing and Semantic Hyperresolution

As said earlier, in standard LKF, not all occurrences of literals can have an arbitrary polarity. Instead, each atom P is given globally a polarity, and $P^\perp$ has the opposite polarity.

Let us first look at the simple case where atoms are given a positive polarity. We recall the completeness proof of LKF:

▶ **Theorem 4** (Corollary of [26, Theorem 17])**.** *If the literals with a positive polarity are exactly the atoms, LKF$^\perp$ is (sound and) complete.*

If we look at the corresponding resolution calculus, Resolution with Selection for this particular instance becomes:

$$\text{R.w.S.}\ \frac{\neg P_1 \curlyvee \ldots \curlyvee \neg P_n \curlyvee C \qquad P'_1 \curlyvee D_1 \quad \ldots \quad P'_n \curlyvee D_n}{\sigma(C \curlyvee D_1 \curlyvee \ldots \curlyvee D_n)}$$

where C and D_i for all i contain only positive literals, and σ is the most general unifier of $P_1 =^? P'_1, \ldots, P_n =^? P'_n$. Note that the clause $\sigma(C \curlyvee D_1 \curlyvee \ldots \curlyvee D_n)$ contains only positive literals, so no literal would be selected in it even if it was an input clause. Besides, Resolution cannot be applied, since there exists no clause $\neg P \curlyvee C$ with $\mathcal{S}(\neg P \curlyvee C) = \emptyset$.

This corresponding resolution calculus is therefore exactly hyperresolution of [29]: premises of an inference contains all only positive literals, except one clause whose all negative literals are resolved at once. Theorem 3 therefore links ordinary focusing with hyperresolution. Consequently, Theorem 4 implies the completeness of hyperresolution.

Chaudhuri et al. [13, Theorem 16] prove a similar result by establishing a correspondence between hyperresolution derivations and proofs in a focused sequent calculus for intuitionistic

linear logic, but only considering Horn clauses. In their setting, choosing a negative polarity for atoms leads to SLD resolution, which is the reasoning mechanism of Prolog.

Let us now look at the general case, where atoms are given an arbitrary polarity. Let us stick to the ground case. We first recall a refinement of resolution called Semantic hyperresolution [31][11, Sect. 1.3.5.3]. Let I be an arbitrary Herbrand interpretation, i.e. a model whose domain is the set of terms interpreted as themselves. Note that I is not assumed to be a model of the input set of clauses (which is fortunate, since one is trying to show that it is unsatisfiable). Given a clause C, the idea of semantic hyperresolution is to resolve all literals of C that are valid in I at once, with clauses whose literals are all not valid in I. This gives the rule:

$$\mathsf{SHR}\ \frac{K_1 \curlyvee \ldots \curlyvee K_n \curlyvee C \qquad {K_1}^{\perp} \curlyvee D_1 \quad \ldots \quad {K_n}^{\perp} \curlyvee D_n}{C \curlyvee D_1 \curlyvee \ldots \curlyvee D_n}$$

where for all i, $I \models K_i$ (and thus $I \not\models K_i^{\perp}$), $I \not\models C$ and $I \not\models D_i$. Note that $I \not\models C \curlyvee D_1 \curlyvee \ldots \curlyvee D_n$.

Semantic hyperresolution for a Herbrand interpretation I can be seen as an instance of Resolution with Input Selection by using the following polarization of atoms: a literal L has a negative polarity iff $I \models L$. In that case, SHR corresponds exactly to Resolution with Selection, and Resolution cannot be applied since we cannot have clauses $P \curlyvee C$ and $\neg P \curlyvee D$ where both P and $\neg P$ are not valid in I.

This particular instance of polarization is in fact the ordinary version of focusing. Indeed, once a global polarity is assigned to each atom, the set of literals whose polarity is negative defines an Herbrand interpretation, and we saw reciprocally how to design a global polarization from the Herbrand interpretation. Theorem 3 therefore links ordinary focusing in the ground case with semantic hyperresolution. They are both complete, thanks to this theorem:

▶ **Theorem 5** (Corollary of [26, Theorem 17]). *Given a global polarization of atoms, where the polarity of $P^{\perp}$ is the opposite of that of P, $LKF^{\perp}$ is (sound and) complete.*

5.2 Deduction Modulo Theory

Deduction Modulo Theory [20] is a framework that consists in applying the inference rules of an existing proof system modulo some congruence over formulas. This congruence represents the theory, and it is in general defined by means of rewriting rules. To be expressive enough, these rules are defined not only at the term level, but also for formulas. To get simpler presentations of theories, we distinguish between rewrite rules that can be applied at positive and at negative positions by giving them a polarity[3], where by negative position we mean under an odd number of $\neg$. We therefore have positive rules $P \to^{+} A$ and negative rules $P \to^{-} A$ where P is an atom and A an arbitrary formula whose free variables appears in P. Given a rule $P \to^{+} A$, the rewrite relation $B_1 \xrightarrow{+} B_2$ is defined as usual by saying that there exists a position $\mathfrak{p}$ and a substitution σ such that the subformula of B_1 at position $\mathfrak{p}$ is σP and B_2 equals B_1 where the subformula at position $\mathfrak{p}$ is replaced by σA. $\xrightarrow{-}$ is defined similarly. In Polarized Sequent Calculus Modulo theory [17], the inference rules of the sequent calculus are applied modulo such a polarized rewriting system, as in for instance in $\vdash\wedge\ \dfrac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash C, \Delta}\ C \xrightarrow{+}{}^{*} A \wedge B$. Note that the implicit semantics of a negative

[3] This polarity must not be confused with the other notions of polarity mentioned in the paper.

$$\frown\vdash \frac{}{\Gamma, L, L^\perp \vdash} \qquad \top\vdash \frac{\Gamma \vdash}{\Gamma, \top \vdash} \qquad \perp\vdash \frac{}{\Gamma, \perp \vdash} \qquad \vee\vdash \frac{\Gamma, A \vdash \qquad \Gamma, B \vdash}{\Gamma, A \vee B \vdash}$$

$$\wedge\vdash \frac{\Gamma, A, B \vdash}{\Gamma, A \wedge B \vdash} \qquad \exists\vdash \frac{\Gamma, A \vdash}{\Gamma, \exists x.\ A \vdash}\ x \text{ not free in } \Gamma \qquad \forall\vdash \frac{\Gamma, \forall x.\ A, \{t/x\}A \vdash}{\Gamma, \forall x.\ A \vdash}$$

$$\uparrow^-\vdash \frac{\Gamma, P, A \vdash}{\Gamma, P \vdash}\ P \xrightarrow{-} A \qquad \uparrow^+\vdash \frac{\Gamma, \neg P, A^\perp \vdash}{\Gamma, \neg P \vdash}\ P \xrightarrow{+} A$$

Figure 3 The sequent calculus PUSC$^\perp$.

rule $P \to^- A$ is therefore $\overline{\forall x}.\ (P \Rightarrow A)$, whereas the semantics of $P \to^+ A$ is $\overline{\forall x}.\ (A \Rightarrow P)$, where $\overline{x}$ are the free variables of P.

With Kirchner [9], we proved the equivalence of Polarized Sequent Calculus Modulo theory to a sequent calculus where polarized rewriting rules are applied only on literals, using explicit rules. This calculus, Polarized Unfolding Sequent Calculus, is almost the calculus PUSC$^\perp$ presented in Figure 3. The only difference is that all formulas are put on the left of the sequent in PUSC$^\perp$. We denote by $\Gamma \vdash_{\mathcal{R}}$ the fact that $\Gamma \vdash$ can be proved in PUSC$^\perp$ using the polarized rewriting system $\mathcal{R}$. Note that the rule for the universal quantifier $\forall\vdash$ as well as the unfolding rules $\uparrow^-\vdash$ and $\uparrow^+\vdash$ contain an implicit contraction rule, as in the sequent calculus G4 of Kleene, in order to ensure that all rules of PUSC$^\perp$ are invertible.

We can translate polarized rewriting rules as formulas with selection, and see PUSC$^\perp$ as an instance of LKF$^\perp$. We first consider how to translate formulas of the right-hand side of polarized rewriting rules. We polarize them by choosing positive connectives for $\vee$ and $\wedge$ and, to unchain the introduction of the universal quantifier, we introduce delays. (Let us recall that a delay δ^+ allows to force a formula to be positive, and it can be encoded using an existantial quantifier.) This gives the translation:

$$|L| = L \quad \text{when } L \text{ is } \top, \perp \text{ or a literal} \qquad |A \wedge B| = |A| \wedge^+ |B|$$
$$|A \vee B| = |A| \vee^+ |B| \qquad |\exists x.\ A| = \exists x.\ |A| \qquad |\forall x.\ A| = \forall x.\ \delta^+|A|$$

▶ **Definition 6.** Given a negative rewriting rule $P \to^- A$ where the free variables of P are $x_1, \ldots, x_n$, its translation as a formula with selection is $[\![P \to^- A]\!] = \forall x_1 \ldots x_n.\ \underline{\neg P} \vee^- \delta^+|A|$.

Given a positive rewriting rule $P \to^+ A$ where the free variables of P are $x_1, \ldots, x_n$, its translation as a formula with selection is $[\![P \to^+ A]\!] = \forall x_1.\ \ldots \forall x_n.\ \underline{P} \vee^- \delta^+|A^\perp|$.

The translation $[\![\mathcal{R}]\!]$ of a polarized rewriting system $\mathcal{R}$ is the multiset of the translation of its rules.

▶ **Definition 7.** Let $N_1, \ldots, N_n$ be a multiset of formulas whose top connective is $\forall$ or $\perp$ or that are literals, and let $P_1, \ldots, P_m$ be a multiset of non-literal formulas whose top connective is neither $\forall$ nor $\perp$, then the translation of the PUSC$^\perp$ sequent $N_1, \ldots, N_n, P_1, \ldots, P_m \vdash$ modulo the rewriting system $\mathcal{R}$ is the LKF$^\perp$ sequent $[\![\mathcal{R}]\!], |N_1|, \ldots, |N_n| \Uparrow |P_1|, \ldots, |P_m| \vdash$.

▶ **Theorem 8.** *With the same assumptions as previous definition,* $N_1, \ldots, N_n, P_1, \ldots, P_m \vdash_{\mathcal{R}}$ *in PUSC*$^\perp$ *iff* $[\![\mathcal{R}]\!], |N_1|, \ldots, |N_n| \Uparrow |P_1|, \ldots, |P_m| \vdash$ *in LKF*$^\perp$.

The proof can be found in the long version of the paper.

Let us now consider the subcase where the rewriting rules are clausal, according to the terminology of [19], e.g. they are of the form $P \to^- C$ or $P \to^+ \neg C$ for some formula C in clausal normal form. In that case, the resolution method based on Deduction Modulo Theory [20] can be refined into what is called Polarized Resolution Modulo theory [19], whose rules are given in Fig. 4. (A refinement of) Polarized Resolution Modulo theory is actually implemented in the automated theorem prover iProverModulo [5].

$$\text{Resolution}\ \frac{P \curlyvee C \qquad \neg Q \curlyvee D}{\sigma(C \curlyvee D)}\ {}^{a} \qquad \text{Factoring}\ \frac{L \curlyvee K \curlyvee C}{\sigma(L \curlyvee C)}\ \sigma = mgu(L,K)$$

$$\text{Ext. Narr.}^{-}\ \frac{P \curlyvee C}{\sigma(D \curlyvee C)}\ {}^{a},\ Q \to^{-} D \qquad \text{Ext. Narr.}^{+}\ \frac{\neg Q \curlyvee D}{\sigma(C \curlyvee D)}\ {}^{a},\ P \to^{+} \neg C$$

[a] $\sigma = mgu(P,Q)$

Figure 4 Inference rules of Polarized Resolution Modulo theory.

By noting that the translation of the rule $Q \to^{-} D$ is $[\![Q \to^{-} D]\!] = \forall x_1.\ \ldots \forall x_n.\ \underline{\neg Q} \vee^{-} \delta^{+}|D|$ whereas $\ulcorner \underline{\neg Q} \curlyvee D \urcorner = \forall x_1.\ \ldots \forall x_n.\ \underline{\neg Q} \vee^{-} \delta^{-}|D|$, we can relate the rule $Q \to^{-} D$ with the clause with selection $\underline{\neg Q} \curlyvee D$, which is called a one-way clause by Dowek [19]. Indeed, the change of phase is always needed in that particular case, so that the delays are in fact useless. Ext. Narr.$^{-}$ can therefore be seen as an instance of the Resolution with Selection rule:

$$\text{Resolution with Selection}\ \frac{\underline{\neg Q} \curlyvee D \qquad P \curlyvee C}{\sigma(D \curlyvee C)}\ \sigma = mgu(P,Q)\ .$$

Similarly, $P \to^{+} \neg C$ is related to $\underline{P} \curlyvee C$.

Consequently, since PUSC$^{\perp}$ corresponds to LKF$^{\perp}$, and Resolution with Input Selection corresponds to Polarized Resolution Modulo theory, Theorem 3 leads to a new and more generic proof of the correspondence between PUSC$^{\perp}$ and Polarized Resolution Modulo theory.

Deduction Modulo Theory is not always complete. This is the case only if the cut rule is admissible in Polarized Sequent Calculus Modulo theory. It holds for some particular theories, e.g. Simple Type Theory [20] and arithmetic [22]. There are more or less powerful techniques that ensures this property [24, 21, 18, 7]. We even proved that any consistent first-order theory can be presented by a rewriting system admitting the cut rule [6]. As presented with Dowek [8] and discussed in the introduction, the fact that completeness is not proved once for all, but needs to be proved for each particular theory, is essential. Indeed, if a theory is presented entirely by rewriting rules, completeness implies the consistency of the theory, since no rule can be applied on the empty set of clauses. Consequently, the proof of the completeness cannot be easier than the proof of consistency of the theory, and, according to Gödel, cannot be proven in the theory itself.

5.3 Beyond Deduction Modulo Theory

We now consider the general case where several literals are selected in a clause, and show how proving completeness in LKF$^{\perp}$ can be reduced to proving completeness of several systems in Deduction Modulo Theory.

▶ **Example 9.** Let us recall the set of clauses from the Introduction:

$$\underline{\neg X \in \mathcal{P}(Y)} \curlyvee \underline{\neg Z \in X} \curlyvee Z \in Y \quad (1) \qquad \underline{X \in \mathcal{P}(Y)} \curlyvee d(X,Y) \in X \quad (2)$$

$$\underline{X \in \mathcal{P}(Y)} \curlyvee \underline{\neg d(X,Y) \in Y} \quad (3)$$

Note that this example is not covered by Ordered Resolution with Selection, at least not if a simplification ordering is used, because we cannot have $X \in \mathcal{P}(Y) \succ \delta(X,Y) \in X$ since with $\theta = \{X \mapsto \mathcal{P}(Z); Y \mapsto Z\}$ their instances are ordered in the wrong direction: $\mathcal{P}(Z) \in \mathcal{P}(Z) \prec \delta(\mathcal{P}(Z), Z) \in \mathcal{P}(Z)$.

The synthetic rules of the example from the Introduction correspond to the derivations when one of the clauses is focused. For instance, if we consider the clause (1), in a context

Γ containing this clause, a proof putting the focus on $\ulcorner(1)\urcorner$ necessarily is of the following shape:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \Downarrow \underline{\neg u \in \mathcal{P}(v)} \vdash}\,\widehat{\Downarrow}\vdash \quad \dfrac{}{\Gamma \Downarrow \underline{\neg t \in u} \vdash}\,\widehat{\Downarrow}\vdash \quad \dfrac{\dfrac{\Gamma, t \in v \Uparrow \vdash}{\Gamma \Uparrow t \in v \vdash}\,\text{Store}}{\Gamma \Downarrow t \in v \vdash}\,\text{Release}}{\Gamma \Downarrow \underline{\neg u \in \mathcal{P}(v)} \vee^- \underline{\neg t \in u} \vee^- t \in v \vdash}\,\Downarrow\vee\vdash}{\Gamma \Downarrow \forall X\ Y\ Z.\ \underline{\neg X \in \mathcal{P}(Y)} \vee^- \underline{\neg Z \in X} \vee^- Z \in Y \vdash}\,\Downarrow\forall\vdash}{\Gamma \Uparrow \vdash}\,\text{Focus}$$

where t, u, v are arbitrary terms, and where, to be able to close the left and middle branches, $u \in \mathcal{P}(v)$ and $t \in u$ must belong to Γ. So Γ is in fact of the form
$\forall X\ Y\ Z.\ \underline{\neg X \in \mathcal{P}(Y)} \vee^+ \underline{\neg Z \in X} \vee^+ Z \in Y, \Delta, u \in \mathcal{P}(v), t \in u$ for some Δ, and the axiom
$\forall X\ Y\ Z.\ \underline{\neg X \in \mathcal{P}(Y)} \vee^+ \underline{\neg Z \in X} \vee^+ Z \in Y$ can be replaced by the synthetic rule:

$$(1)\vdash \dfrac{\Delta, u \in \mathcal{P}(v), t \in u, t \in v \Uparrow \vdash}{\Delta, u \in \mathcal{P}(v), t \in u \Uparrow \vdash}\ .$$

The computation of the other synthetic rules is left as an exercise for the reader.

The resolution rules given in the Introduction corresponds to the ground instances of Resolution with Selection with our three input clauses.

The question that remains is how we can prove the completeness of such a selection. We can in fact consider only subselections.

▶ **Definition 10** (Singleton subselection)**.** Given a selection function $\mathcal{S}$, the selection function $\mathcal{S}_1$ is a singleton subselection of $\mathcal{S}$ if
- $\mathcal{S}_1(C) \subseteq \mathcal{S}(C)$ for all C
- if $\mathcal{S}(C) \neq \emptyset$ then $\text{card}(\mathcal{S}_1(C)) = 1$

▶ **Example 11.** A singleton subselection of Example 9 can be

$$\neg X \in \mathcal{P}(Y) \curlyvee \underline{\neg Z \in X} \curlyvee Z \in Y \qquad \underline{X \in \mathcal{P}(Y)} \curlyvee d(X,Y) \in X \qquad \underline{X \in \mathcal{P}(Y)} \curlyvee \neg d(X,Y) \in Y$$

▶ **Theorem 12.** *Resolution with input selection $\mathcal{S}$ is complete iff for all singleton subselections $\mathcal{S}_1$ of $\mathcal{S}$, Resolution with input selection $\mathcal{S}_1$ is complete.*

The proof can be found in the long version of the paper.

Since singleton subselections can be linked with rewriting systems in Deduction Modulo Theory according to last subsection, we can reduce the problem of completeness in our framework to several problems of completeness in Deduction Modulo Theory.

Conclusion and Further Work

We generalized focusing and resolution with selection, proved that they correspond, and showed how known calculi are instances of this framework, namely ordinary focusing, hyper-resolution and Deduction Modulo Theory. In the long version of the paper, other frameworks, such as Superdeduction [4] or Schroeder-Heister's Definitional reflection [30], are also considered. Furthermore, we presented how to reduce completeness of this framework to several completeness proofs in Deduction Modulo Theory. We can therefore reuse the various techniques for proving completeness in Deduction Modulo Theory [24, 21, 18, 7] in our framework. As Deduction Modulo Theory already gives significant results in industrial applications when the theory is a variant of set theory (more precisely, set theory of the B method) [10], we

can expect our framework to lead to even better outcomes. The notable results presented here raise the following new areas of investigations.

First, we need to study how to apply selection also in the generated clauses. This should allow us to cover the cases of Ordered Resolution with Selection and of Semantic Resolution in the first-order case. Dually, in the sequent calculus part, this would correspond to the possibility to dynamically add selection in formulas of subderivations. This could probably be linked with the work of Deplagne [16] where rewrite rules corresponding to induction hypotheses are dynamically added in the rewriting system of a sequent calculus for Deduction Modulo Theory. Note that we already have one direction, namely from Resolution with Input Selection to $\mathrm{LKF}^{\perp}$, since the proof for this direction (see the long version) does not assume anything on the generated clauses; except, for Factoring, that it selects only instances of literals that were already selected. The converse direction would require a meta-theorem of completeness, since obviously it is not complete for all possible dynamic choices of selection.

Since focusing is defined not only for classical first-order logic but also for linear, intuitionistic, modal logics, the work in this paper could serve as a starting point to study how to get automated proof search methods for these logics with a selection mechanism.

Another worthwhile point is how equality should be handled in our framework. In particular, it would be interesting to see how paramodulation calculi, in particular superposition, can be embedded into a sequent calculus.

Finally, it would be worth investigating whether completeness proofs based on model construction, such as semantic completeness proofs of tableaux (related to sequent calculus), and completeness proof of superposition [2], can be linked in our framework.

References

1 Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

2 L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):1–31, 1994.

3 Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 19–99. Elsevier and MIT Press, 2001.

4 Paul Brauner, Clément Houtmann, and Claude Kirchner. Principle of superdeduction. In Luke Ong, editor, *LICS*, pages 41–50, 2007.

5 Guillaume Burel. Experimenting with deduction modulo. In Viorica Sofronie-Stokkermans and Nikolaj Bjørner, editors, *CADE*, volume 6803 of *LNCS*, pages 162–176. Springer, 2011.

6 Guillaume Burel. From axioms to rewriting rules. Available on author's web page, 2013.

7 Guillaume Burel. Cut admissibility by saturation. In Gilles Dowek, editor, *RTA-TLCA*, volume 8560 of *LNCS*, pages 124–138. Springer, 2014.

8 Guillaume Burel and Gilles Dowek. How can we prove that a proof search method is not an instance of another? In *LFMTP*, ACM International Conference Proceeding Series, pages 84–87. ACM, 2009.

9 Guillaume Burel and Claude Kirchner. Regaining cut admissibility in deduction modulo using abstract completion. *Information and Computation*, 208(2):140–164, 2010.

10 Guillaume Bury, David Delahaye, Damien Doligez, Pierre Halmagrand, and Olivier Hermant. Automated deduction in the B set theory using typed proof search and deduction modulo. In Ansgar Fehnker, Annabelle McIver, Geoff Sutcliffe, and Andrei Voronkov, editors, *LPAR*, volume 35 of *EPiC Series in Computing*, pages 42–58. EasyChair, 2015.

11 Samuel R. Buss, editor. *Handbook of proof theory*. Studies in logic and the foundations of mathematics. Elsevier, Amsterdam, 1998.

12 Kaustuv Chaudhuri and Frank Pfenning. A focusing inverse method theorem prover for first-order linear logic. In Robert Nieuwenhuis, editor, *CADE*, volume 3632 of *LNCS*, pages 69–83. Springer, 2005.

13 Kaustuv Chaudhuri, Frank Pfenning, and Greg Price. A logical characterization of forward and backward chaining in the inverse method. *Journal of Automated Reasoning*, 40(2-3):133–177, 2008.

14 Zakaria Chihani, Tomer Libal, and Giselle Reis. The proof certifier checkers. In *TABLEAUX*, volume 9323 of *LNCS*, pages 201–210, Wroclaw, Poland, 2015. Springer.

15 Zakaria Chihani, Dale Miller, and Fabien Renaud. Foundational proof certificates in first-order logic. In Maria Paola Bonacina, editor, *CADE*, volume 7898 of *LNCS*, pages 162–177. Springer, 2013.

16 Eric Deplagne and Claude Kirchner. Induction as deduction modulo. Rapport de recherche, LORIA, Nov 2004. URL: `https://hal.inria.fr/LORIA/inria-00099871`.

17 Gilles Dowek. What is a theory? In Helmut Alt and Afonso Ferreira, editors, *STACS*, volume 2285 of *LNCS*, pages 50–64. Springer, 2002.

18 Gilles Dowek. Truth values algebras and proof normalization. In Thorsten Altenkirch and Conor McBride, editors, *TYPES*, volume 4502 of *LNCS*, pages 110–124. Springer, 2006.

19 Gilles Dowek. Polarized resolution modulo. In Cristian S. Calude and Vladimiro Sassone, editors, *IFIP TCS*, volume 323 of *IFIP AICT*, pages 182–196. Springer, 2010.

20 Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem proving modulo. *Journal of Automated Reasoning*, 31(1):33–72, 2003.

21 Gilles Dowek and Benjamin Werner. Proof normalization modulo. *The Journal of Symbolic Logic*, 68(4):1289–1316, 2003.

22 Gilles Dowek and Benjamin Werner. Arithmetic as a theory modulo. In Jürgen Giesl, editor, *RTA*, volume 3467 of *LNCS*, pages 423–437. Springer, 2005.

23 Mahfuza Farooque, Stéphane Graham-Lengrand, and Assia Mahboubi. A bisimulation between DPLL(T) and a proof-search strategy for the focused sequent calculus. In Alberto Momigliano, Brigitte Pientka, and Randy Pollack, editors, *LFMTP*, pages 3–14. ACM, 2013.

24 Olivier Hermant. *Méthodes Sémantiques en Déduction Modulo*. PhD thesis, École Polytechnique, 2005.

25 Olivier Hermant. Resolution is cut-free. *Journal of Automated Reasoning*, 44(3):245–276, 2009.

26 Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46):4747–4768, 2009. Abstract Interpretation and Logic Programming: In honor of professor Giorgio Levi.

27 Sean McLaughlin and Frank Pfenning. Imogen: Focusing the polarized inverse method for intuitionistic propositional logic. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *LPAR*, volume 5330 of *Lecture Notes in Computer Science*, pages 174–181. Springer, 2008.

28 Dale Miller and Marco Volpe. Focused labeled proof systems for modal logic. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *LPAR*, volume 9450 of *LNCS*, pages 266–280. Springer, 2015.

29 J. A. Robinson. Automatic deduction with hyper-resolution. *International Journal of Computer Mathematics*, 1:227–234, 1965.

30 Peter Schroeder-Heister. Cut elimination for logics with definitional reflection. In *Non-classical Logics and Information Processing*, volume 619 of *LNCS*, pages 146–171. Springer, 1990.

31 James R. Slagle. Automatic theorem proving with renamable and semantic resolution. *J. ACM*, 14(4):687–697, 1967.

32 Larry Wos, George A. Robinson, and Daniel F. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *J. ACM*, 12(4):536–541, 1965.

Team Semantics for the Specification and Verification of Hyperproperties

Andreas Krebs
Universität Tübingen, Wilhelm-Schickard-Institut für Informatik, Tübingen, Germany
krebs@informatik.uni-tuebingen.de

Arne Meier[1]
Leibniz Universität Hannover, Institut für Theoretische Informatik, Hannover, Germany
meier@thi.uni-hannover.de
https://orcid.org/0000-0002-8061-5376

Jonni Virtema
Hasselt University, Databases and Theoretical Computer Science Group, Diepenbeek, Belgium
jonni.virtema@uhasselt.be
https://orcid.org/0000-0002-1582-3718

Martin Zimmermann[2]
Saarland University, Reactive Systems Group, Saarbrücken, Germany
zimmermann@react.uni-saarland.de

Abstract

We develop team semantics for Linear Temporal Logic (LTL) to express hyperproperties, which have recently been identified as a key concept in the verification of information flow properties. Conceptually, we consider an asynchronous and a synchronous variant of team semantics. We study basic properties of this new logic and classify the computational complexity of its satisfiability, path, and model checking problem. Further, we examine how extensions of these basic logics react on adding other atomic operators. Finally, we compare its expressivity to the one of HyperLTL, another recently introduced logic for hyperproperties. Our results show that LTL under team semantics is a viable alternative to HyperLTL, which complements the expressivity of HyperLTL and has partially better algorithmic properties.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic

Keywords and phrases LTL, Hyperproperties, Team Semantics, Model Checking, Satisfiability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.10

Related Version [21], `https://arxiv.org/abs/1709.08510`

Acknowledgements We thank Alexander Weinert for numerous fruitful discussions.

1 Introduction

Guaranteeing security and privacy of user information is a key requirement in software development. However, it is also one of the hardest goals to accomplish. One reason for this difficulty is that such requirements typically amount to reasoning about the flow of information and relating different execution traces of the system. In particular, these

[1] Funded by the German Research Foundation DFG, project ME 4279/1-2.
[2] Funded by the German Research Foundation DFG, project "TriCS" (ZI 1516/1-1).

requirements are no longer trace properties, i.e., properties whose satisfaction can be verified by considering each trace in isolation. For example, the property "the system terminates eventually" is satisfied if every trace eventually reaches a final state. Formally, a trace property φ is a set of traces and a system satisfies φ if each of its traces is in φ.

In contrast, the property "the system terminates within a bounded amount of time" is no longer a trace property; consider a system that has a trace t_n for every n, so that t_n only reaches a final state after n steps. This system does not satisfy the bounded termination property, but each individual trace t_n could also stem from a system that does satisfy it. Thus, satisfaction of the property cannot be verified by considering each trace in isolation.

Properties with this characteristic were termed *hyperproperties* by Clarkson and Schneider [6]. Formally, a hyperproperty φ is a set of sets of traces and a system satisfies φ if its set of traces is contained in φ. The conceptual difference to trace properties allows specifying a much richer landscape of properties including information flow and trace properties. Further, one can also express specifications for symmetric access to critical resources in distributed protocols and Hamming distances between code words in coding theory [29]. However, the increase in expressiveness requires novel approaches to specification and verification.

HyperLTL. Trace properties are typically specified in temporal logics, most prominently in Linear Temporal Logic (LTL) [28]. Verification of LTL specifications is routinely employed in industrial settings and marks one of the most successful applications of formal methods to real-life problems. Recently, this work has been extended to hyperproperties: HyperLTL, LTL equipped with trace quantifiers, has been introduced to specify hyperproperties [5]. Accordingly, a model of a HyperLTL formula is a set of traces and the quantifiers range over these traces. This logic is able to express the majority of the information flow properties found in the literature (we refer to Section 3 of [5] for a full list). The satisfiability problem for HyperLTL is undecidable [10] while the model checking problem is decidable, albeit of non-elementary complexity [5, 13]. In view of this, the full logic is too strong. Fortunately most information flow properties found in the literature can be expressed with at most one quantifier alternation and consequently belong to decidable (and tractable) fragments. Further works have studied runtime verification [2, 11], connections to first-order logic [14], provided tool support [13, 10], and presented applications to "software doping" [7] and the verification of web-based workflows [12]. In contrast, there are natural properties, e.g., bounded termination, which are not expressible in HyperLTL (which is an easy consequence of a much stronger non-expressibility result [3]).

Team Semantics. Intriguingly, there exists another modern family of logics, *Dependence Logics* [32, 9], which operate as well on sets of objects instead of objects alone. Informally, these logics extend first-order logic (FO) by atoms expressing, e.g., that "the value of a variable x functionally determines the value of a variable y" or that "the value of a variable x is informationally independent of the value of a variable y". Obviously, such statements only make sense when being evaluated over a set of assignments. In the language of dependence logic, such sets are called *teams* and the semantics is termed *team semantics*.

In 1997, Hodges introduced compositional semantics for Hintikka's Independence-friendly logic [19]. This can be seen as the cornerstone of the mathematical framework of dependence logics. Intuitively, this semantics allows for interpreting a team as a database table. In this approach, variables of the table correspond to attributes and assignments to rows or records. In 2007, Väänänen [32] introduced his modern approach to such logics and adopted team semantics as a core notion, as dependence atoms are meaningless under Tarskian semantics.

After the introduction of dependence logic, a whole family of logics with different atomic statements have been introduced in this framework: *independence logic* [17] and *inclusion logic* [15] being the most prominent. Interest in these logics is rapidly growing and the research community aims to connect their area to a plethora of disciplines, e.g., linguistics [16], biology [16], game [4] and social choice theory [30], philosophy [30], and computer science [16]. We are the first to exhibit connections to formal languages via application of Büchi automata (see Theorem 4.3). Team semantics has also found their way into modal [33] and temporal logic [20], as well as statistics [8].

Recently, Krebs et al. [20] proposed team semantics for Computation Tree Logic (CTL), where a team consists of worlds of the transition system under consideration. They considered synchronous and asynchronous team semantics, which differ in how time evolves in the semantics of the temporal operators. They proved that satisfiability is EXPTIME-complete under both semantics while model checking is PSPACE-complete under synchronous semantics and P-complete under asynchronous semantics.

Our Contribution. The conceptual similarities between HyperLTL and team semantics raise the question how an LTL variant under team semantics relates to HyperLTL. For this reason, we develop team semantics for LTL, analyse the complexity of its satisfiability and model checking problems, and subsequently compare the novel logic to HyperLTL.

When defining the logic, we follow the approach of Krebs et al. [20] for defining team semantics for CTL: we introduce synchronous and asynchronous team semantics for LTL, where teams are now sets of traces. In particular, as a result, we have to consider potentially uncountable teams, while all previous work on model checking problems for logics under team semantics has been restricted to the realm of finite teams.

We prove that the satisfiability problem for team LTL is PSPACE-complete under both semantics, by showing that the problems are equivalent to LTL satisfiability under classical semantics. Generally, we observe that for the basic asynchronous variant all of our investigated problems trivially reduce to and from classical LTL semantics. However, for the synchronous semantics this is not the case for two variants of the model checking problem. As there are uncountably many traces, we have to represent teams, i.e., sets of traces, in a finitary manner. The path checking problem asks to check whether a finite team of ultimately periodic traces satisfies a given formula. As our main result, we establish this problem to be PSPACE-complete for synchronous semantics. In the (general) model checking problem, a team is represented by a finite transition system. Formally, given a transition system and a formula, the model checking problem asks to determine whether the set of traces of the system satisfies the formula. For the synchronous case we give a polynomial space algorithm for the model checking problem for the disjunction-free fragment, while we leave open the complexity of the general problem. Disjunction plays a special role in team semantics, as it splits a team into two. As a result, this operator is commonly called *splitjunction* instead of disjunction. In our setting, the splitjunction requires us to deal with possibly infinitely many splits of uncountable teams, if a splitjunction is under the scope of a G-operator, which raises interesting language-theoretic questions.

Further, we study the effects for complexity that follow when our logics are extended by dependence atoms and the contradictory negation. Finally, we show that LTL under team semantics is able to specify properties which are not expressible in HyperLTL and *vice versa*.

Recall that satisfiability for HyperLTL is undecidable and model checking of non-elementary complexity. Our results show that similar problems for LTL under team semantics have a much simpler complexity while some hyperproperties are still expressible (e.g., input

determinism, see page 11, or bounded termination). This proposes LTL under team semantics to be a significant alternative for the specification and verification of hyperproperties that complements HyperLTL.

2 Preliminaries

The non-negative integers are denoted by $\mathbb{N}$ and the power set of a set S is denoted by 2^S. Throughout the paper, we fix a finite set AP of atomic propositions.

Computational Complexity. We will make use of standard notions in complexity theory. In particular, we will use the complexity classes P and PSPACE. Most reductions used in the paper are $\leq_{\mathrm{m}}^{\mathrm{p}}$-reductions, that is, polynomial time, many-to-one reductions.

Traces. A *trace* over AP is an infinite sequence from $(2^{\mathrm{AP}})^\omega$; a finite trace is a finite sequence from $(2^{\mathrm{AP}})^*$. The length of a finite trace t is denoted by $|t|$. The empty trace is denoted by ε and the concatenation of two finite traces t_0 and t_1 by t_0t_1. Unless stated otherwise, a trace is always assumed to be infinite. A *team* is a (potentially infinite) set of traces.

Given a trace $t = t(0)t(1)t(2)\cdots$ and $i \geq 0$, we define $t[i,\infty) := t(i)t(i+1)t(i+2)\cdots$, which we lift to teams $T \subseteq (2^{\mathrm{AP}})^\omega$ by defining $T[i,\infty) := \{t[i,\infty) \mid t \in T\}$. A trace t is *ultimately periodic*, if it is of the form $t = t_0 \cdot t_1^\omega = t_0t_1t_1t_1\cdots$ for two finite traces t_0 and t_1 with $|t_1| > 0$. As a result, an ultimately periodic trace t is finitely represented by the pair (t_0, t_1); we define $[\![(t_0,t_1)]\!] = t_0t_1^\omega$. Given a set $\mathcal{T}$ of such pairs, we define $[\![\mathcal{T}]\!] = \{[\![(t_0,t_1)]\!] \mid (t_0,t_1) \in \mathcal{T}\}$, which is a team of ultimately periodic traces. We call $\mathcal{T}$ a team encoding of $[\![\mathcal{T}]\!]$.

Linear Temporal Logic. The formulas of Linear Temporal Logic (LTL) [28] are defined via the grammar $\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \mathsf{F}\varphi \mid \mathsf{G}\varphi \mid \varphi\mathsf{U}\varphi \mid \varphi\mathsf{R}\varphi$, where p ranges over the atomic propositions in AP. The length of a formula is defined to be the number of Boolean and temporal connectives occurring in it. The length of an LTL formula is often defined to be the number of syntactically different subformulas, which might be exponentially smaller. Here, we need to distinguish syntactically equal subformulas which becomes clearer after defining the semantics (see also Example 2.1 afterwards on this). As we only consider formulas in negation normal form, we use the full set of temporal operators.

Next, we recall the classical semantics of LTL before we introduce team semantics. For traces $t \in (2^{\mathrm{AP}})^\omega$ we define

$$
\begin{aligned}
t &\models^{\mathrm{c}} p && \text{if } p \in t(0),\\
t &\models^{\mathrm{c}} \neg p && \text{if } p \notin t(0),\\
t &\models^{\mathrm{c}} \psi \wedge \varphi && \text{if } t \models^{\mathrm{c}} \psi \text{ and } t \models^{\mathrm{c}} \varphi,\\
t &\models^{\mathrm{c}} \psi \vee \varphi && \text{if } t \models^{\mathrm{c}} \psi \text{ or } t \models^{\mathrm{c}} \varphi,\\
t &\models^{\mathrm{c}} \mathsf{X}\varphi && \text{if } t[1,\infty) \models^{\mathrm{c}} \varphi,\\
t &\models^{\mathrm{c}} \mathsf{F}\varphi && \text{if } \exists k \geq 0 : t[k,\infty) \models^{\mathrm{c}} \varphi,\\
t &\models^{\mathrm{c}} \mathsf{G}\varphi && \text{if } \forall k \geq 0 : t[k,\infty) \models^{\mathrm{c}} \varphi,\\
t &\models^{\mathrm{c}} \psi\mathsf{U}\varphi && \text{if } \exists k \geq 0 : t[k,\infty) \models^{\mathrm{c}} \varphi \text{ and}\\
& && \quad \forall k' < k : t[k',\infty) \models^{\mathrm{c}} \psi,\\
t &\models^{\mathrm{c}} \psi\mathsf{R}\varphi && \text{if } \forall k \geq 0 : t[k,\infty) \models^{\mathrm{c}} \varphi \text{ or}\\
& && \quad \exists k' < k : t[k',\infty) \models^{\mathrm{c}} \psi.
\end{aligned}
$$

Team Semantics for LTL. Next, we introduce two variants of team semantics for LTL, which differ in their interpretation of the temporal operators: a synchronous semantics ($\models^{\mathrm{s}}$), where time proceeds in lockstep along all traces of the team, and an asynchronous semantics ($\models^{\mathrm{a}}$) in which, on each trace of the team, time proceeds independently. We write $\models^{\star}$ whenever

property	definition	$\models^{a}$	$\models^{s}$
empty team property	$\emptyset \models^{\star} \varphi$	✓	✓
downwards closure	$T \models^{\star} \varphi$ implies $\forall T' \subseteq T$: $T' \models^{\star} \varphi$	✓	✓
union closure	$T \models^{\star} \varphi$, $T' \models^{\star} \varphi$ implies $T \cup T' \models^{\star} \varphi$	✓	×
flatness	$T \models^{\star} \varphi$ if and only if $\forall t \in T$: $\{t\} \models^{\star} \varphi$	✓	×
singleton equivalence	$\{t\} \models^{\star} \varphi$ if and only if $t \models^{c} \varphi$	✓	✓

Figure 1 Structural properties overview.

a definition coincides for both semantics. For teams $T \subseteq (2^{\mathrm{AP}})^{\omega}$ let

$T \models^{\star} p$ if $\forall t \in T : p \in t(0)$,
$T \models^{\star} \neg p$ if $\forall t \in T : p \notin t(0)$,
$T \models^{\star} \psi \wedge \varphi$ if $T \models^{\star} \psi$ and $T \models^{\star} \varphi$,
$T \models^{\star} \psi \vee \varphi$ if $\exists T_1 \cup T_2 = T : T_1 \models^{\star} \psi$ and $T_2 \models^{\star} \varphi$,
$T \models^{\star} \mathsf{X}\varphi$ if $T[1, \infty) \models^{\star} \varphi$.

This concludes the cases where both semantics coincide. Next, we present the remaining cases for the synchronous semantics, which are inherited from the classical semantics of LTL.

$T \models^{s} \mathsf{F}\varphi$ if $\exists k \geq 0 : T[k, \infty) \models^{s} \varphi$,
$T \models^{s} \mathsf{G}\varphi$ if $\forall k \geq 0 : T[k, \infty) \models^{s} \varphi$,
$T \models^{s} \psi \mathsf{U} \varphi$ if $\exists k \geq 0 : T[k, \infty) \models^{s} \varphi$ and $\forall k' < k : T[k', \infty) \models^{s} \psi$, and
$T \models^{s} \psi \mathsf{R} \varphi$ if $\forall k \geq 0 : T[k, \infty) \models^{s} \varphi$ or $\exists k' < k : T[k', \infty) \models^{s} \psi$.

Finally, we present the remaining cases for the asynchronous semantics. Note that, here there is no unique timepoint k, but a timepoint k_t for every trace t, i.e., time evolves asynchronously between different traces.

$T \models^{a} \mathsf{F}\varphi$ if $\exists k_t \geq 0$, for each $t \in T$: $\{t[k_t, \infty) \mid t \in T\} \models^{a} \varphi$
$T \models^{a} \mathsf{G}\varphi$ if $\forall k_t \geq 0$, for each $t \in T$: $\{t[k_t, \infty) \mid t \in T\} \models^{a} \varphi$,
$T \models^{a} \psi \mathsf{U} \varphi$ if $\exists k_t \geq 0$, for each $t \in T$: $\{t[k_t, \infty) \mid t \in T\} \models^{a} \varphi$, and
$\forall k'_t < k_t$, for each $t \in T$: $\{t[k'_t, \infty) \mid t \in T\} \models^{a} \psi$, and
$T \models^{a} \psi \mathsf{R} \varphi$ if $\forall k_t \geq 0$, for each $t \in T$: $\{t[k_t, \infty) \mid t \in T\} \models^{a} \varphi$ or
$\exists k'_t < k_t$, for each $t \in T$: $\{t[k'_t, \infty) \mid t \in T\} \models^{a} \psi$.

We call expressions of the form $\psi \vee \varphi$ *splitjunctions* to emphasise on the team semantics where we split a team into two parts. Similarly, the $\vee$-operator is referred to as a *splitjunction*.

Let us illustrate the difference between synchronous and asynchronous semantics with an example involving the F operator. Similar examples can be constructed for the other temporal operators (but for X) as well.

▶ **Example 2.1.** Let $T = \{\{p\}\emptyset^{\omega}, \emptyset\{p\}\emptyset^{\omega}\}$. We have that $T \models^{a} \mathsf{F}p$, as we can pick $k_t = 0$ if $t = \{p\}\emptyset^{\omega}$, and $k_t = 1$ if $t = \emptyset\{p\}\emptyset^{\omega}$. On the other hand, there is no single k such that $T[k, \infty) \models^{s} p$, as the occurrences of p are at different positions. Consequently $T \not\models^{s} \mathsf{F}p$.

Moreover, consider the formula $\mathsf{F}p \vee \mathsf{F}p$ which is satisfied by T on both semantics. However, $\mathsf{F}p$ is not satisfied by T under synchronous semantics. Accordingly, we need to distinguish the two disjuncts $\mathsf{F}p$ and $\mathsf{F}p$ of $\mathsf{F}p \vee \mathsf{F}p$ to assign them to different teams.

In contrast, synchronous satisfaction implies asynchronous satisfaction, i.e., $T \models^{s} \varphi$ implies $T \models^{a} \varphi$. The simplest way to prove this is by applying downward closure, singleton equivalence, and flatness (see Fig. 1). Example 2.1 shows that the converse does not hold.

Next, we define the most important verification problems for LTL in team semantics setting, namely satisfiability and two variants of the model checking problem: For classical

LTL, one studies the path checking problem and the model checking problem. The difference between these two problems lies in the type of structures one considers. Recall that a model of an LTL formula is a single trace. In the path checking problem, a trace t and a formula φ are given, and one has to decide whether $t \models^{c} \varphi$. This problem has applications to runtime verification and monitoring of reactive systems [23, 26]. In the model checking problem, a Kripke structure $\mathcal{K}$ and a formula φ are given, and one has to decide whether every execution trace t of $\mathcal{K}$ satisfies φ.

The satisfiability problem of LTL under team semantics is defined as follows.

Problem: LTL satisfiability w.r.t. teams (TSAT*) for $\star \in \{\mathrm{a}, \mathrm{s}\}$.
Input: LTL formula φ.
Question: Is there a non-empty team T such that $T \models^{\star} \varphi$?

The non-emptiness condition is necessary, as otherwise every formula is satisfiable due to the empty team property (see Fig. 1).

We consider the generalisation of the path checking problem for LTL (denoted by LTL-PC), which asks for a given ultimately periodic trace t and a given formula φ, whether $t \models^{c} \varphi$ holds. In the team semantics setting, the corresponding question is whether a given finite team comprised of ultimately periodic traces satisfies a given formula. Such a team is given by a team encoding $\mathcal{T}$. To simplify our notation, we will write $\mathcal{T} \models^{\star} \varphi$ instead of $[\![\mathcal{T}]\!] \models^{\star} \varphi$.

Problem: TeamPathChecking (TPC*) for $\star \in \{\mathrm{a}, \mathrm{s}\}$.
Input: LTL formula φ and a finite team encoding $\mathcal{T}$.
Question: $\mathcal{T} \models^{\star} \varphi$?

Consider the generalised model checking problem where one checks whether the team of traces of a Kripke structure satisfies a given formula. This is the natural generalisation of the model checking problem for classical semantics, denoted by LTL-MC, which asks, for a given Kripke structure $\mathcal{K}$ and a given LTL formula φ, whether $t \models^{c} \varphi$ for every trace t of $\mathcal{K}$.

A Kripke structure $\mathcal{K} = (W, R, \eta, w_I)$ consists of a finite set W of worlds, a left-total transition relation $R \subseteq W \times W$, a labeling function $\eta \colon W \to 2^{\mathrm{AP}}$, and an initial world $w_I \in W$. A path π through $\mathcal{K}$ is an infinite sequence $\pi = \pi(0)\pi(1)\pi(2)\cdots \in W^\omega$ such that $\pi(0) = w_I$ and $(\pi(i), \pi(i+1)) \in R$ for every $i \geq 0$. The trace of π is defined as $t(\pi) = \eta(\pi(0))\eta(\pi(1))\eta(\pi(2))\cdots \in (2^{\mathrm{AP}})^\omega$. The Kripke structure $\mathcal{K}$ induces the team $T(\mathcal{K}) = \{t(\pi) \mid \pi \text{ is a path through } \mathcal{K}\}$.

Problem: TeamModelChecking (TMC*) for $\star \in \{\mathrm{a}, \mathrm{s}\}$.
Input: LTL formula φ and a Kripke structure $\mathcal{K}$.
Question: $T(\mathcal{K}) \models^{\star} \varphi$?

3 Basic Properties

We consider several standard properties of team semantics (cf., e.g. [9]) and verify which of these hold for our two semantics for LTL. These properties are later used to analyse the complexity of the satisfiability and model checking problems. To simplify our notation, $\models^{\star}$ denotes $\models^{\mathrm{a}}$ or $\models^{\mathrm{s}}$. See Figure 1 for the definitions of the properties and a summary for which semantics the properties hold. The positive results follow via simple inductive arguments. For the fact that synchronous semantics is not union closed, consider teams $T = \{\{p\}\emptyset^\omega\}$ and $T' = \{\emptyset\{p\}\emptyset^\omega\}$. Then, we have $T \models^{\mathrm{s}} \mathsf{F}p$ and $T' \models^{\mathrm{s}} \mathsf{F}p$ but $T \cup T' \not\models^{\mathrm{s}} \mathsf{F}p$. Note also that flatness is equivalent of being both downward and union closed.

It turns out that, by Figure 1, LTL under asynchronous team semantics is essentially classical LTL with a bit of universal quantification: for a team T and an LTL-formula φ,

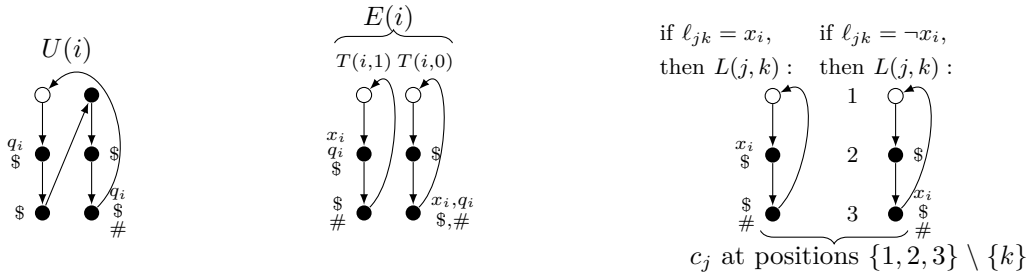


Figure 2 Traces for the reduction defined in the proof of Lemma 4.1.

we have $T \models^{\mathrm{a}} \varphi$ if and only if $\forall t \in T : t \models^{\mathrm{c}} \varphi$. This however does not mean that LTL under asynchronous team semantics is not worth of a study; it only means that asynchronous LTL is essentially classical LTL if we do not introduce additional atomic formulas that describe properties of teams directly. This is a common phenomenon in the team semantics setting. For instance, team semantics of first-order logic has the flatness property, but its extension by so-called *dependence atoms*, is equi-expressive with existential second-order logic [32]. Extensions of LTL under team semantics are discussed in Section 5.

At this point, it should not come as a surprise that, due to the flatness property and singleton equivalence, the complexity of satisfiability, path checking, and model checking for LTL under asynchronous team semantics coincides with those of classical LTL semantics. Firstly, note that an LTL-formula φ is satisfiable under asynchronous or synchronous team semantics if and only if there is a singleton team that satisfies the formula. Secondly, note that to check whether a given team satisfies φ under asynchronous semantics, it is enough to check whether each trace in the team satisfies φ under classical LTL; this can be computed by an AC^0-circuit using oracle gates for LTL-PC. Putting these observations together, we obtain the following results from the identical results for LTL under classical semantics [22, 23, 26, 31].

The circuit complexity class AC^i encompass of polynomial sized circuits of depth $O(\log^i(n))$ and unbounded fan-in; NC^i is similarly defined but with bounded fan-in. A language A is *constant-depth reducible* to a language B, in symbols $A \leq_{\mathrm{cd}} B$, if there exists a logtime-uniform AC^0-circuit family with oracle gates for B that decides membership in A. In this context, *logtime-uniform* means that there exists a deterministic Turing machine that can check the structure of the circuit family $\mathcal{C}$ in time $O(\log |\mathcal{C}|)$. For further information on circuit complexity, we refer the reader to the textbook of Vollmer [35]. Furthermore, logDCFL is the set of languages which are logspace reducible to a deterministic context-free language.

▶ **Proposition 3.1.**
1. $\mathrm{TMC}^{\mathrm{a}}$, $\mathrm{TSAT}^{\mathrm{a}}$, *and* $\mathrm{TSAT}^{\mathrm{s}}$ *are* PSPACE*-complete w.r.t.* $\leq^{\mathrm{p}}_{\mathrm{m}}$*-reductions.*
2. $\mathrm{TPC}^{\mathrm{a}}$ *is in* $\mathrm{AC}^1(\mathrm{logDCFL})$ *and* NC^1*-hard w.r.t.* $\leq_{\mathrm{cd}}$*-reductions.*

4 Classification of Decision Problems Under Synchronous Semantics

In this section, we examine the computational complexity of path and model checking with respect to the synchronous semantics. Our main result settles the complexity of $\mathrm{TPC}^{\mathrm{s}}$. It turns out that this problem is harder than the asynchronous version.

▶ **Lemma 4.1.** $\mathrm{TPC}^{\mathrm{s}}$ *is* PSPACE*-hard w.r.t.* $\leq^{\mathrm{p}}_{\mathrm{m}}$*-reductions.*

Proof. Determining whether a given quantified Boolean formula (qBf) is valid (QBF-VAL) is a well-known PSPACE-complete problem [25]. The problem stays PSPACE-complete if the matrix (i.e., the propositional part) of the given qBf is in 3CNF. To prove the claim of the lemma, we will show that QBF-VAL $\leq_{\mathrm{m}}^{\mathrm{p}}$ TPC$^{\mathrm{s}}$. Given a quantified Boolean formula φ, we stipulate, w.l.o.g., that φ is of the form $\exists x_1 \forall x_2 \cdots Q x_n \chi$, where $\chi = \bigwedge_{j=1}^{m} \bigvee_{k=1}^{3} \ell_{jk}$, $Q \in \{\exists, \forall\}$, and $x_1, \ldots, x_n$ are exactly the free variables of χ and pairwise distinct.

In the following we define a reduction which is composed of two functions f and g. Given a qBf φ, the function f will define an LTL-formula and g will define a team such that φ is valid if and only if $g(\varphi) \models^{\mathrm{s}} f(\varphi)$. Essentially, the team $g(\varphi)$ will contain three kinds of traces, see Figure 2: (i) traces which are used to mimic universal quantification ($U(i)$ and $E(i)$), (ii) traces that are used to simulate existential quantification ($E(i)$), and (iii) traces used to encode the matrix of φ ($L(j,k)$). Moreover the trace $T(i,1)$ ($T(i,0)$, resp.) is used inside the proof to encode an assignment that maps the variable x_i true (false, resp.). Note that, $U(i), T(i,1), T(i,0), L(j,k)$ are technically singleton sets of traces. For convenience, we identify them with the traces they contain.

Next we inductively define the reduction function f that maps qBf-formulas to LTL-formulas:

$$f(\chi) := \bigvee_{i=1}^{n} \mathsf{F} x_i \vee \bigvee_{i=1}^{m} \mathsf{F} c_i,$$

where χ is the 3CNF-formula $\bigwedge_{j=1}^{m} \bigvee_{k=1}^{3} \ell_{jk}$ with free variables $x_1, \ldots, x_n$,

$$f(\exists x_i \psi) := (\mathsf{F} q_i) \vee f(\psi),$$
$$f(\forall x_i \psi) := \big(\$ \vee (\neg q_i \mathsf{U} q_i) \vee \mathsf{F}[\# \wedge \mathsf{X} f(\psi)]\big) \mathsf{U} \#.$$

The reduction function g that maps qBf-formulas to teams is defined as follows with respect to the traces in Figure 2.

$$g(\chi) := \bigcup_{j=1}^{m} L(j,1) \cup L(j,2) \cup L(j,3),$$

where χ is the 3CNF-formula $\bigwedge_{j=1}^{m} \bigvee_{k=1}^{3} \ell_{jk}$ with free variables $x_1, \ldots, x_n$ and

$$g(\exists x_i \psi) := E(i) \cup g(\psi),$$
$$g(\forall x_i \psi) := U(i) \cup E(i) \cup g(\psi).$$

In Fig. 2, the first position of each trace is marked with a white circle. For instance, the trace of $U(i)$ is then encoded via

$$(\varepsilon, \emptyset\{q_i, \$\}\{\$\}\emptyset\{\$\}\{q_i, \$, \#\}).$$

The reduction function showing QBF-VAL $\leq_{\mathrm{m}}^{\mathrm{p}}$ TPC$^{\mathrm{s}}$ is then $\varphi \mapsto \langle g(\varphi), f(\varphi) \rangle$. Clearly $f(\varphi)$ and $g(\varphi)$ can be computed in linear time with respect to $|\varphi|$.

Intuitively, for the existential quantifier case, the formula $(\mathsf{F} q_i) \vee f(\psi)$ allows to continue in $f(\psi)$ with exactly one of $T(i,1)$ or $T(i,0)$. If $b \in \{0,1\}$ is a truth value then selecting $T(i,b)$ in the team is the same as setting x_i to b. For the case of $f(\forall x_i \psi)$, the formula $(\neg q_i \mathsf{U} q_i) \vee \mathsf{F}[\# \wedge \mathsf{X} f(\psi)]$ with respect to the team $(U(i) \cup E(i))[0, \infty)$ is similar to the existential case choosing x_i to be 1 whereas for $(U(i) \cup E(i))[3, \infty)$ one selects x_i to be 0. The use of the until operator in combination with \$ and # then forces both cases to happen.

Let $\varphi' = Q'x_{n'+1} \cdots Qx_n\chi$, where $Q', Q \in \{\exists, \forall\}$ and let I be an assignment of the variables in $\{x_1, \ldots, x_{n'}\}$ for $n' \leq n$. Then, let

$$g(I, \varphi') := g(\varphi') \cup \bigcup_{x_i \in \mathrm{Dom}(I)} T(i, I(x_i)).$$

We claim $I \models \varphi'$ if and only if $g(I, \varphi') \models^{\mathrm{s}} f(\varphi')$.

Note that when $\varphi' = \varphi$ it follows that $I = \emptyset$ and that $g(I, \varphi') = g(\varphi)$. Accordingly, the lemma follows from the claim of correctness. The claim is proven by induction on the number of quantifier alternations in φ'. The details can be found in the full version [21]. ◀

The matching upper bound follows via a PSPACE algorithm implementing the semantics in straightforward way. The details can be found in the full version [21].

▶ **Theorem 4.2.** $\mathrm{TPC}^{\mathrm{s}}$ *is* PSPACE*-complete w.r.t.* $\leq^{\mathrm{p}}_{\mathrm{m}}$*-reductions.*

The next theorem deals with model checking of the splitjunction-free fragment of LTL under synchronous team semantics.

▶ **Theorem 4.3.** $\mathrm{TMC}^{\mathrm{s}}$ *restricted to splitjunction-free formulas is in* PSPACE.

Proof. Fix $\mathcal{K} = (W, R, \eta, w_I)$ and a splitjunction-free formula φ. We define $S_0 = \{w_I\}$ and $S_{i+1} = \{w' \in W \mid (w, w') \in R \text{ for some } w \in S_i\}$ for all $i \geq 0$. By the pigeonhole principle, this sequence is ultimately periodic with a characteristic (s, p) with $s + p \leq 2^{|W|}$.[3] Next, we define a trace t over $\mathrm{AP} \cup \{\overline{p} \mid p \in \mathrm{AP}\}$ via

$$t(i) = \{p \in \mathrm{AP} \mid p \in \eta(w) \text{ for all } w \in S_i\} \cup \{\overline{p} \mid p \notin \eta(w) \text{ for all } w \in S_i\}$$

that reflects the team semantics of (negated) atomic formulas, which have to hold in every element of the team.

An induction over the construction of φ shows that $T(\mathcal{K}) \models^{\mathrm{s}} \varphi$ if and only if $t \models^{\mathrm{c}} \overline{\varphi}$, where $\overline{\varphi}$ is obtained from φ by replacing each negated atomic proposition $\neg p$ by $\overline{p}$. To conclude the proof, we show that $t \models^{\mathrm{c}} \overline{\varphi}$ can be checked in non-deterministic polynomial space, exploiting the fact that t is ultimately periodic and of the same characteristic as $S_0S_1S_2\cdots$. However, as $s + p$ might be exponential, we cannot just construct a finite representation of t of characteristic (s, p) and then check satisfaction in polynomial space.

Instead, we present an on-the-fly approach which is inspired by similar algorithms in the literature. It is based on two properties:

1. Every S_i can be represented in polynomial space, and from S_i one can compute S_{i+1} in polynomial time.
2. For every LTL formula $\overline{\varphi}$, there is an equivalent non-deterministic Büchi automaton $\mathcal{A}_{\overline{\varphi}}$ of exponential size (see, e.g., [1] for a formal definition of Büchi automata and for the construction of $\mathcal{A}_{\overline{\varphi}}$). States of $\mathcal{A}_{\overline{\varphi}}$ can be represented in polynomial space and given two states, one can check in polynomial time, whether one is a successor of the other.

These properties allow us to construct both t and a run of $\mathcal{A}_{\overline{\varphi}}$ on t on the fly. The details can be found in the full version [21]. ◀

[3] The characteristic of an encoding (t_0, t_1) of an ultimately periodic trace $t_0t_1t_1t_1\cdots$ is the pair $(|t_0|, |t_1|)$. Slightly abusively, we say that $(|t_0|, |t_1|)$ is the characteristic of $t_0t_1t_1t_1\cdots$, although this is not unique.

The complexity of general model checking problem is left open. It is trivially PSPACE-hard, due to Theorem 4.2 and the fact that finite teams of ultimately periodic traces can be represented by Kripke structures. However, the problem is potentially much harder, as one has to deal with infinitely many splits of possibly uncountable teams with non-periodic traces, if a split occurs under the scope of a G-operator. Currently, we are working on interesting language-theoretic problems one encounters when trying to generalise our algorithms for the general path checking problem and for the splitjunction-free model checking problem, e.g., how complex can an LTL-definable split be, if the team to be split is one induced by a Kripke structure.

5 Extensions

In this section we take a brief look into extensions of our logics by dependence atoms and contradictory negation. Contradictory negation combined with team semantics allows for powerful constructions. For instance, the complexity of model checking for propositional logic jumps from NC^1 to PSPACE [27], whereas the complexity of validity and satisfiability jumps all the way to alternating exponential time with polynomially many alternations (ATIME-ALT(exp, pol)) [18].

Formally, we define that $T \models^{\star} \sim\varphi$ if $T \not\models \varphi$. Note that the negation $\sim$ is not equivalent to the negation $\neg$ of atomic propositions defined earlier, i.e., $\sim p$ and $\neg p$ are not equivalent. In the following, problems of the form $\mathrm{TPC}^{\mathrm{a}}(\sim)$, etc., refer to LTL-formulas with negation $\sim$.

Also, we are interested in atoms expressible in first-order (FO) logic over the atomic propositions; the most widely studied ones are dependence, independence, and inclusion atoms [9]. The notion of generalised atoms in the setting of first-order team semantics was introduced by Kuusisto [24]. It turns out that the algorithm for $\mathrm{TPC}^{\mathrm{s}}$ is very robust to such strengthenings of the logic under consideration.

We consider FO-formulas over the signature $(A_p)_{p\in\mathrm{AP}}$, where each A_p is a unary predicate. Furthermore, we interpret a team T as a relational structure $\mathfrak{A}(T)$ over the same signature with universe T such that $t \in T$ is in $A_p^{\mathfrak{A}}$ if and only if $p \in t(0)$. The formulas then express properties of the atomic propositions holding in the initial positions of traces in T. An FO-formula φ *FO-defines* the atomic formula D with $T \models^{\star} D \iff \mathfrak{A}(T) \models \varphi$. In this case, D is also called an *FO-definable generalised atom*.

For instance, the dependence atom $\mathrm{dep}(x; y)$ is FO-definable by $\forall t\forall t'((A_x(t) \leftrightarrow A_x(t')) \to \big(A_y(t) \leftrightarrow A_y(t')\big))$, for $x, y \in \mathrm{AP}$. We call an LTL-formula extended by a generalised atom D an $\mathrm{LTL}(D)$-formula. Similarly, we lift this notion to *sets of generalised atoms* as well as to the corresponding decision problems, i.e., $\mathrm{TPC}^{\mathrm{s}}(D)$ is the path checking problem over synchronous semantics with LTL formulas which may use the generalised atom D.

The result of Theorem 4.2 can be extended to facilitate also the contradictory negation and first-order definable generalised atoms.

▶ **Theorem 5.1.** *Let $\mathcal{D}$ be a finite set of first-order definable generalised atoms. Then $\mathrm{TPC}^{\mathrm{s}}(\mathcal{D})$ and $\mathrm{TPC}^{\mathrm{s}}(\sim)$ are* PSPACE*-complete w.r.t. $\leq^{\mathrm{p}}_{\mathrm{m}}$-reductions.*

The next proposition translates a result from Hannula et al. [18] to our setting. They show completeness for ATIME-ALT(exp, pol) for the satisfiability problem of propositional team logic with negation. This logic coincides with LTL-formulas without temporal operators under team semantics.

▶ **Proposition 5.2** ([18])**.** $\mathrm{TSAT}^{\mathrm{a}}(\sim)$ *and* $\mathrm{TSAT}^{\mathrm{s}}(\sim)$ *for formulas without temporal operators are complete for* ATIME-ALT(exp, pol) *w.r.t. $\leq^{\mathrm{p}}_{\mathrm{m}}$-reductions.*

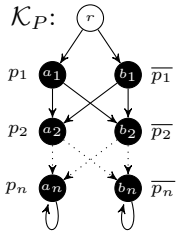


Figure 3 Kripke structure for the proof of Theorem 5.3.

▶ **Theorem 5.3.** $\mathrm{TMC}^{\mathrm{a}}(\sim)$ *and* $\mathrm{TMC}^{\mathrm{s}}(\sim)$ *are hard for* $\mathrm{ATIME\text{-}ALT}(\exp, \mathrm{pol})$ *w.r.t.* $\leq^{\mathrm{p}}_{\mathrm{m}}$*-reductions.*

Proof. We will state a reduction from the satisfiability problem of propositional team logic with negation $\sim$ (short $\mathrm{PL}(\sim)$). The stated hardness then follows from Proposition 5.2.

For $P = \{p_1, \ldots, p_n\}$, consider the traces starting from the root r of the Kripke structure $\mathcal{K}_P$ depicted in Figure 3 using proposition symbols $p_1, \ldots, p_n, \overline{p_1}, \ldots, \overline{p_n}$. Each trace in the model corresponds to a propositional assignment on P. For $\varphi \in \mathrm{PL}(\sim)$, let φ^* denote the $\mathrm{LTL}(\sim)$-formula obtained by simultaneously replacing each (non-negated) variable p_i by $\mathsf{F}p_i$ and each negated variable $\neg p_i$ by $\mathsf{F}\overline{p_i}$. Let P denote the set of variables that occur in φ. Define $\top := (p \vee \neg p)$ and $\bot := p \wedge \neg p$, then $T(\mathcal{K}_P) \models^\star \big(\top \vee ((\sim\bot) \wedge \varphi^*)\big)$ *if and only if* $T' \models^\star \varphi^*$ for some non-empty $T' \subseteq T(\mathcal{K}_P)$. It is easy to check that $T' \models^\star \varphi^*$ *if and only if* the propositional team corresponding to T' satisfies φ and thus the above holds if and only if φ is satisfiable. ◀

In the following, we define the semantics for dependence atoms. For Teams $T \subseteq (2^{\mathrm{AP}})^\omega$ we define $T \models^\star \mathrm{dep}(p_1, \ldots, p_n; q_1, \ldots, q_m)$ if

$$\forall t, t' \in T: \ (t(0) \overset{p_1}{\Leftrightarrow} t'(0), \ldots, t(0) \overset{p_n}{\Leftrightarrow} t'(0)) \text{ implies } (t(0) \overset{q_1}{\Leftrightarrow} t'(0), \ldots, t(0) \overset{q_m}{\Leftrightarrow} t'(0)),$$

where $t(i) \overset{p}{\Leftrightarrow} t(j)$ means the sets $t(i)$ and $t(j)$ agree on proposition p, i.e., both contain p or not. Observe that the formula $\mathrm{dep}(;p)$ merely means that p has to be constant on the team. Often, due to convenience we will write $\mathrm{dep}(p)$ instead of $\mathrm{dep}(;p)$. Note that the hyperproperties 'input determinism' now can be very easily expressed via the formula $\mathrm{dep}(i_1, \ldots, i_n; o_1, \ldots, o_m)$, where i_j are the (public) input variables and o_j are the (public) output variables.

Problems of the form $\mathrm{TSAT}^{\mathrm{a}}(\mathrm{dep})$, etc., refer to LTL-formulas with dependence operator dep. The following proposition follows from the corresponding result for classical LTL using downwards closure and the fact that on singleton teams dependence atoms are trivially fulfilled.

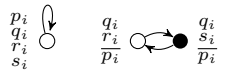
▶ **Proposition 5.4.** $\mathrm{TSAT}^{\mathrm{a}}(\mathrm{dep})$ *and* $\mathrm{TSAT}^{\mathrm{s}}(\mathrm{dep})$ *are* PSPACE*-complete.*

In the following, we will show a lower bound while the matching upper bound still is open.

▶ **Theorem 5.5.** $\mathrm{TPC}^{\mathrm{a}}(\mathrm{dep})$ *is* PSPACE*-hard w.r.t.* $\leq^{\mathrm{p}}_{\mathrm{m}}$*-reductions.*

Proof. As in the proof of Lemma 4.1, we reduce from QBF-VAL.

Consider a given quantified Boolean formula $\exists x_1 \forall x_2 \cdots Q x_n \chi$, where $\chi = \bigwedge_{j=1}^{m} \bigvee_{k=1}^{3} \ell_{jk}$, $Q \in \{\exists, \forall\}$, and $x_1, \ldots, x_n$ are exactly the free variables of χ and pairwise distinct. We will use two traces for each variable x_i (gadget for x_i) as shown in Figure 4.



everywhere $p_j, \overline{p_j}$ for $i \neq j$

Figure 4 Traces in the proof of Theorem 5.5.

Intuitively, the proposition p_i marks that the variable x_i is set true while the proposition $\overline{p}_i$ marks that x_i is set false, q_i encodes that the gadget is used to quantify x_i, and s_i, r_i are auxiliary propositions. Picking the left trace corresponds to setting x_i to true and picking the right trace corresponds to setting x_i to false. In the following, we omit the p_j and $\overline{p_j}$, when $j \neq i$, for readability. Then, the team T is defined as

$$T := \{(\varepsilon, \{p_i, q_i, r_i, s_i\}), (\varepsilon, \{q_i, r_i, \overline{p}_i\}\{q_i, s_i, \overline{p}_i\}) \mid 1 \leq i \leq n\}.$$

Next, we recursively define the LTL(dep)-formula used in the reduction: $f(\chi)$ is obtained from χ by substituting every positive literal x_i by p_i and negated literal $\neg x_i$ by $\overline{p_i}$, $f(\exists x_i \psi) := \big(q_i \wedge \mathrm{dep}(p_i)\big) \vee f(\psi)\big)$, and

$$f(\forall x_i \psi) := \mathsf{G}\Big(\big(\mathrm{dep}(p_i) \wedge q_i \wedge r_i\big) \vee \big(s_i \wedge f(\psi)\big)\Big).$$

In the existential quantification of x_i, the splitjunction requires for the x_i-trace-pair to put $(\varepsilon, \{p_i, q_i, r_i, s_i\})$ into the left or right subteam (of the split). The trace $(\varepsilon, \{q_i, r_i, \overline{p}_i\}\{q_i, s_i, \overline{p}_i\})$ has to go to the opposite subteam as $\mathrm{dep}(p_i)$ requires p_i to be of constant value. (Technically both of the traces could be put to the right subteam, but this logic is downwards closed and, accordingly, this allows to omit this case.) As explained before, we existentially quantify x_i by this split. For universal quantification, the idea is a bit more involved. To verify $T \models^{\star} \mathsf{G}\theta$, where $\mathsf{G}\theta = f(\forall x_i \psi)$ essentially two different teams T' for which $T' \models \theta$ need to be verified.
(1.) $(\varepsilon, \{p_i, q_i, r_i, s_i\}), (\varepsilon, \{q_i, r_i, \overline{p}_i\}\{q_i, s_i, \overline{p}_i\}) \in T'$. In this case, $(\varepsilon, \{p_i, q_i, r_i, s_i\})$ must be put to the right subteam of the split and $(\varepsilon, \{q_i, r_i, \overline{p}_i\}\{q_i, s_i, \overline{p}_i\})$ to the left subteam, setting x_i true.
(2.) $(\varepsilon, \{p_i, q_i, r_i, s_i\}), (\varepsilon, \{q_i, s_i, \overline{p}_i\}\{q_i, r_i, \overline{p}_i\}) \in T'$. In this case, $(\varepsilon, \{p_i, q_i, r_i, s_i\})$ must be put to the left and $(\varepsilon, \{q_i, s_i, \overline{p}_i\}\{q_i, r_i, \overline{p}_i\})$ to the right subteam, implicitly forcing x_i to be false. These observations are utilised to prove that $\langle \exists x_1 \forall x_2 \cdots Q x_n \chi \rangle \in$ QBF-VAL if and only if $\langle f(\exists x_1 \forall x_2 \cdots Q x_n \chi), T \rangle \in \mathrm{TPC}^{\mathrm{a}}(\mathrm{dep})$. The reduction is polynomial time computable in the input size. ◀

The following result from Virtema talks about the validity problem of propositional team logic.

▶ **Proposition 5.6** ([34]). *Validity of propositional logic with dependence atoms is* NEXPTIME*-complete w.r.t.* $\leq^{\mathrm{p}}_{\mathrm{m}}$*-reductions.*

▶ **Theorem 5.7.** $\mathrm{TMC}^{\mathrm{a}}(\mathrm{dep})$ *and* $\mathrm{TMC}^{\mathrm{s}}(\mathrm{dep})$ *are* NEXPTIME*-hard w.r.t.* $\leq^{\mathrm{p}}_{\mathrm{m}}$*-reductions.*

Proof. The proof of this result uses the same construction idea as in the proof of Theorem 5.3, but this time from a different problem, namely, validity of propositional logic with dependence atoms which settles the lower bound by Proposition 5.6. Due to downwards closure the validity of propositional formulas with dependence atoms boils down to model checking the maximal team in the propositional (and not in the trace) setting, which essentially is achieved by $T(\mathcal{K})$, where $\mathcal{K}$ is the Kripke structure from the proof of Theorem 5.3. ◀

6 LTL under Team Semantics vs. HyperLTL

LTL under team semantics expresses hyperproperties [6], that is, sets of teams, or equivalently, sets of sets of traces. Recently, HyperLTL [5] was proposed to express information flow properties, which are naturally hyperproperties. For example, input determinism can be expressed as follows: every pair of traces that coincides on their input variables, also coincides on their output variables (this can be expressed in LTL with team semantics by a dependence atom dep as sketched is Section 5). To formalise such properties, HyperLTL allows to quantify over traces. This results in a powerful formalism with vastly different properties than LTL [14]. After introducing syntax and semantics of HyperLTL, we compare the expressive power of LTL under team semantics and HyperLTL.

The formulas of HyperLTL are given by the grammar

$$\varphi ::= \exists \pi.\varphi \mid \forall \pi.\varphi \mid \psi, \qquad \psi ::= p_\pi \mid \neg\psi \mid \psi \vee \psi \mid \mathsf{X}\psi \mid \psi \mathsf{U} \psi,$$

where p ranges over atomic propositions in AP and where π ranges over a given countable set $\mathcal{V}$ of *trace variables*. The other Boolean connectives and the temporal operators release R, eventually F, and always G are derived as usual, due to closure under negation. A sentence is a closed formula, i.e., one without free trace variables.

The semantics of HyperLTL is defined with respect to trace assignments that are a partial mappings $\Pi\colon \mathcal{V} \to (2^{\mathrm{AP}})^\omega$. The assignment with empty domain is denoted by $\Pi_\emptyset$. Given a trace assignment Π, a trace variable π, and a trace t, denote by $\Pi[\pi \to t]$ the assignment that coincides with Π everywhere but at π, which is mapped to t. Further, $\Pi[i, \infty)$ denotes the assignment mapping every π in Π's domain to $\Pi(\pi)[i, \infty)$. For teams T and trace-assignments Π we define

$$\begin{array}{ll}
(T,\Pi) \models^{\mathrm{h}} p_\pi & \text{if } p \in \Pi(\pi)(0), \\
(T,\Pi) \models^{\mathrm{h}} \neg\psi & \text{if } (T,\Pi) \not\models^{\mathrm{h}} \psi, \\
(T,\Pi) \models^{\mathrm{h}} \psi_1 \vee \psi_2 & \text{if } (T,\Pi) \models^{\mathrm{h}} \psi_1 \text{ or } (T,\Pi) \models^{\mathrm{h}} \psi_2, \\
(T,\Pi) \models^{\mathrm{h}} \mathsf{X}\psi & \text{if } (T,\Pi[1,\infty)) \models^{\mathrm{h}} \psi, \\
(T,\Pi) \models^{\mathrm{h}} \psi_1 \mathsf{U} \psi_2 & \text{if } \exists k \geq 0 : (T,\Pi[k,\infty)) \models^{\mathrm{h}} \psi_2 \text{ and } \forall 0 \leq k' < k : (T,\Pi[k',\infty)) \models^{\mathrm{h}} \psi_1, \\
(T,\Pi) \models^{\mathrm{h}} \exists \pi.\psi & \text{if } \exists t \in T : (T,\Pi[\pi \to t]) \models^{\mathrm{h}} \psi, \text{ and} \\
(T,\Pi) \models^{\mathrm{h}} \forall \pi.\psi & \text{if } \forall t \in T : (T,\Pi[\pi \to t]) \models^{\mathrm{h}} \psi.
\end{array}$$

We say that T satisfies a sentence φ, if $(T, \Pi_\emptyset) \models^{\mathrm{h}} \varphi$, and write $T \models^{\mathrm{h}} \varphi$. The semantics of HyperLTL are synchronous, i.e., the semantics of the until refers to a single k. Accordingly, one could expect that HyperLTL is closer related to LTL under synchronous team semantics than to LTL under asynchronous team semantics. In the following, we refute this intuition.

Formally, a HyperLTL sentence φ and an LTL formula φ' under synchronous (asynchronous) team semantics are equivalent, if for all teams T: $T \models^{\mathrm{h}} \varphi$ if and only if $T \models^{\mathrm{s}} \varphi'$ ($T \models^{\mathrm{a}} \varphi'$). In the following, let $\forall$-HyperLTL denote that set of HyperLTL sentences of the form $\forall \pi.\, \psi$ with quantifier-free ψ, i.e., sentences with a single universal quantifier.

▶ **Theorem 6.1.** 1. *No* LTL*-formula under synchronous or asynchronous team semantics is equivalent to* $\exists \pi.p_\pi$.
2. *No* HyperLTL *sentence is equivalent to* $\mathsf{F}p$ *under synchronous team semantics.*
3. LTL *under asynchronous team semantics is as expressive as* $\forall$-HyperLTL.

Proof.

1. Consider $T = \{\emptyset^\omega, \{p\}\emptyset^\omega\}$. We have $T \models^{\mathrm{h}} \exists \pi.p_\pi$. Assume there is an equivalent LTL formula under team semantics, call it φ. Then, $T \models^{\star} \varphi$ and thus $\{\emptyset^\omega\} \models^{\star} \varphi$ by downwards closure. Hence, by equivalence, $\{\emptyset^\omega\} \models^{\mathrm{h}} \exists \pi.p_\pi$, yielding a contradiction.

2. Bozzelli et al. proved that the property encoded by $\mathsf{F}p$ under synchronous team semantics cannot be expressed in HyperLTL [3].
3. Let φ be an LTL-formula and define $\varphi_h := \forall\pi.\varphi'$, where φ' is obtained from φ by replacing each atomic proposition p by p_π. Then, due to singleton equivalence, $T \models^{\mathrm{a}} \varphi$ if and only if $T \models^{\mathrm{h}} \varphi_h$. For the other implication, let $\varphi = \forall\pi.\psi$ be a HyperLTL sentence with quantifier-free ψ and let ψ' be obtained from ψ by replacing each atomic proposition p_π by p. Then, again due to the singleton equivalence, we have $T \models^{\mathrm{h}} \varphi$ if and only if $T \models^{\mathrm{a}} \psi'$.

◀

Note that these separations are obtained by very simple formulas, and are valid for LTL(dep) formulas, too. In particular, the HyperLTL formulas are all negation-free.

▶ **Corollary 6.2.** HyperLTL *and* LTL *under synchronous team semantics are of incomparable expressiveness and* HyperLTL *is strictly more expressive than* LTL *under asynchronous team semantics.*

7 Conclusion

We introduced synchronous and asynchronous team semantics for linear temporal logic LTL, studied complexity and expressive power of related logics, and compared them to HyperLTL. We concluded that LTL under team semantics is a valuable logic which allows to express relevant hyperproperties and complements the expressiveness of HyperLTL while allowing for computationally simpler decision problems. We conclude with some directions of future work and open problems.

1. We showed that some important properties that cannot be expressed in HyperLTL (such as uniform termination) can be expressed by LTL-formulas in synchronous team semantics. Moreover input determinism can be expressed in LTL(dep). What other important and practical hyperproperties can be expressed in LTL under team semantics? What about in its extensions with dependence, inclusion, and independence atoms, or the contradictory negation.
2. We showed that with respect to expressive power HyperLTL and LTL under synchronous team semantics are incomparable. What about the extensions of LTL under team semantics? For example the HyperLTL formula $\exists\pi.p_\pi$ is expressible in LTL($\sim$). Can we characterise the expressive power of relevant extensions of team LTL as has been done in first-order and modal contexts?
3. We studied the complexity of path-checking, model checking, and satisfiability problems of team LTL and its extensions with dependence atoms and the contradictory negation. Many problems are still open: Can we show matching upper bounds for the hardness results of Section 5? What is the complexity of $\mathrm{TMC}^{\mathrm{s}}$ when splitjunctions are allowed? What happens when LTL is extended with inclusion or independence atoms?
4. Can we give a natural team semantics to CTL^* and compare it to $\mathrm{HyperCTL}^*$ [5]?

References

1 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking.* The MIT Press, 2008.

2 Borzoo Bonakdarpour and Bernd Finkbeiner. Runtime verification for HyperLTL. In Yliès Falcone and César Sánchez, editors, *RV 2016*, volume 10012 of *LNCS*, pages 41–45. Springer, 2016.

3 Laura Bozzelli, Bastien Maubert, and Sophie Pinchinat. Unifying hyper and epistemic temporal logics. In Andrew M. Pitts, editor, *FoSSaCS 2015*, volume 9034 of *LNCS*, pages 167–182. Springer, 2015.

4 Julian Bradfield. On the structure of events in boolean games. In *Logics for Dependence and Independence*. Dagstuhl Reports, 2015.

5 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer, editors, *POST 2014*, volume 8414 of *LNCS*, pages 265–284. Springer, 2014.

6 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

7 Pedro R. D'Argenio, Gilles Barthe, Sebastian Biewer, Bernd Finkbeiner, and Holger Hermanns. Is your software on dope? - formal analysis of surreptitiously "enhanced" programs. In Hongseok Yang, editor, *ESOP 2017*, volume 10201 of *LNCS*, pages 83–110. Springer, 2017.

8 Arnaud Durand, Miika Hannula, Juha Kontinen, Arne Meier, and Jonni Virtema. Approximation and dependence via multiteam semantics. In *FoIKS 2016*, pages 271–291, 2016.

9 Arnaud Durand, Juha Kontinen, and Heribert Vollmer. Expressivity and complexity of dependence logic. In *Dependence Logic: Theory and Applications*, pages 5–32. Birkhäuser, 2016.

10 Bernd Finkbeiner and Christopher Hahn. Deciding Hyperproperties. In Josée Desharnais and Radha Jagadeesan, editors, *CONCUR 2016*, volume 59 of *LIPIcs*, pages 13:1–13:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

11 Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Monitoring hyperproperties. In Shuvendu K. Lahiri and Giles Reger, editors, *RV 2017*, volume 10548 of *LNCS*, pages 190–207. Springer, 2017.

12 Bernd Finkbeiner, Christian Müller, Helmut Seidl, and Eugen Zalinescu. Verifying security policies in multi-agent workflows with loops. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *CCS 2017*, pages 633–645. ACM, 2017. doi:10.1145/3133956.

13 Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking HyperLTL and HyperCTL*. In Daniel Kroening and Corina S. Pasareanu, editors, *CAV 2015 (Part I)*, volume 9206 of *LNCS*, pages 30–48. Springer, 2015.

14 Bernd Finkbeiner and Martin Zimmermann. The first-order logic of hyperproperties. In Heribert Vollmer and Brigitte Vallée, editors, *STACS 2017*, volume 66 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

15 Pietro Galliani. Inclusion and exclusion dependencies in team semantics - on some logics of imperfect information. *Ann. Pure Appl. Logic*, 163(1):68–84, 2012.

16 Erich Grädel, Juha Kontinen, Jouko Väänänen, and Heribert Vollmer. Logics for dependence and independence (Dagstuhl seminar 15261). *Dagstuhl Reports*, 5(6):70–85, 2015.

17 Erich Grädel and Jouko Väänänen. Dependence and independence. *Studia Logica*, 101(2):399–410, 2013.

18 Miika Hannula, Juha Kontinen, Jonni Virtema, and Heribert Vollmer. Complexity of propositional logics in team semantic. *ACM Trans. Comput. Logic*, 19(1):2:1–2:14, 2018.

19 Wilfrid Hodges. Compositional semantics for a language of imperfect information. *Logic Journal of the IGPL*, 5(4):539–563, 1997.

20 Andreas Krebs, Arne Meier, and Jonni Virtema. A team based variant of CTL. In Fabio Grandi, Martin Lange, and Alessio Lomuscio, editors, *TIME 2015*, pages 140–149. IEEE Computer Society, 2015.

21 Andreas Krebs, Arne Meier, Jonni Virtema, and Martin Zimmermann. Team semantics for the specification and verification of hyperproperties. *CoRR*, abs/1709.08510, 2017. arXiv:1709.08510.

22 Lars Kuhtz. *Model checking finite paths and trees*. PhD thesis, Saarland University, 2010.

23 Lars Kuhtz and Bernd Finkbeiner. LTL path checking is efficiently parallelizable. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *ICALP 2009 (Part II)*, volume 5556 of *LNCS*, pages 235–246. Springer, 2009.

24 Antti Kuusisto. A Double Team Semantics for Generalized Quantifiers. *Journal of Logic, Language and Information*, 24(2):149–191, 2015.

25 Richard Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comput.*, 6(3):467–480, 1977.

26 Nicolas Markey and Philippe Schnoebelen. Model checking a path. In Roberto M. Amadio and Denis Lugiez, editors, *CONCUR 2003*, volume 2761 of *LNCS*, pages 248–262. Springer, 2003.

27 Julian-Steffen Müller. *Satisfiability and Model Checking in Team Based Logics*. PhD thesis, Leibniz University of Hannover, 2014.

28 Amir Pnueli. The temporal logic of programs. In *FOCS 1977*, pages 46–57. IEEE Computer Society, 1977.

29 Markus N. Rabe. *A Temporal Logic Approach to Information-flow Control*. PhD thesis, Saarland University, 2016.

30 Ilya Shpitser. Causal inference and logics of dependence and independence. In *Logics for Dependence and Independence*. Dagstuhl Reports, 2015.

31 A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.

32 Jouko Väänänen. *Dependence Logic*. Cambridge University Press, 2007.

33 Jouko Väänänen. Modal Dependence Logic. In *New Perspectives on Games and Interaction*. Amsterdam University Press, Amsterdam, 2008.

34 Jonni Virtema. Complexity of validity for propositional dependence logics. *Inf. Comput.*, 253:224–236, 2017. doi:10.1016/j.ic.2016.07.008.

35 Heribert Vollmer. *Introduction to Circuit Complexity – A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999.

Consistency for Counting Quantifiers

Florent R. Madelaine
LIMOS, Université d'Auvergne, Clermont-Ferrand, France

Barnaby Martin
Department of Computer Science, Durham University, U.K.

Abstract

We apply the algebraic approach for Constraint Satisfaction Problems (CSPs) with counting quantifiers, developed by Bulatov and Hedayaty, for the first time to obtain classifications for computational complexity. We develop the consistency approach for expanding polymorphisms to deduce that, if H has an expanding majority polymorphism, then the corresponding CSP with counting quantifiers is tractable. We elaborate some applications of our result, in particular deriving a complexity classification for partially reflexive graphs endowed with all unary relations. For each such structure, either the corresponding CSP with counting quantifiers is in P, or it is NP-hard.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases Quantified Constraints, Constraint Satisfaction, Logic in Computer Science, Universal Algebra, Computational Complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.11

1 Introduction

The *constraint satisfaction problem*, CSP($\mathcal{B}$), originating in artificial intelligence, is known to admit several equivalent formulations. Two of the best known consider the parameter $\mathcal{B}$ to be a relational structure and may be phrased as the problem of query evaluation of primitive positive (pp) sentences – those involving only $\{\exists, \wedge, =\}$ – on $\mathcal{B}$, and the homomorphism problem to $\mathcal{B}$ (see, e.g., [19]). For finite $\mathcal{B}$, CSP($\mathcal{B}$) is NP-complete in general, and a great deal of effort was expended in classifying its complexity in various different classes. It was conjectured by Feder and Vardi [13] that all such CSP($\mathcal{B}$) are either in P or NP-complete and this was finally proved last year independently by Bulatov [6] and Zhuk [23].

A popular generalisation of the CSP involves considering the query evaluation problem for the logic involving only $\{\forall, \exists, \wedge, =\}$. (This logic admits various names but we will leave it nameless in this work as was the case in the foundational [2].) The resulting *Quantified Constraint Satisfaction Problem*, QCSP($\mathcal{B}$), allows for a broader class, used in artificial intelligence to capture non-monotonic reasoning, whose complexities rise to Pspace-complete.

In this paper, we study counting quantifiers of the form $\exists^{\geq j}$, which allow one to assert the existence of at least j elements such that the ensuing property holds. Thus, on a structure $\mathcal{B}$ with domain of size n, the quantifiers $\exists^{\geq 1}$ and $\exists^{\geq n}$ are precisely $\exists$ and $\forall$, respectively. Counting quantifiers have been fiercely studied in finite model theory (see [12, 22]), where the focus is on supplementing the descriptive power of various logics. Of wider interest is the majority quantifier $\exists^{\geq n/2}$ (on a structure of domain size n), which sits broadly midway between $\exists$ and $\forall$. Majority quantifiers turn up across diverse fields of logic and have various practical applications, e.g. in cognitive appraisal and voting theory [11].

We postulate variants of CSP($\mathcal{B}$) in which the input sentence to be evaluated on $\mathcal{B}$ (of size $|B|$) remains positive conjunctive in its quantifier-free part, but is quantified by various counting quantifiers from some non-empty set.

For $X \subseteq \{1, \ldots, |B|\}$, $X \neq \emptyset$, the X-CSP($\mathcal{B}$), introduced in [21], takes as input a sentence given by a conjunction of atoms quantified by quantifiers of the form $\exists^{\geq j}$ for $j \in X$ (this logic is termed X-pp). It then asks whether this sentence is true on $\mathcal{B}$. In the present paper, we will mostly consider the situation in which all counting quantifiers are present, and we will denote this problem CQCSP($\mathcal{B}$), instead of $\{1, \ldots, |B|\}$-CSP($\mathcal{B}$). The corresponding logic, involving only $\{\exists^{\geq 1}, \ldots, \exists^{\geq |B|}, \wedge, =\}$, we will call cq-pp.

The algebraic method has been very potent in understanding the complexity of CSPs and QCSPs [5, 6, 23, 10]. Recently, an algebraic theory tailored to counting quantifiers has been given [8] (early version was [7]).

A *polymorphism* of a structure $\mathcal{B}$ is a homomorphism from $\mathcal{B}^k$ to $\mathcal{B}$, for some k. Let $\{1\} \subseteq X \subseteq \{1, \ldots, |B|\}$. Call a function $f : B^k \to B$ *expanding on* X if, for all $X_1, \ldots, X_k \subseteq B$ such that $|X_1| = \ldots = |X_k| = j \in X$, we have $|f(X_1, \ldots, X_k)| \geq j$. This condition at $j = 1$ is trivial (it says that f is a function) and at $j = |B|$ asserts surjectivity. If $X = \{1, \ldots, |B|\}$ we simply term f *expanding*.

▶ **Lemma 1** (Theorem 8 [7]; Corollary 14 [8]). *The relations that are cq-pp-definable over $\mathcal{B}$ are exactly those that are preserved by the expanding polymorphisms of $\mathcal{B}$.*

In this paper, we will only make use of the "easy" direction of Lemma 1, that is, any relation that is cq-pp-definable over $\mathcal{B}$ is preserved by the expanding polymorphisms of $\mathcal{B}$.

The *list homomorphism* problem, which we will call List-CSP($\mathcal{B}$), is defined as CSP($\mathcal{B}$), save that one gives lists for each input variable stating which elements of the domain B that variable may be evaluated on. This is equivalent to CSP($\mathcal{B}^*$), where $\mathcal{B}^*$ is $\mathcal{B}$ endowed with additional unary relations for each subset of B. Indeed, this class of CSPs was among the first to be proved in line with the Feder-Vardi dichotomy conjecture [4]. The key class of polymorphisms here is known as *conservative* and the property they have is that $f(x_1, \ldots, x_k) \in \{x_1, \ldots, x_k\}$, for all $x_1, \ldots, x_k$ in the domain. Let us give explicitly the classification for this problem in the special case of graphs. We call a k-ary operation *near-unanimity*, for $k \geq 3$, if it returns the repeated argument when all but at most one of its arguments is the same. Ternary near-unanimity operations are called *majority*. We refer to a graph as *partially reflexive* to indicate that each vertex may or may not have a self-loop.

▶ **Theorem 2** (From Theorem 5.3 [3] and Theorem 2.1 [15]). *Let $\mathcal{H}^*$ be a partially reflexive graph expanded with all possible unary relations. Then either $\mathcal{H}^*$ admits a conservative majority polymorphism and* CSP*($\mathcal{H}^*$) is in* P*; or* CSP*($\mathcal{H}^*$) is* NP*-complete.*

Contribution

It is easy to see, but does not appear to have been noted, that conservative polymorphisms are expanding polymorphisms *in excelsis*. That is, they are the most natural examples of such polymorphisms that one is likely to imagine.

▶ **Lemma 3.** *Let f be a k-ary operation that is conservative. Then f is also expanding.*

Proof. Consider k subsets of the domain A of f, $A_1, \ldots, A_k$, each of size $m \leq |A|$. We need to argue that $|f(A_1, \ldots, A_k)| \geq m$. We proceed by induction on m where the base case $m = 1$ is trivial. Suppose it holds for m but does not hold for $m + 1$. Take $A'_1, \ldots, A'_k$, each of size $m+1 \leq |A|$. There must be $a'_1 \in A'_1, \ldots, a'_k \in A'_k$ so that none of $a'_1, \ldots, a'_k \in f(A'_1, \ldots, A'_k)$, since $|f(A'_1, \ldots, A'_k)| < m + 1$. By inductive hypothesis, $|f(A'_1 \setminus \{a'_1\}, \ldots, A'_k \setminus \{a'_k\})| \geq m$. But $f(a'_1, \ldots, a'_k) \in \{a'_1, \ldots, a'_k\}$ by conservativity, which is a contradiction. ◀

We prove that if a finite structure $\mathcal{B}$ admits an expanding majority polymorphism, then CQCSP($\mathcal{B}$) is in P. In doing so, we answer Question 1 of [21], for the case in the paragraph immediately after it. The algorithm is rather more sophisticated than in the case of CSP or QCSP. We note that a majority that is not expanding can appear as a polymorphism of $\mathcal{B}$ despite that CQCSP($\mathcal{B}$) is NP-hard. We derive as a corollary a complexity classification for CQCSP($\mathcal{H}^*$), where $\mathcal{H}^*$ is a partially reflexive graph endowed with all unary relations. This classification is in line with that of Theorem 2. We further derive a classification for successive approximations to CQCSP($\mathcal{B}$), where $\mathcal{B}$ is a binary first-order expansion of $(\mathbb{Z}; succ)$, whose relations (as digraphs) have bounded-degree. We then make some further observations on the usefulness of expanding majority polymorphisms and relate our work to some recent developments in surjective CSP involving the concept of endo-triviality.

Structure of the paper

This paper is organised as follows. After the preliminaries, Section 3 elaborates the consistency algorithm, and Section 4 gives some applications of this algorithm to complexity classifications. In Section 5, we close with some final remarks about the relationship between List-CSP and CQCSP. Owing to reasons of space, some proofs are deferred to the appendix.

2 Preliminaries

The reader will probably already have picked up that, if $\mathcal{B}$ is a relational structure, then B is its domain and $|B|$ the size of its domain. A *homomorphism*, from a structure $\mathcal{A}$ to a structure $\mathcal{B}$ over the same signature σ, is a function $h : A \to B$ such that, for each relation $R \in \sigma$, if $(x_1, \ldots, x_r) \in R^{\mathcal{A}}$, then $(h(x_1), \ldots, h(x_r)) \in R^{\mathcal{B}}$. A *k-ary polymorphism* of $\mathcal{B}$ is a k-ary operation f on B so that, $(x_1^1, \ldots, x_r^1), \ldots, (x_1^k, \ldots, x_r^k) \in R^{\mathcal{B}}$, then also $(f(x_1^1, \ldots, x_1^k), \ldots, f(x_r^1, \ldots, x_r^k)) \in R^{\mathcal{B}}$.

Given a set B, and an integer $i \geq 0$, we denote its ith power by B^i (B^0 being $\emptyset$). For an integer $c \geq 1$ We write $\binom{B}{c}$ for the following set of subsets of B : $\{S \subseteq B$ such that S has c elements$\}$. A *Skolem (partial) function* g_x for a variable x quantified as $\exists^{\geq c} x$ in the sentence is a partial function to $\binom{B}{c}$, whose arity is the number of variables coming before x in the quantifier prefix of the formula.

The Skolem functions g_i from B^{i-1} to $\binom{B}{c_i}$ $(1 \leq i \leq m)$ *witness that* φ *holds in* $\mathcal{B}$ iff $\forall b_1 \in g_1 \forall b_2 \in g_2(b_1) \ldots \forall b_n \in g_n(b_1, b_2, \ldots, b_{n-1})$ $\mathcal{B} \models \varphi(b_1, b_2, \ldots, b_m)$. If there are such Skolem functions then $\mathcal{B}$ models φ.

For a r-ary relation R in σ and sets $B_1, B_2, \ldots, B_r$, we write that $R(B_1, B_2, \ldots, B_r)$ holds in $\mathcal{B}$ iff for every $1 \leq i \leq r$ and every b_i in B_i, it is the case that $R(b_1, b_2, \ldots, b_r)$ holds in $\mathcal{B}$.

Let us note that counting quantifiers of the same cardinality do not in general commute. In particular, for every choice of $1 < i < n$, there exists a structure $\mathcal{B}$ over the signature of digraph (a single binary predicate E) of size $|B| = n$, such that $\exists^{\geq i} x \exists^{\geq i} y\, E(x, y)$ holds in $\mathcal{B}$ but $\exists^{\geq i} y \exists^{\geq i} x\, E(x, y)$ does not. For more on this, see [21].

3 An algorithm for consistency

In this section we will prove the following main theorem.

▶ **Theorem 4.** *Suppose* $\mathcal{B}$ *has an expanding majority polymorphism. Then CQCSP*($\mathcal{B}$) *is in* P.

Just as in the case of CSP and QCSP, by monotonicity, a sentence does not hold if any subsentence does not. Here, by subsentence we mean the sentence induced by selecting some variables. This means that for any structure, a not necessarily complete but polynomial algorithm consists in selecting some subsentences of bounded size and checking whether they hold : if one subsentence fails to hold, then we may answer no. A slightly cleverer way of doing this consists in propagating a potential solution from subsentences with overlapping variables. This is a basic approach known as enforcing local consistency, which is known to imply global consistency for CSP whenever the constraint language is closed under a majority operation [16, 18]. Our algorithm is a careful adaptation to our context.

The consistency argument will be somewhat more fiddly than for CSP. This is due to the fact that quantifiers do not commute and also that we have counting quantifiers and need to keep track of Skolem functions that witness (un)satisfiability of a sentence with counting quantifiers.

The consistency algorithm for establishing our Theorem 4 that we propose does this for the constraints induced by subsentences obtained by selecting up to 3 variables of the prefix and the atoms involving them in the quantifier-free part (we assume w.l.o.g. that the sentence is in prenex form) and maintaining consistency between the witnesses. These witnesses are sets of suitable size, namely the range of the Skolem functions corresponding to the counting quantifiers.

In the following and unless specified otherwise, subset means subset of the domain B of the structure $\mathcal{B}$. We assume some arbitrary order over B and subsets are ordered accordingly.

3.1 Sentences with three variables

Let us examine first a 3 variable sentence φ of the following form:

$$\exists^{\geq c_1} x_1 \exists^{\geq c_2} x_2 \exists^{\geq c_3} x_3 R_{1,2}(x_1, x_2) \wedge R_{2,3}(x_2, x_3) \wedge R_{1,3}(x_1, x_3).$$

For a subset S of size c_1, and subsets T_i of size c_2, we write $OK_{1,2}(S, T_1, \ldots, T_{c_1})$ whenever $R_{1,2}(s_i, T_i)$ holds for all s_i in S (recall that sets are ordered). We proceed similarly to define the $c_1 + 1$-ary predicate $OK_{1,3}$ between a subset of size c_1 and c_1 subsets of size c_3 and the $c_2 + 1$-ary predicate $OK_{2,3}$ between a subset of size c_2 and c_2 subsets of size c_3. The sentence φ holds whenever there is a subset S of size c_1, subsets T_i of size c_2 with $1 \leq i \leq c_1$, subsets $U_{i,j}$ of size c_3 with $1 \leq j \leq c_2$ such that :

$$OK_{1,2}(S, T_1, \ldots, T_{c_1}) \wedge \bigwedge_{1 \leq i \leq c_1} OK_{2,3}(T_i, U_{i,1}, \ldots, U_{i,c_2}) \wedge \bigwedge_{1 \leq j_1 \leq c_2} \cdots \bigwedge_{1 \leq j_{c_1} \leq c_2} OK_{1,3}(S, U_{1,j_1}, \ldots, U_{c_1,j_{c_1}}).$$

3.2 Data structure

With this small example in mind, the following data structure used by our algorithm should become clearer.

- Each variable $\exists^{\geq c_i} x_i$ is represented by a domain that consists of subsets S of size c_i.
- We maintain a $c_i + 1$-ary predicate $OK_{i,j}$ as in the above example between the domains of any pair of variables x_i, x_j as long as x_j comes after x_i in the prefix of quantification and that x_i and x_j occur both in some atom.

3.3 Binary Predicates Only

Of course, unlike in our small example, the input sentence φ' may well have non binary atoms and the parameter structure $\mathcal{B}'$ corresponding relations of arity 3 or more. We project almost in the usual fashion all atoms/relations involving two variables x_1 and x_2 into a single binary constraint R_{x_1,x_2} (if there are constraints, otherwise there is no binary constraint).

Unlike in the CSP case, *we check that counting requirements induced by the sentence are met.* Formally, for every pair of distinct variables x_1, x_2 quantified as $\exists^{\geq c_1} x_1 \exists^{\geq c_2} x_2$, we consider the binary constraint R_{x_1,x_2} to be the intersection of the binary relations R'_{x_1,x_2} induced by atoms $R'(\bar{y})$ such that both x_1 and x_2 occur in $\bar{y}$ as follows. $R'_{x_1,x_2}(b_1, b_2)$ holds whenever for any variable y distinct from both x_1 and x_2 with quantifier prefix $\exists^{\geq c_y} y$ occurring at position i in $\bar{y}$ (to distinguish the potentially many occurrences of y, we will write y_i for the occurrence of y at position i) there exists a set B_i of size at least c_y such that $R'(\pi(\bar{y}))$ holds where $\pi(x_1) = b_1, \pi(x_2) = b_2$ and $\pi(y_i) = B_i$.

We denote by φ this sentence with binary atoms and by $\psi(\bar{x})$ its subsentence induced naturally by the variables $\bar{x}$. We write $\mathcal{B}$ for the structure with binary relations. Note that these relations are cq-pp interpretations of the relations of $\mathcal{B}'$.

▶ **Proposition 1.** *If $\mathcal{B}'$ has an expanding majority f, then*

(i) *$\mathcal{B}$ has also f as an expanding majority*

(ii) *$\mathcal{B}$ models φ (binary setting) iff $\mathcal{B}'$ models φ' (general setting).*

Proof. Since $\mathcal{B}$ was obtained by cq-pp interpretation from $\mathcal{B}'$, it follows that $\mathcal{B}$ has also a majority polymorphism f (via the easy direction of the Galois connection of Lemma 1).

We now show that a collection of Skolem functions witnesses φ iff it does also for φ'. The right to left implication holds by construction and for any structure $\mathcal{B}'$. We only need to establish the left to right implication in the presence of an expanding majority f.

Let $g_1, g_2, \ldots, g_n$ be a collection of Skolem functions witnessing that $\mathcal{B} \models \varphi$. Let $b_1 \in g_1, b_2 \in g_2(b_1) \ldots b_n \in g_n(b_1, b_2, \ldots, b_{n-1})$. We write $g_i(\bar{b})$ as an abbreviation for $g_i(b_1, b_2, \ldots, b_{i-1})$.

Let $R(x_{i_1}, x_{i_2}, \ldots, x_{i_r})$ be some r-ary atom of φ' with $r \geq 3$. We write c_{i_j} to denote the counting requirement on variable i_j for $1 \leq j \leq r$.

Since $R_{x_{i_1},x_{i_2}}(g_{i_1}(\bar{b}), g_{i_2}(\bar{b}))$ holds in $\mathcal{B}$, by construction there are some set of values $S_{i_3}, S_{i_4}, \ldots, S_{i_r}$ of respective sizes $c_{i_3}, c_{i_4}, \ldots, c_{i_r}$. Similarly, there are some sets of the correct count such that $R(g_{i_1}(\bar{b}), S'_{i_2}, g_{i_3}(\bar{b}), S'_{i_4}, \ldots, S'_{i_r})$ and $R(S''_{i_1}, g_{i_2}(\bar{b}), g_{i_3}(\bar{b}), S''_{i_4}, \ldots, S''_{i_r})$. Applying f, since it is a majority, it means the following holds.

$$R(g_{i_1}(\bar{b}), g_{i_2}(\bar{b}), g_{i_3}(\bar{b}), f(S_{i_4}, S'_{i_4}, S''_{i_4}), \ldots, f(S_{i_r}, S'_{i_r}, S''_{i_r})).$$

Since it is expanding, we may select arbitrarily subsets $\widetilde{S_{i_4}} \subseteq f(S_{i_4}, S'_{i_4}, S''_{i_4}) \ldots \widetilde{S_{i_r}} \subseteq f(S_{i_r}, S'_{i_r}, S''_{i_r})$ of respective sizes $c_{i_4}, \ldots, c_{i_r}$ such that the following holds.

$$R(g_{i_1}(\bar{b}), g_{i_2}(\bar{b}), g_{i_3}(\bar{b}), \widetilde{S_{i_4}}, \ldots, \widetilde{S_{i_r}}).$$

Note that there is nothing special about the position 1, 2 and 3 within the tuple R. The same argument applies to any choice of three positions. Furthermore, there is nothing special in our argument using the fact that we have only three positions that agree with the value of the Skolem functions. So we can bootstrap the same argument to extend progressively the tuple by one position and show eventually that : $R(g_{i_1}(\bar{b}), g_{i_2}(\bar{b}), g_{i_3}(\bar{b}), \ldots, g_{i_r}(\bar{b}))$ holds. ◀

From now on, instead of considering a structure $\mathcal{B}'$, in the light of Proposition 1, we will concentrate on the corresponding binary structure $\mathcal{B}$ (to fulfill this we may need to expand the signature but it will still remain finite).

3.4 The Algorithm: path consistency for counting quantifiers (PCCQ)

Initialisation

- The domain of x_i contains all subsets that are consistent with all unary atoms involving x_i, that is $\{S \in \binom{B}{c_i}$ such that $S \subseteq M^{\mathcal{B}}$ for every unary atom $M(x_i)$ of $\varphi\}$
- For every binary relation $R_{i,j}$, the predicate $OK_{i,j}$ holds between any set S in the domain of x_i and c_i sets $T_1, \ldots, T_{c_i}$ in the domain of x_j whenever $R_{i,j}(s_k, T_k)$ holds for any $1 \leq k \leq c_i$

Maintaining consistency

Do
For all triples of variables $x_{i_1}, x_{i_2}, x_{i_3}$ (in the order of quantification),
For every distinct k, l in $\{i_1, i_2, i_3\}$,
For every S in the domain of x_k,
If there are no OK tuple $OK_{k,l}$ mentioning S (in the first coordinate), then
discard S and all other OK tuples that mention S.
For every $OK_{k,l}$ tuple t
if there are no additional OK tuples witnessing that t participates in a solution to $\varphi(x_{i_1}, x_{i_2}, x_{i_3})$
Remove the $OK_{k,l}$ tuple t.
If there are no more $OK_{k,l}$ tuples then reject.
Loop until no further OK tuples are deleted.

3.5 Properties of the PCCQ algoritm

▶ **Proposition 2.** *PCCQ runs in polynomial time.*

Proof. Let $\sharp v$ denote the number of variables of φ. The data structure needs to store at most $|B|^j \leq 2^{|B|}$ sets of size at most $j \leq |B|$ for each variable associated with a count j. One OK tuple originating from this variable with count j to a variable with count k will relate at most $j+1$ sets, one of size j and the others of size k. There are therefore at most $2^j.(2^k)^j \leq (2^{|B|})^{|B|+1}$ such OK tuples for one binary constraint. There are at most $\sharp v(\sharp v - 1)$ such constraints. The algorithm runs clearly in time polynomial in these quantities, and $(2^{|B|})^{|B|+1}$ is a constant since $|B|$ is fixed. ◀

Let $\overline{OK_{i,j}(S, T_1, \ldots, T_{c_i})}$ be a list of some OK tuples, as many as the arity of an expanding polymorphism f. Applying f coordinate wise, as we would for an ordinary tuple, we have $f(\overline{S}) = S'$ and $f(\overline{T_j}) = T'_j$ for any $1 \leq j \leq c_i$. However, the images $S', T'_1, \ldots, T'_{c_i}$ may be too large to feature in an OK tuple. We will say that an OK tuple (with aptly sized sets) $OK_{i,j}(S'', T''_1, \ldots, T''_{c_i})$ belongs to $f(\overline{OK_{i,j}(S, T_1, \ldots, T_{c_i})})$, whenever $S'' \subseteq S', T''_j \subseteq T'_j$ for all $1 \leq j \leq c_i$.

We say that a set $\mathcal{R}$ of OK tuples are *preserved* by f if, and only if, for any OK tuples $\overline{OK_{i,j}(S, T_1, \ldots, T_{c_i})}$ in $\mathcal{R}$, any OK tuple that belongs to $f(\overline{OK_{i,j}(S, T_1, \ldots, T_{c_i})})$, also belongs to $\mathcal{R}$.

▶ **Proposition 3.** *Let f be an expanding polymorphism of $\mathcal{B}$. If the algorithm PCCQ does not reject, the OK tuples that remain when the algorithm stops are preserved by f.*

Proof. Let $OK_{i,j}(S'', T''_1, \ldots, T''_{c_i})$ be an OK tuple in the image $f(\overline{OK_{i,j}(S, T_1, \ldots, T_{c_i})})$ under f of remaining OK tuples $\overline{OK_{i,j}(S, T_1, \ldots, T_{c_i})}$. We prove that $OK_{i,j}(S'', T''_1, \ldots, T''_{c_i})$ can not be removed by the algorithm as follows.

Initially, the relations are preserved under f, so it is straightforward to verify that OK tuples are also closed under f. So this removal of $OK_{i,j}(S'', T_1'', \ldots, T_{c_i}'')$ must happen after initialisation. We shall assume further that $OK_{i,j}(S'', T_1'', \ldots, T_{c_i}'')$ is the first OK tuple in the image of f of remaining OK tuples that is removed by the algorithm PCCQ.

Assume further that $OK_{i,j}(S'', T_1'', \ldots, T_{c_i}'')$ is removed by the algorithm while checking the sentence with some other variable k. Assume for now that the order of quantification induces the order i, j, k over the indices.

Since the tuples $\overline{OK_{i,j}(S, T_1, \ldots, T_{c_i})}$ are remaining OK tuples, there must be *remaining* tuples $OK_{i,k}$ and $OK_{j,k}$ witnessing that each of them participate in a solution to $\varphi(x_i, x_j, x_k)$.

Taking the image of these witnesses under f provide us with $OK_{i,k}$ and $OK_{i,k}$ witnessing that $OK_{i,j}(S'', T_1'', \ldots, T_{c_i}'')$ participate in a solution to $\varphi(x_i, x_j, x_k)$.

By time minimality of the removal of $OK_{i,j}(S'', T_1'', \ldots, T_{c_i}'')$, these last witnesses may not be remaining tuples but they must remain at the time of removal of $OK_{i,j}(S'', T_1'', \ldots, T_{c_i}'')$. This contradicts the fact that the algorithm could remove $OK_{i,j}(S'', T_1'', \ldots, T_{c_i}'')$.

To conclude the proof, note further that the above argument applies independently of the quantification order of i, j and k. ◀

▶ **Proposition 4.** *If $\mathcal{B}$ has an expanding majority f and the algorithm PCCQ does not reject, then $\mathcal{B}$ models φ.*

Proof. Let $x_1, x_2, \ldots, x_n$ be the variables occurring in φ. For any choice of variables $\overline{x}$ in $\{x_1, x_2, \ldots, x_n\}$, we denote by $\psi(\overline{x})$ the subsentence of φ induced by the variables $\overline{x}$.

We prove by induction on $2 \leq i < n$ that : for any choice of i variables $\overline{x}$, for any additional variable z occurring after the variables $\overline{x}$ in the order of quantification, any Skolem witnesses $\{g_1, g_2, \ldots, g_i\}$ for $\psi(\overline{x})$ can be extended by an i-ary Skolem function g_z for the variable z such that $\{g_1, g_2, \ldots, g_i, g_z\}$ witnesses that $\varphi(\overline{x}, z)$ holds. Moreover, this Skolem function ranges over sets that were not removed by the algorithm from the domain of z.

The base case for $i = 2$ holds : this is precisely the property that is enforced by the consistency algorithm we outlined.

We proceed to show the induction step. Let $x_1, x_2, x_3, \ldots, x_i$ be a choice of $i \geq 3$ variables and z a variable occurring after them. Let $\{g_1, g_2, g_3, \ldots, g_i\}$ be a collection of Skolem functions witnessing that $\psi(x_1, x_2, x_3, \ldots, x_i)$ holds.

We write I_1 for the image of g_1 and for $1 < j \leq i$, we write I_j for $g_j(I_1, \ldots, I_{j-1})$. Let $\alpha : \emptyset \to I_1$, $\beta : I_1 \to I_2$ and $\gamma : I_1 \times I_2 \to I_3$. We pick only such functions that are consistent with the fact that $\{g_1, g_2, g_3, \ldots, g_i\}$ are Skolem functions, namely we insist that for any b_1 in I_1, $\beta(b_1)$ belongs to the image of $g_2(b_1)$ and for any b_1 in I_1, and any b_2 in $g_2(b_1)$, $\gamma(b_1, b_2)$ lies in the image of $g_3(b_1, b_2)$.

We derive naturally three collections of $i - 1$ Skolem functions by essentially fixing the first, second or third coordinate of the i Skolem functions at hand. Each collection witnesses the subsentence obtained by removal of x_1, x_2 or x_3.

- Let the Skolem functions $\{g_2^\alpha, g_3^\alpha, \ldots, g_i^\alpha\}$ be defined as $g_j^\alpha(x_2, \ldots, x_{j-1}) = g_j(\alpha, x_2, \ldots, x_{j-1})$[1]. By construction, they are witnessing that $\psi(x_2, x_3, \ldots, x_i)$ holds. By the induction hypothesis, they can be extended by some $(i-1)$-ary function g_z^α witnessing $\psi(x_2, x_3, \ldots, x_i, z)$.

[1] If g_j is undefined, we let g_j^α be also undefined. Alternatively, we could have defined our Skolem functions precisely where we cared, e.g. for any x_2 in $g_1(\alpha)$, any x_3 in $g(\alpha, x_2)$, etc. But this would only introduce unnecessary notation.

- Similarly, we derive Skolem functions $\{g_1^\beta, g_3^\beta, \ldots, g_i^\beta\}$ witnessing $\psi(x_1, x_3, \ldots, x_i)$ from $\{g_1, g_2, \ldots, g_i\}$ by setting $g_1^\beta = g_1$ and for any $3 \leq j \leq i$, and any b_1 in I_1, we define $g_j^\beta(b_1, x_3, x_4, \ldots, x_{j-1}) := g_j(b_1, \beta(b_1), x_3, \ldots, x_{j-1})$. By the induction hypothesis, they can be extended by some $(i-1)$-ary function g_z^β witnessing $\psi(x_1, x_3, \ldots, x_i, z)$.
- Finally, we derive Skolem functions $\{g_1^\gamma, g_2^\gamma, g_4^\gamma \ldots, g_i^\gamma\}$ witnessing $\psi(x_1, x_2, x_4 \ldots, x_i)$ from $\{g_1, g_2, \ldots, g_i\}$ by setting $g_1^\gamma = g_1$, $g_2^\gamma = g_2$ and for any $4 \leq j \leq i$ any b_1 in I_1 and any b_2 in $g_2(b_1)$ that $g_j^\gamma(b_1, b_2, x_4, \ldots, x_{j-1}) := g_j(b_1, b_2, \gamma(b_1, b_2), x_4, \ldots, x_{j-1})$. By the induction hypothesis, they can be extended by some $(i-1)$-ary function g_z^γ witnessing $\psi(x_1, x_2, x_4, \ldots, x_i, z)$.

We will define the Skolem function g_z piecewise for each choice of the first three variables.

For specific b_1 in I_1 and b_2 in $g_2(b_1)$ and b_3 in $g_3(b_1, b_2)$, we set $\alpha() := b1$, $\beta(b_1) := b_2$, and $\gamma(b_1, b_2) := b_3$. The other values of β and γ are arbitrary but constrained as explained above.

Recall that f is an expanding majority of $\mathcal{B}$.

We define the Skolem function g_z as follows for this choice to the first three variables :

$$g_z(b_1, b_2, b_3, x_4, \ldots, x_i) := f(g_z^\alpha(b_2, b_3, x_4 \ldots, x_i), g_z^\beta(b_1, b_3, x_4 \ldots, x_i), g_z^\gamma(b_1, b_2, x_4 \ldots, x_i)).$$

The fact that f is expanding[2] implies that g_z has a range of correct size.

Note that this definition ensures that indeed g_z ranges over sets that were not filtered out by the algorithm from the domain of z by the (previous) Proposition 3.

The fact that f is a majority will allow us to derive that g_z is indeed an extension of $\{g_1, g_2, g_3, g_4, \ldots, g_i\}$ witnessing $\varphi(x_1, x_2, x_3, x_4 \ldots, x_i, z)$. We need only check this independently for each pair of variables x_j, z, since all atoms are binary. Since, we defined g_z piecewise, we can also check this independently for each piece, induced by the choices of b_1, b_2, b_3. For simplicity, we denote by R an atom that should hold between x_j and z.

- If $j \geq 4$, then applying majority on the variants α, β and γ works naturally, since the value for j is the same for each variant by construction and f is idempotent.
  With full notational details : by assumption $R(g_j^\alpha(b_2, b_3, x_4, \ldots, x_{j-1}), g_z^\alpha(b_2, b_3, x_4 \ldots, x_i))$ holds and $R(g_j^\beta(b_1, b_3, x_4, \ldots, x_{j-1}), g_z^\beta(b_1, b_3, x_4 \ldots, x_i))$ holds and

  $$R(g_j^\gamma(b_1, b_2, x_4, \ldots, x_{j-1}), g_z^\gamma(b_1, b_2, x_4 \ldots, x_i))$$

  holds. By construction of $g_j^\alpha, g_j^\beta, g_j^\gamma$ and the specific choice of values b_1, b_2, b_3, we have

  $$g_j^\alpha(b_2, b_3, x_4, \ldots, x_{j-1}) = g_j^\beta(b_1, b_3, x_4, \ldots, x_{j-1}) =$$
  $$g_j^\gamma(b_1, b_2, x_4, \ldots, x_{j-1}) = g_j(b_1, b_2, b_3, x_4, \ldots, x_{j-1}).$$

  Hence the image of the first coordinate under f is $g_j(b_1, b_2, b_3, x_4, \ldots, x_{j-1})$ since f is idempotent. The second coordinates is precisely the value we defined for g_z. Thus we conclude that $R(g_j(b_1, b_2, b_3, x_4, \ldots, x_{j-1}), g_z(b_1, b_2, b_3, x_4, \ldots, x_i))$ holds as required.
- If $j = 1$. The value for $g_z^\alpha(b_2, b_3, x_4 \ldots, x_i)$ occurs as a set in the domain of the variable z after variable x_1. So the algorithm must have left an OK tuple between x_1 and z that mentions $g_z^\alpha(b_2, b_3, x_4 \ldots, x_i)$. This means that there is a singleton b_1' such that $R(b_1', g_z^\alpha(b_2, b_3, x_4 \ldots, x_i))$ holds. Further, by assumption $R(g_1^\beta, g_z^\beta(b_1, b_3, x_4 \ldots, x_i))$ holds and $R(g_1^\gamma, g_z^\gamma(b_1, b_2, x_4 \ldots, x_i))$ holds. Since $g_1^\beta = g_1^\gamma = b_1$, applying f we obtain b_1 for the first coordinate since f is a majority operation. For the second coordinate we obtain the value we defined for g_z. Thus we conclude that $R(b_1, g_z(b_1, b_2, b_3, x_4, \ldots, x_i))$ holds as required.

- If $j = 2$, then similarly to the previous case, there is some singleton b'_2 in the domain of x_2 such that $R(b'_2, g^\beta_z(b_1, b_3, x_4 \ldots, x_i))$ holds. Further, by assumption $R(g^\alpha_2(b_1), g^\alpha_z(b_2, b_3, x_4 \ldots, x_i))$ holds and $R(g^\gamma_2(b_1), g^\gamma_z(b_1, b_2, x_4 \ldots, x_i))$ holds. Since $g^\alpha_2(b_1) = g^\gamma_2(b_1) = b_2$, applying f we obtain b_2 for the first coordinate since f is a majority operation. For the second coordinate we obtain the value we defined for g_z. Thus we conclude that $R(b_2, g_z(b_1, b_2, b_3, x_4, \ldots, x_i))$ holds as required.
- If $j = 3$, then similarly to the two previous cases, there is some singleton b'_3 in the domain of x_3 such that $R(b'_3, g^\gamma_z(b_1, b_2, x_4 \ldots, x_i))$ holds. Further, by assumption $R(g^\alpha_3(b_2), g^\alpha_z(b_2, b_3, x_4 \ldots, x_i))$ holds and $R(g^\beta_3(b_1), g^\beta_z(b_1, b_2, x_4 \ldots, x_i))$ holds. Since $g^\alpha_3(b_2) = g^\beta_3(b_1) = g_3(b_1, b_2) = b_3$, applying f we obtain b_3 for the first coordinate since f is a majority operation. For the second coordinate we obtain the value we defined for g_z. Thus we conclude that $R(b_3, g_z(b_1, b_2, b_3, x_4, \ldots, x_i))$ holds as required. ◀

We can now wrap-up to complete the proof of our main theorem.

Proof of Theorem 4. By Proposition 1, we reduce the question whether φ' holds on $\mathcal{B}'$ to the question whether φ holds on $\mathcal{B}$. This can be achieved in polynomial time, since we assume we assume a fixed signature, and have therefore bounded arity. We know that $\mathcal{B}$ is also preserved by the same expanding majority, thus we can appeal to Proposition 4, which states that if PCCQ does not reject then the sentence φ holds in $\mathcal{B}$. Since PCCQ runs in polynomial time by Proposition 2, we are done. ◀

Suppose now that X is some strict subset of $\{1, \ldots, |B|\}$. The variant of Lemma 1 that talks of X-pp-definability and polymorphisms that expand at cardinalities in X is not explicit in [8]. However, the easy direction, that X-pp-definability entails preservation by polymorphisms that expand at cardinalities in X, is straightforward to prove.

▶ **Theorem 5.** *Suppose $\mathcal{B}$ has an majority polymorphism that expands at cardinalities $\{c_1, \ldots, c_m\}$. Then $\{c_1, \ldots, c_m\}$-CSP($\mathcal{B}$) is in* P.

3.6 Expanding polymorphisms are necessary

We will now argue that the condition of expansion was necessary in Theorem 4, since there is a structure admitting non-expanding majority whose CQCSP is NP-hard. Let $\mathcal{H}_4$ be the 4-vertex graph built from the irreflexive triangle $\mathcal{K}_3$ on $\{1, 2, 3\}$ by adding a dominating vertex 0 with a self-loop. It is easy to verify that $\mathcal{H}_4$ enjoys the majority polymorphism f that maps any tuple of distinct arguments to 0. This f is clearly not conservative and it even violates the condition of expansion because $|f(\{0, 1\}, \{0, 2\}, \{0, 3\})| = 1$.

▶ **Lemma 6.** *CQCSP($\mathcal{H}_4$) is* NP*-hard.*

Proof. By reduction from 3-COL, a.k.a. CSP($\mathcal{K}_3$). Take an input φ for CSP($\mathcal{K}_3$) and build an input ψ for CQCSP($\mathcal{H}_4$) by changing all $\exists$ quantifiers to $\exists^{\geq 2}$.

($\mathcal{K}_3 \models \varphi$ implies $\mathcal{H}_4 \models \psi$.) Evaluate each variables v in ψ according to its evaluation φ but additionally with the second possibility 0.

($\mathcal{H}_4 \models \psi$ implies $\mathcal{K}_3 \models \varphi$.) Evaluate each variable v in φ according to one of the possibilities for v in ψ that is not equal to 0. ◀

4 Applications of our result

We will now see that conservative majority polymorphisms demarcate tractability in diverse places.

▶ **Corollary 7.** *Let $\mathcal{H}^*$ be a partially reflexive graph $\mathcal{H}$ endowed with all unary relations. Either $\mathcal{H}^*$ admits an expanding majority and CQCSP($\mathcal{H}^*$) is in* P*, or CQCSP($\mathcal{H}^*$) is* NP*-hard.*

Proof. We know all polymorphisms of $\mathcal{H}^*$ are conservative since it has all unary relations. From Theorem 2 we further know that either $\mathcal{H}^*$ admits a conservative majority polymorphism or CSP($\mathcal{H}^*$) is NP-hard. The result follows from Lemma 3 and Theorem 4. ◀

The following is a strengthening of Theorem 7.16 of [21] in the case of paths.[3]

▶ **Corollary 8.** *Let $\mathcal{P}$ be an irreflexive (undirected) path. Then CQCSP($\mathcal{P}$) is in* P*.*

Proof. Suppose $\mathcal{P}$ is over vertices $\{1, \ldots, n\}$ so that $(i, i+1) \in E^{\mathcal{P}}$. Then $\mathcal{P}$ admits the conservative majority polymorphism m communicated to us by Tomás Feder: $m(x, y, z)$ is defined to be the median of x, y, z, if they all have the same parity; otherwise it is the smaller of the pair with repeated parity. The result follows from Lemma 3 and Theorem 4. ◀

Sadly we cannot use conservative majorities for irreflexive trees, since it is well-known that the tree $\mathcal{T}_{10}$, built from three paths on four vertices by identifying one end of each of these three paths as a single vertex, does not admit a conservative majority. This has been known, based on complexity-theoretic assumptions, since [14, 4] but we have checked also using the polymorphism program of Miklós Maróti[4].

We will now see how to apply our result to infinite-domain (CQ)CSPs. The (d-)*modular median* operation of [1] is defined on $\mathbb{Z}$ as follows. $f(x, y, z) = median(x, y, z)$, if $x \equiv y \equiv z \bmod d$. If two among $\{x, y, z\}$ are equivalent mod d, then $f(x, y, z)$ is the minimum of these two; otherwise $f(x, y, z) = x$. Note that these modular median operations are conservative majorities.

▶ **Corollary 9.** *Let $\mathcal{B}$ be a finite-signature binary first-order expansion of $(\mathbb{Z}; succ)$ whose relations, viewed as digraphs, have bounded degree. Either $\mathcal{B}$ admits a modular median polymorphism, and, for each j, $\{1, \ldots, j\}$-CSP($\mathcal{B}$) is in* P*, or CSP($\mathcal{B}$) is* NP*-hard.*

Proof. By Proposition 6 in [1],[5] we know that if $\mathcal{B}$ omits all modular median operations, then CSP($\mathcal{B}$) is NP-hard. Thus, we are left with the question of tractability. Let e be maximal so that $(x, x+e)$ appears in some relation of $\mathcal{B}$. Let ϕ be an input for CQCSP($\mathcal{B}$) involving n variables. Now, we can see that ϕ is true on $\mathcal{B}$ just in case it is true on the substructure $\mathcal{B}'$of $\mathcal{B}$ induced by the interval $[0, ne]$. $\mathcal{B}'$ admits the same conservative majority that $\mathcal{B}$ does and the result follows from Propositions 4 when we consider from the proof of Proposition 2 that the size of subsets in the OK tuples is bounded by j. This is because the number of OK tuples per binary constraint is bound by $(j(ne)^j)^{j+1}$ (which takes the place of the term $(2^{|B|})^{|B|+1}$ in the calculation for complexity in Proposition 2). ◀

To consider CQCSP over an infinite-domain structure, albeit with a finite signature, one must consider how to encode i in $\exists^{\geq i}$. The most natural encoding here is binary. We leave as an open question whether CQCSP($\mathcal{B}$) is in P, whenever $\mathcal{B}$ is a finite-signature binary first-order expansion of $(\mathbb{Z}; succ)$ whose relations, viewed as digraphs, have bounded degree,

[3] Theorem 7.16 of [21] deals with $\{1, 2\}$-CSP on trees, but its very long proof does not become much simpler if one restricts to paths.

[4] See: `http://www.math.u-szeged.hu~maroti/applets/GraphPoly.html`

[5] Proposition 6 lacks a counterpart in the journal version of [1] For a proof, see Proposition 35 in v2 of the arxiv version.

which admits a modular median polymorphism. Note that this question remains open even if we choose the unary encoding for i.

4.1 Endo-triviality

The concept of endo-triviality has recently been introduced in the context of surjective CSPs [20]. We note here that endo-triviality is strong enough to deduce results also for CQCSPs. An *endomorphism* of a digraph $\mathcal{H}$ is a homomorphism from $\mathcal{H}$ to itself. Call $\mathcal{H}$ a *core* if all of its endomorphisms of $\mathcal{H}$ are automorphisms (the importance of cores is discussed, e.g., in [17]). Call $\mathcal{H}$ *endo-trivial* if all of its endomorphisms either have range of size 1 or are automorphisms.

The *retraction* problem $\mathrm{Ret}(\mathcal{H})$ takes as input a graph $\mathcal{G}$ containing $\mathcal{H}$ as an induced substructure and asks whether there is a homomorphism from $\mathcal{G}$ to $\mathcal{H}$ that is the identity on $\mathcal{H}$ (such an endomorphism of $\mathcal{G}$ is termed a *retraction to* $\mathcal{H}$)

The proofs of the following are deferred to the appendix.

▶ **Lemma 10.** *Let $\mathcal{H}$ be a graph that is endo-trivial. The there is a polynomial-time reduction from Ret($\mathcal{H}$) to CQCSP($\mathcal{H}$).*

▶ **Corollary 11.** *Let $\mathcal{C}$ be a reflexive directed cycle. If $\mathcal{C}$ is of length 2 then CQCSP($\mathcal{C}$) is in* L*, otherwise CQCSP($\mathcal{C}$) is* NP*-hard.*

5 Final remarks

Near-unanimity polymorphisms. Note that Theorem 4 relativises to any subset of counts $X \subset \{1, 2 \ldots, |B|\}$ for the problem X-CSP($\mathcal{B}$) with the weaker hypothesis that requires that $\mathcal{B}$ has a majority f that is expanding on X. Note that, if $1 \notin X$, one has to move to partial polymorphisms. Indeed, we do not need f to be a majority, only that it satisfies the identities of a majority where we replace uniformly the variables by set variables of the same size from X.

We can also generalise the algorithm and the proof principle to a larger class of structures.

▶ **Theorem 12.** *If $\mathcal{B}$ has an expanding near unanimity polymorphism. Then CQCSP($\mathcal{B}$) is in* P*.*

CQCSP and List-CSP. We have seen that conservative operations are expanding, but what is the actual relationship between CQCSP and List-CSP? Does ability to quantify set cardinalities with $\exists^{\geq j}$ relate to talking about subsets of size j? For this latter question, it seems the answer is no. Designate $\{1, 2\}$-List-CSP the restriction of List-CSP in which only subsets of size 1 and 2 are available. Recall the tree $\mathcal{T}_{10}$, built from three paths on four vertices by identifying one end of each of these three paths as a single vertex. List-CSP($\mathcal{T}_{10}$) is known to be NP-complete since [14]. NP-completeness for $\{1, 2\}$-List-CSP($\mathcal{T}_{10}$) follows from [4]. On the other hand, $\{1, 2\}$-CSP($\mathcal{T}_{10}$) is in P, as proved in Theorem 7.16 of [21]. However, we are still missing an exemplar $\mathcal{B}$ so that one of CQCSP($\mathcal{B}$) and List-CSP($\mathcal{B}$) is tractable and the other is not.

CQCSP and Retraction. In Lemma 10, we show a sufficient condition for which Ret($\mathcal{B}$) is polynomially reducible to CQCSP($\mathcal{B}$). It should be possible to reconstruct the argument from [20] in order to prove that, if $\mathcal{H}$ is a reflexive tournament, then either $\mathcal{H}$ has a conservative majority polymorphism (the median) and CQCSP($\mathcal{B}$) is in P; or Ret($\mathcal{H}$) can be polynomially

reduced to CQCSP($\mathcal{H}$) and both are NP-hard. Note that a classification for QCSP on reflexive tournaments is not yet known. However, what we would like is much stronger : is it the case that for all finite $\mathcal{B}$, Ret($\mathcal{H}$) can be polynomially reduced to CQCSP($\mathcal{H}$)? That is, are all constants cq-pp-definable up to isomorphism?

Core-ness and finite categoricity. Closely related to the previous question is whether all non-isomorphic finite structures can be distinguished by cq-pp. Let us explore this question through the Weisfeiler-Lehman (WL) method, as discussed in [9] (where logics with counting also play a central role). The *degree sequence* of a graph is a non-increasing list of positive integers that list the degrees of its vertices. This can be thought of as a 0-dimensional WL descriptor. Obviously, if two graphs are isomorphic, then they have the same degree sequence, but the converse is not necessarily true. Cq-pp can not specify vertex degree but it can specify a lower bound for it. Firstly, then, two graphs on vertex sets of distinct sizes can be distinguished by some $\exists^{\geq a_1} x\,(x = x)$. For two graphs with vertex sets the same size, if their two degree sequences differ, with the first being lexicographically the larger, then counting down from the top until the first difference, one will find necessarily some a_1, a_2 so that $\exists^{\geq a_1} x_1 \exists^{\geq a_2} x_2\, E(x_1, x_2)$ is true on the first graph but false on the second. We do this by setting $a_1 - 1$ to be the number of vertices before the degree sequence differs and a_2 to be the degree at which the degree sequences diverge.

The 1-dimensional WL descriptor is defined inductively by expanding each integer associated with a vertex from the 0-dimensional WL descriptor into a tree of depth one whose leaves list, in descending order, the degrees of that vertex's neighbours. These leaves are now associated with that corresponding neighbour. The process is then iterated, and would go on for ever, save that we stop it when a fixed-point is reached in terms of the subtrees added being endlessly the same. Now suppose two graphs each give rise to a forest built in this fashion and let k be the height at which these forests first differ (else they are indistinguishable by 1-dimensional WL) and let the first graph be lexicographically the smaller (apply closeness to the root as higher in the lexicography). We can follow the previous reasoning, and the path through the forests on which the graphs differ, to find some $\exists^{\geq a_1} x_1 \exists^{\geq a_2} x_2 \ldots \exists^{\geq a_k} x_k \exists^{\geq a_{k+1}} x_{k+1}\, E(x_1, x_2) \wedge \ldots \wedge E(x_k, x_{k+1})$ that is true on the first graph but not the second.

The 1-dimensional WL descriptor does not capture isomorphism, and unfortunately, we do not see an implementation of the more general r-dimensional WL descriptor in cq-pp, since this can measure isomorphism type of an induced subgraph of size r.

References

1 Manuel Bodirsky, Víctor Dalmau, Barnaby Martin, and Michael Pinsker. Distance constraint satisfaction problems. In *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, pages 162–173, 2010. doi:10.1007/978-3-642-15155-2_16.

2 F. Börner, A. Krokhin, A. Bulatov, and P. Jeavons. Quantified constraints and surjective polymorphisms. Technical Report PRG-RR-02-11, Oxford University, 2002.

3 Richard C. Brewster, Tomás Feder, Pavol Hell, Jing Huang, and Gary MacGillivray. Near-unanimity functions and varieties of reflexive graphs. *SIAM J. Discrete Math.*, 22(3):938–960, 2008. doi:10.1137/S0895480103436748.

4 A. Bulatov. Tractable conservative constraint satisfaction problems. In *Proceedings of LICS'03*, pages 321–330, 2003.

5 A. Bulatov, A. Krokhin, and P. G. Jeavons. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34:720–742, 2005.

6 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. *Proc. FOCS 2017*, pages 319–330, 2017.

7 Andrei A. Bulatov and Amir Hedayaty. Counting predicates, subset surjective functions, and counting csps. In *42nd IEEE International Symposium on Multiple-Valued Logic, ISMVL 2012*, pages 331–336, 2012.

8 Andrei A. Bulatov and Amir Hedayaty. Galois correspondence for counting quantifiers. *Multiple-Valued Logic and Soft Computing*, 24:405–424, 2015.

9 Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, Dec 1992. doi:10.1007/BF01305232.

10 Hubie Chen. The complexity of quantified constraint satisfaction: Collapsibility, sink algebras, and the three-element case. *SIAM J. Comput.*, 37(5):1674–1701, 2008. doi:10.1137/060668572.

11 Robin Clark and Murray Grossman. Number sense and quantifier interpretation. *Topoi*, 26(1):51–62, 2007.

12 H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1999. 2nd edition.

13 T. Feder and M. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal on Computing*, 28:57–104, 1999.

14 Tomás Feder, Pavol Hell, and Jing Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19(4):487–505, 1999. URL: http://link.springer.de/link/service/journals/00493/bibs/9019004/90190487.htm.

15 Tomás Feder, Pavol Hell, and Jing Huang. Bi-arc graphs and the complexity of list homomorphisms. *J. Graph Theory*, 42:61–80, 2003.

16 E.C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982.

17 P. Hell and J. Nešetřil. *Graphs and Homomorphisms*. OUP, 2004.

18 Peter Jeavons, David Cohen, and Martin Cooper. Constraints, consistency and closure. *AI*, 101(1-2):251–265, 1998.

19 P. G. Kolaitis and M. Y. Vardi. *Finite Model Theory and Its Applications (Texts in Theoretical Computer Science. An EATCS Series)*, chapter A logical Approach to Constraint Satisfaction. Springer-Verlag New York, Inc., 2005.

20 Benoit Larose, Barnaby Martin, and Daniël Paulusma. Surjective h-colouring over reflexive digraphs. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 49:1–49:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.49.

21 Barnaby Martin, Florent R. Madelaine, and Juraj Stacho. Constraint satisfaction with counting quantifiers. *SIAM J. Discrete Math.*, 29(2):1065–1113, 2015. doi:10.1137/140981332.

22 M. Otto. *Bounded variable logics and counting – A study in finite models*, volume 9. Springer-Verlag, 1997. IX+183 pages.

23 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. *Proc. FOCS 2017*, pages 331–342, 2017.

The b-Branching Problem in Digraphs

Naonori Kakimura[1]
Keio University, Kanagawa 223-8522, Japan
kakimura@math.keio.ac.jp

Naoyuki Kamiyama[2]
Kyushu University and JST, PRESTO, Fukuoka 819-0395, Japan
kamiyama@imi.kyushu-u.ac.jp

Kenjiro Takazawa[3]
Hosei University, Tokyo 184-8584, Japan
takazawa@hosei.ac.jp

Abstract

In this paper, we introduce the concept of b-branchings in digraphs, which is a generalization of branchings serving as a counterpart of b-matchings. Here b is a positive integer vector on the vertex set of a digraph, and a b-branching is defined as a common independent set of two matroids defined by b: an arc set is a b-branching if it has at most $b(v)$ arcs sharing the terminal vertex v, and it is an independent set of a certain sparsity matroid defined by b. We demonstrate that b-branchings yield an appropriate generalization of branchings by extending several classical results on branchings. We first present a multi-phase greedy algorithm for finding a maximum-weight b-branching. We then prove a packing theorem extending Edmonds' disjoint branchings theorem, and provide a strongly polynomial algorithm for finding optimal disjoint b-branchings. As a consequence of the packing theorem, we prove the integer decomposition property of the b-branching polytope. Finally, we deal with a further generalization in which a matroid constraint is imposed on the $b(v)$ arcs sharing the terminal vertex v.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Greedy Algorithm, Packing, Matroid Intersection, Sparsity Matroid, Arborescence

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.12

Related Version A full version of the paper is available at [31], `https://arxiv.org/abs/1802.02381`.

1 Introduction

Since the pioneering work of Edmonds [12, 14], the importance of *matroid intersection* has been well appreciated. A special case of matroid intersection is *branchings* (or *arborescences*) in digraphs. Branchings have several good properties which do not hold for general matroid intersection. The objective of this paper is to propose a class of the matroid intersection problem which generalizes branchings and inherits those good properties of branchings.

[1] Supported by JST ERATO Grant Number JPMJER1201, JSPS KAKENHI Grant Number JP17K00028, Japan.
[2] Supported by JST PRESTO Grant Number JPMJPR14E1, Japan.
[3] Supported by JST CREST Grant Number JPMJCR1402, JSPS KAKENHI Grant Numbers JP16K16012, JP26280001, Japan.

One of the good properties of branchings is that a maximum-weight branching can be found by a simple combinatorial algorithm [4, 6, 11, 23]. This algorithm is much simpler than general weighted matroid intersection algorithms, and is referred to as a "multi-phase greedy algorithm" in the textbook by Kleinberg and Tardos [35].

Another good property is the elegant theorem for packing disjoint branchings [13]. In terms of matroid intersection, this theorem says that, if there exist k disjoint bases in each of the two matroids, then there exist k disjoint common bases. This packing theorem leads to a proof that the branching polytope has the *integer decomposition property* (defined in Section 2).

In this paper, we propose *b-branchings*, a class of matroid intersection generalizing branchings, while maintaining the above two good properties. This offers a new direction of fundamental extensions of the classical theorems on branchings.

Let $D = (V, A)$ be a digraph and let $b \in \mathbb{Z}^V_{++}$ be a positive integer vector on V. For $v \in V$ and $F \subseteq A$, let $\delta^-_F(v)$ denote the set of arcs in F entering v, and let $d^-_F(v) = |\delta^-_F(v)|$. One matroid $\mathbf{M}_{\mathrm{in}}$ on A has its independent set family $\mathcal{I}_{\mathrm{in}}$ defined by

$$\mathcal{I}_{\mathrm{in}} = \{F \subseteq A \colon d^-_F(v) \leq b(v) \text{ for each } v \in V\}. \tag{1}$$

That is, $\mathbf{M}_{\mathrm{in}}$ is the direct sum of a uniform matroid on $\delta^-_A(v)$ of rank $b(v)$ for every $v \in V$. Hence, each vertex can have indegree at most $b(v)$, which can be more than one. Indeed, this is the reason why we refer to it as a b-branching, as a counterpart of a b-matching.

In order to make b-branchings a satisfying generalization of branchings, the other matroid should be defined appropriately. Our answer is a *sparsity matroid* determined by D and b, which is defined as follows. For $F \subseteq A$ and $X \subseteq V$, let $F[X]$ denote the set of arcs in F induced by X. Also, denote $\sum_{v \in X} b(v)$ by $b(X)$. Now define a matroid $\mathbf{M}_{\mathrm{sp}}$ on A with independent set family $\mathcal{I}_{\mathrm{sp}}$ by

$$\mathcal{I}_{\mathrm{sp}} = \{F \subseteq A \colon |F[X]| \leq b(X) - 1 \ (\emptyset \neq X \subseteq V)\}. \tag{2}$$

It is known that $\mathbf{M}_{\mathrm{sp}}$ is a matroid [20, Theorem 13.5.1], referred to as a *count matroid* or a *sparsity matroid*.

Now we refer to an arc set $F \subseteq A$ as a *b-branching* if $F \in \mathcal{I}_{\mathrm{in}} \cap \mathcal{I}_{\mathrm{sp}}$. It is clear that a branching is a special case of a b-branching where $b(v) = 1$ for each $v \in V$. We demonstrate that b-branchings yield a reasonable generalization of branching by proving that the two fundamental results on branchings can be extended. That is, we present a multi-phase greedy algorithm for finding a maximum-weight b-branching, and a theorem for packing disjoint b-branchings.

Our multi-phase greedy algorithm is an extension of the weighted branching algorithm [4, 6, 11, 23], and it has the following features. First, its running time is $\mathrm{O}(|V||A|)$, which is as fast as a simple implementation of the weighted branching algorithm [4, 6, 11, 23], and faster than the current best general weighted matroid intersection algorithm. Second, our algorithm also finds an optimal dual solution, which is integer if the arc weights are integer. Thus, the algorithm constructively proves the total dual integrality of the associated linear system. Finally, the algorithm leads to a characterization of the existence of a b-branching with prescribed indegree, which is a generalization of that for an arborescence [4, 11, 23].

This characterization theorem is extended to a theorem on packing disjoint b-branchings. Let k be a positive integer, and $b_1, \ldots, b_k$ be nonnegative integer vectors on V such that $b_i(v) \leq b(v)$ for each $v \in V$ and $b_i \neq b$ $(i = 1, \ldots, k)$. Note that, when there exists a b-branching B_i satisfying $d^-_{B_i}(v) = b_i(v)$ for each $v \in V$ $(i = 1, \ldots, k)$, these assumptions about b_i follow from the definition (1) and (2) of b-branchings. We provide a necessary and sufficient

condition for D to contain k disjoint b-branchings $B_1, \ldots, B_k$ satisfying $d^-_{B_i}(v) = b_i(v)$ for every $v \in V$ and $i = 1, \ldots, k$, which extends Edmonds' disjoint branching theorem [13]. We then show such disjoint b-branchings $B_1, \ldots, B_k$ can be found in strongly polynomial time by at most $|A|$ times of submodular function minimization [28, 37, 46]. We further prove that, when the arc-weight vector $w \in \mathbb{R}^A_+$ is given, disjoint b-branchings $B_1, \ldots, B_k$ that minimize $w(B_1) + \cdots + w(B_k)$ can be found in strongly polynomial time by optimization over a submodular flow polyhedron [16, 21, 29, 30]. By utilizing our disjoint b-branchings theorem, we also prove the integer decomposition property of the b-branching polytope.

We further deal with a generalized class of *matroid-restricted b-branchings*. This is a special case of matroid intersection in which $\mathbf{M}_{\mathrm{in}}$ is the direct sum of an arbitrary matroid on $\delta^-_A(v)$ of rank $b(v)$ for all $v \in V$. Note that, in the class of b-branchings, the matroid $\mathbf{M}_{\mathrm{in}}$ is the direct sum of a uniform matroid on $\delta^-_A(v)$ of rank $b(v)$. We show that our multi-phase greedy algorithm can be extended to this generalized class.

Let us conclude this section with describing related work. The weighted matroid intersection problem is a common generalization of various combinatorial optimization problems such as bipartite matchings, packing spanning trees, and branchings (or arborescences) in a digraph. The problem has also been applied to various engineering problems, e.g., in electric circuit theory [43, 44], rigidity theory [44], and network coding [8, 25]. Since 1970s, quite a few algorithms have been proposed for matroid intersection problems, e.g., [5, 18, 27, 37, 39, 40] (See [26] for further references). However, all known algorithms are not greedy, but based on augmentation; repeatedly incrementing a current solution by exchanging some elements.

The matroids in branchings are a partition matroid and a graphic matroid, which are interconnected by a given digraph. Such interconnection makes branchings more interesting. As mentioned before, branchings have properties that matroid intersection of an arbitrary pair of a partition matroid and a graphic matroid does not have. In particular, extending the packing theorem of branchings [13] is indeed a recent active topic. Kamiyama, Katoh, and Takizawa [33] presented a fundamental extension based on reachability in digraphs, which is followed by a further extension based on convexity in digraphs due to Fujishige [22]. Durand de Gevigney, Nguyen, and Szigeti [9] proved a theorem for packing arborescences with matroid constraints. Király [34] generalized the result of [9] in the same direction of [33]. A matroid-restricted packing of arborescences [3, 19] is another generalization concerning a matroid constraint. We remark that our packing and matroid restriction for b-branchings differ from the above matroidal extensions of packing of arborescences.

The organization of this paper is as follows. In Section 2, we review the literature of branchings and matroid intersection, including algorithmic, polyhedral, and packing results. In Section 3, we present a multi-phase greedy algorithm for finding a maximum-weight b-branching. Section 4 is devoted to proving a theorem on packing disjoint b-branchings. In Section 5, we extend the multi-phase greedy algorithm to matroid-restricted b-branchings. In Section 6, we conclude this paper with a couple of remarks.

2 Preliminaries

In this section, we review fundamental results on branchings and related theory of matroid intersection and polyhedral combinatorics. For more details, refer to [32, 36, 47].

In a digraph $D = (V, A)$, an arc subset $B \subseteq A$ is a *branching* if, in the subgraph (V, B), the indegree of every vertex is at most one and there does not exist a cycle in the undirected sense. In terms of matroid intersection, a branching is a common independent set of a

partition matroid and a graphic matroid, i.e., intersection of

$$\{F \subseteq A\colon d^-_F(v) \leq 1 \text{ for each } v \in V\}, \tag{3}$$
$$\{F \subseteq A\colon |F[X]| \leq |X| - 1 \ (\emptyset \neq X \subseteq V)\}. \tag{4}$$

Recall that a branching is a special case of a b-branching where $b(v) = 1$ for each $v \in V$. Indeed, by putting $b(v) = 1$ for each $v \in V$ in (1) and (2), we obtain (3) and (4), respectively.

As stated in Section 1, a maximum-weight branching can be found by a multi-phase greedy algorithm [4, 6, 11, 23], which appears in standard textbooks such as [35, 36, 47]. To the best of our knowledge, we have no other nontrivial special case of matroid intersection which can be solved greedily. For example, intersection of two partition matroids is equivalent to bipartite matching. This seems the simplest nontrivial example of matroid intersection, but we do not know a greedy algorithm for finding a maximum bipartite matching.

Another important result on branchings is the disjoint branchings theorem by Edmonds [13], described as follows. For a positive integer k, the set of integers $\{1, \ldots, k\}$ is denoted by $[k]$. For $F \subseteq A$ and $X \subseteq V$, let $\delta^-_F(X) \subseteq A$ denote the set of arcs in F from $V \setminus X$ to X, and let $d^-_F(X) = |\delta^-_F(X)|$.

▶ **Theorem 1** (Edmonds [13]). *Let $D = (V, A)$ be a digraph and k be a positive integer, and $U_1, \ldots, U_k$ be subsets of V. Then, there exist disjoint branchings $B_1, \ldots, B_k$ such that $U_i = \{v \in V\colon d^-_{B_i}(v) = 1\}$ for each $i \in [k]$ if and only if*

$$d^-_A(X) \geq |\{i \in [k]\colon X \subseteq U_i\}| \quad (\emptyset \neq X \subseteq V).$$

From Theorem 1, we obtain a theorem on covering a digraph by branchings [17, 41].

▶ **Theorem 2** ([17, 41]). *Let $D = (V, A)$ be a digraph and let k be a nonnegative integer. Then, the arc set A can be covered by k branchings if and only if*

$$d^-_A(v) \leq k \quad (v \in V),$$
$$|A[X]| \leq k(|X| - 1) \quad (\emptyset \neq X \subseteq V).$$

Theorem 2 leads to the *integer decomposition property* of the branching polytope. The *branching polytope* is a convex hull of the characteristic vectors of all branchings. It follows from the total dual integrality of matroid intersection [12] that the branching polytope is determined by the following linear system:

$$x(\delta^-(v)) \leq 1 \qquad (v \in V), \tag{5}$$
$$x(A[X]) \leq |X| - 1 \quad (\emptyset \neq X \subseteq V), \tag{6}$$
$$x(a) \geq 0 \qquad (a \in A). \tag{7}$$

▶ **Theorem 3** (see [47]). *The linear system (5)–(7) is totally dual integral.*

▶ **Corollary 4** (see [47]). *The linear system (5)–(7) determines the branching polytope.*

For a polytope P and a positive integer k, define $kP = \{x\colon \exists x' \in P, x = kx'\}$. A polytope P has the *integer decomposition property* if, for each positive integer k, any integer vector $x \in kP$ can be represented as the sum of k integer vectors in P. The integer decomposition property of the branching polytope is a direct consequence of Theorem 2 and Corollary 4.

▶ **Corollary 5** ([1]). *The branching polytope has the integer decomposition property.*

We remark that the integer decomposition property does not hold for an arbitrary matroid intersection polytope. Schrijver [47] presents an example of matroid intersection defined on the edge set of K_4 without integer decomposition property. Indeed, finding a class of polyhedra with integer decomposition property is a classical topic in combinatorics. Typical examples of polyhedra with integer decomposition property include polymatroids [1, 24], the branching polytope [1], and intersection of two strongly base orderable matroids [7, 42]. While there is some recent progress [2], the integer decomposition property of polyhedra is far from being well understood. In Section 4, we will prove that the b-branching polytope is a new example of polytopes with integer decomposition property.

3 Multi-phase greedy algorithm

3.1 Algorithm description

In this subsection, we present a multi-phase greedy algorithm for finding a maximum-weight b-branching by extending the one for branchings [4, 6, 11, 23]. Let $D = (V, A)$ be a digraph and $b \in \mathbb{Z}^V_{++}$ be a positive integer vector on V. Recall that an arc set $F \subseteq A$ is a b-branching if $F \in \mathcal{I}_{\mathrm{in}} \cap \mathcal{I}_{\mathrm{sp}}$, where $\mathcal{I}_{\mathrm{in}}$ and $\mathcal{I}_{\mathrm{sp}}$ are defined by (1) and (2), respectively.

We first show a key property of $\mathbf{M}_{\mathrm{in}}$ and $\mathbf{M}_{\mathrm{sp}}$, which plays an important role in our algorithm. Its proof can be found in the full version [31].

▶ **Lemma 6.** *An independent set F in $\mathbf{M}_{\mathrm{in}}$ is not independent in $\mathbf{M}_{\mathrm{sp}}$ if and only if (V, F) has a strong component X such that*

$$|F[X]| = b(X). \tag{8}$$

Moreover, for every strong component X in (V, F) satisfying (8), $F[X]$ is a circuit of $\mathbf{M}_{\mathrm{sp}}$.

Lemma 6 enables us to design the following multi-phase greedy algorithm for finding a maximum-weight b-branching:

- Find a maximum-weight independent set F in $\mathbf{M}_{\mathrm{in}}$.
- If (V, F) has a strong component X satisfying (8), then contract X, reset b and the weights of the remaining arcs appropriately, and recurse.

At the end of the algorithm, we expand every contracted component X in the following manner. Suppose that the solution F has an arc a' entering the vertex v_X created when contracting X. Denote the terminal vertex of a' before contracting X by $v' \in X$. In expanding X, we add $b(X) - 1$ arcs to F, consisting of $b(v)$ heaviest arcs among $\delta^-_A(v) \cap A[X]$ for each $v \in X \setminus \{v'\}$ and $b(v') - 1$ heaviest arcs among $\delta^-_A(v') \cap A[X]$. If F has no arc entering v_X, then we add $b(X) - 1$ arcs to F, consisting of $b(v)$ heaviest arcs among $\delta^-_A(v) \cap A[X]$ for each $v \in X$ except for the arc of minimum weight among those $b(X)$ arcs.

A formal description of the algorithm is as follows. We denote an arc $a \in A$ with initial vertex u and terminal vertex v by (u, v). We assume that the arc weights are nonnegative, which are represented by a vector $w \in \mathbb{R}^A_+$. For $F \subseteq A$, we denote $w(F) = \sum_{a \in F} w(a)$.

The complexity of Algorithm bB is analyzed as follows. It is clear that there are at most $|V|$ iterations. It is also straightforward to see that the i-th iteration requires $O(|A^{(i)}|)$ time: Steps 2, 3, and 4 respectively require $O(|A^{(i)}|)$ time. Thus, the total time complexity of the algorithms is $O(|V||A|)$.

3.2 Optimality of the algorithm and totally dual integral system

In this subsection, we prove that the output of Algorithm bB is a maximum-weight b-branching by the following primal-dual argument. We first present a linear program describing

Algorithm 1 ALGORITHM bB.

Input. A digraph $D = (V, A)$, and vectors $b \in \mathbb{Z}^V_{++}$ and $w \in \mathbb{R}^A_+$.
Output. A b-branching $F \subseteq A$ maximizing $w(F)$.
Step 1. Set $i := 0$, $D^{(0)} := D$, $b^{(0)} := b$, and $w^{(0)} := w$.
Step 2. Define a matroid $\mathbf{M}^{(i)}_{\mathrm{in}} = (A^{(i)}, \mathcal{I}^{(i)}_{\mathrm{in}})$ accordingly to $D^{(i)}$ and $b^{(i)}$ by (1). Then, find $F^{(i)} \in \mathcal{I}^{(i)}_{\mathrm{in}}$ maximizing $w^{(i)}(F^{(i)})$.
Step 3. If $(V^{(i)}, F^{(i)})$ has a strong component X such that

$$|F^{(i)}[X]| = b^{(i)}(X), \tag{9}$$

then go to Step 4. Otherwise, let $F := F^{(i)}$ and go to Step 5.
Step 4. Denote by $\mathcal{X} \subseteq 2^{V^{(i)}}$ the family of strong components X in $(V^{(i)}, F^{(i)})$ satisfying (9). Execute the following updates to construct $D^{(i+1)} = (V^{(i+1)}, A^{(i+1)})$, $b^{(i+1)} \in \mathbb{Z}^{V^{(i+1)}}_{++}$, and $w^{(i+1)} \in \mathbb{R}^{A^{(i+1)}}_+$.

- For each $X \in \mathcal{X}$, execute the following updates. First, contract X to obtain a new vertex v_X. Then, for every arc $a = (z, y) \in A^{(i)}$ with $z \in V^{(i)} \setminus X$ and $y \in X$,

$$\begin{aligned} z' &:= \begin{cases} v_{X'} & (z \in X' \text{ for some } X' \in \mathcal{X}), \\ z & (\text{otherwise}), \end{cases} \\ a' &:= (z', v_X), \\ \Psi(a') &:= a, \\ w^{(i+1)}(a') &:= w^{(i)}(a) - w^{(i)}(\alpha(a, F^{(i)})) + w^{(i)}(a_X), \end{aligned}$$

  where $\alpha(a, F^{(i)})$ is an arc in $\delta^-_{F^{(i)}}(y)$ minimizing $w^{(i)}$, and a_X is an arc in $F^{(i)}[X]$ minimizing $w^{(i)}$.[4]
- Define $b^{(i+1)} \in \mathbb{Z}^{V^{(i+1)}}_{++}$ by

$$b^{(i+1)}(v) := \begin{cases} 1 & (v = v_X \text{ for some } X \in \mathcal{X}), \\ b^{(i)}(v) & (\text{otherwise}). \end{cases}$$

Let $i := i + 1$ and go back to Step 2.
Step 5. If $i = 0$, then return F.
Step 6. For every strong component X in $(V^{(i-1)}, F^{(i-1)})$ such that (9) holds, apply the following update:

$$F := \begin{cases} ((F \setminus \{a'\}) \cup \{\Psi(a')\}) \cup (F^{(i-1)}[X] \setminus \{\alpha(\Psi(a'), F^{(i-1)})\}) & (\exists a' = (z, v_X) \in F), \\ F \cup (F^{(i-1)}[X] \setminus \{a_X\}) & (\text{otherwise}). \end{cases}$$

Let $i := i - 1$ and go back to Step 5.

the maximum-weight b-branching problem. It is a special case of the linear program for weighted matroid intersection, and hence we already know that the linear system is endowed with total dual integrality. Here we show an algorithmic proof for the total dual integrality. That is, we show that, when w is an integer vector, integral optimal primal and dual solutions can be computed via ALGORITHM bB.

Consider the following linear program, in variable $x \in \mathbb{R}^A$, associated with the maximum-weight b-branching problem:

$$\text{maximize} \quad \sum_{a \in A} w(a)x(a) \tag{10}$$

$$\text{subject to} \quad x(\delta_A^-(v)) \leq b(v) \quad (v \in V), \tag{11}$$

$$x(A[X]) \leq b(X) - 1 \quad (\emptyset \neq X \subseteq V), \tag{12}$$

$$0 \leq x(a) \leq 1 \quad (a \in A). \tag{13}$$

The constraints (11)–(13) are indeed a special case of a linear system describing the common independent sets in two matroids, which is totally dual integral (see [47]).

▶ **Theorem 7.** *The linear system* (11)–(13) *is totally dual integral. In particular, the linear system* (11)–(13) *determines the b-branching polytope.*

The dual problem of (10)–(13), in variable $p \in \mathbb{R}^{2^V}$ and $q \in \mathbb{R}^A$, is described as follows.

$$\text{minimize} \quad \sum_{v \in V} b(v)p(v) + \sum_{X\colon \emptyset \neq X \subseteq V} (b(X) - 1)p(X) + \sum_{a \in A} q(a) \tag{14}$$

$$\text{subject to} \quad p(v) + \sum_{X\colon a \in A[X]} p(X) + q(a) \geq w(a) \quad (a = uv \in A), \tag{15}$$

$$p(X) \geq 0 \quad (X \subseteq V), \tag{16}$$

$$q(a) \geq 0 \quad (a \in A). \tag{17}$$

Note that the dual variable $p(X)$ corresponds to the primal constraint (11) if $|X| = 1$, and to (12) if $|X| \geq 2$. The primal constraint (12) for X with $|X| = 1$ does not have a corresponding dual variable, since it is redundant in the linear problem (10)–(13).

An optimal solution (p^*, q^*) is computed via Algorithm bB in the following manner. At the beginning of Algorithm bB, set $w^\circ = w$, $p(X) = 0$ for each $X \subseteq V$, and $q(a) = 0$ for each $a \in A$. In Step 4 of Algorithm bB, for each strong component $X \in \mathcal{X}$, define $p^*(X) \in \mathbb{R}$ by

$$p^*(X) = \min\{\min\{w^\circ(\alpha^\circ(a)) - w^\circ(a)\colon a \in \delta^-_{A^{(i)}}(X)\}, \min\{w^\circ(a')\colon a' \in F^{(i)}[X]\}\},$$

where $\alpha^\circ(a)$ is the $b(y)$-th optimal arc with respect to w° among the arcs sharing the terminal vertex $y \in V$ with a in the original digraph D. Then for each arc $a \in A$ such that $a \in A^{(i)}[X]$ or a is deleted in the contraction of X' with $v_{X'}$ included in X, set $w^\circ(a) := w^\circ(a) - p^*(X)$. After the termination of Algorithm bB, let the value $p^*(v)$ be equal to the $b(v)$-th maximum value among $\{w^\circ(a)\colon a \in \delta_A^-(v)\}$ for each vertex $v \in V$. Finally, let $q^*(a) = \max\{w(a) - p^*(v) - \sum_{X\colon a \in A[X]} p^*(X), 0\}$. Observe that $w^\circ(a) \geq 0$ holds for $a \in F$, and $w^\circ(a) \leq 0$ for $a \notin F$.

We can prove the optimality of F and (p^*, q^*) in the following way. We first show that F is a b-branching. It is straightforward to see that $F \in \mathcal{I}_{\text{in}}$. Then, it follows from Lemma 6 that $F \in \mathcal{I}_{\text{sp}}$ as well, since a strong component X satisfying (8) is always contracted in the algorithm and hence never exists in the output F. Next, the feasibility of (p^*, q^*) for (15)–(17) is obvious. Finally, we prove that the characteristic vector χ_F of the output F and (p^*, q^*) satisfy the complementary slackness condition as follows:

- Suppose $\chi_F(\delta_A^-(v)) < b(v)$ for $v \in V$. If v is not contained in a contracted vertex set, then, by Step 2 of Algorithm bB, $\delta_A^-(v)$ contains less than $b(v)$ arcs with positive weight, and hence $p^*(v) = 0$. If v is contained in a contracted vertex set $X \subseteq V^{(i)}$, it follows that v is the terminal vertex of $a_X \in F^{(i)}[X]$ and $p^*(X) = w^\circ(a_X)$. Then $w^\circ(a_X)$ is the $b(v)$-th maximum value among $\{w^\circ(a) \colon a \in \delta_A^-(v)\}$ and becomes zero after contracting X, which implies that $p^*(v) = 0$.
- If $\chi_F(A[X]) < b(X) - 1$ for $X \subseteq V$ with $|X| \geq 2$, it follows that X is not contracted in the algorithm, and thus $p^*(X)$ is never changed from zero.
- If $\chi_F(a) > 0$ for $a \in A$, then it follows that $w^\circ(a) = w(a) - p^*(v) - \sum_{X \colon a \in A[X]} p^*(X) \geq 0$, and thus $q^*(a) = w(a) - p^*(v) - \sum_{X \colon a \in A[X]} p^*(X)$, implying the equality in (15).
- If $\chi_F(a) < 1$ for $a \in A$, then it follows that $w^\circ(a) = w(a) - p^*(v) - \sum_{X \colon a \in A[X]} p^*(X) \leq 0$ and thus $q^*(a) = 0$.

Therefore, F and (p^*, q^*) are optimal solutions for the linear programs (10)–(13) and (14)–(17), respectively. Moreover, (p^*, q^*) is integer if w is integer, which implies that (11)–(13) is totally dual integral.

3.3 Existence of a b-branching with prescribed indegree

Our algorithm leads to the following theorem characterizing the existence of b-branching with prescribed indegree, which is an extension of that for arborescences [4, 11, 23].

▶ **Theorem 8.** *Let $D = (V, A)$ be a digraph and $b \in \mathbb{Z}_{++}^V$ be a positive integer vector on V. Let $b' \in \mathbb{Z}_+^V$ be a nonnegative integer vector such that $b'(v) \leq b(v)$ for every $v \in V$ and $b' \neq b$. Then, D has a b-branching B such that $d_B^-(v) = b'(v)$ for each $v \in V$ if and only if*

$$d_A^-(v) \geq b'(v) \quad (v \in V), \tag{18}$$

$$d_A^-(X) \geq 1 \quad (\emptyset \neq X \subsetneq V,\ b'(X) = b(X) \neq 0). \tag{19}$$

Let $r \in V$ be a specified vertex. A characterization of the existence of an r-arborescence [4, 11, 23] is obtained as a special case of Theorem 8, by putting $b(v) = 1$ for every $v \in V$, $b'(v) = 1$ for every $v \in V \setminus \{r\}$, and $b'(r) = 0$.

Theorem 8 can be proved in two ways. The necessity of (18) and (19) is clear. One way to derive the sufficiency of (18) and (19) is Algorithm bB. Apply Algorithm bB to the case where $b = b'$ and $w(a) = 1$ for each $a \in A$. Then, (18) and (19) certify that $F^{(i)}$ found in Step 2 of Algorithm bB is always a base of $\mathbf{M}_{\mathrm{in}}^{(i)}$. It thus follows that the output F of Algorithm bB is a b-branching with $d_F^- = b'$. An alternative proof for the sufficiency of (18) and (19) is implied by the proof for Theorem 10 in Section 4, which extends Theorem 8 to a characterization of the existence of disjoint b-branchings with prescribed indegree.

4 Packing disjoint b-branchings

In this section, we present a theorem on packing disjoint b-branchings $B_1, \ldots, B_k$ with prescribed indegree, which extends Theorem 1, as well as Theorem 8. We then show that such disjoint b-branchings can be found in strongly polynomial time. We further show that disjoint b-branchings $B_1, \ldots, B_k$ minimizing the weight $w(B_1) + \cdots + w(B_k)$ can be found in strongly polynomial time. Finally, as a consequence of our packing theorem, we prove the integer decomposition property of the b-branching polytope.

4.1 Characterizing theorem for disjoint b-branchings

Let $D = (V, A)$ be a digraph, $b \in \mathbb{Z}^V_{++}$ be a positive integer vector on V, and k be a positive integer. For $i \in [k]$, let $b_i \in \mathbb{Z}^V_+$ be a nonnegative integer vector such that $b_i(v) \leq b(v)$ for every $v \in V$ and $b_i \neq b$. We present a theorem for chracterizing whether D contains disjoint b-branchings $B_1, \ldots, B_k$ such that $d^-_{B_i} = b_i$ for each $i \in [k]$.

We begin with introducing a function which plays a key role in the sequel. Define a function $g : 2^V \to \mathbb{Z}_+$ by

$$g(X) = |\{i \in [k] \colon b_i(X) = b(X) \neq 0\}| \quad (X \subseteq V). \tag{20}$$

The following lemma is straightforward to observe. Its proof is described in the full version [31].

▶ **Lemma 9.** *The function g is supermodular.*

Our characterization theorem is described as follows.

▶ **Theorem 10.** *Let $D = (V, A)$ be a digraph, $b \in \mathbb{Z}^V_{++}$ be a positive integer vector on V, and k be a positive integer. For $i \in [k]$, let $b_i \in \mathbb{Z}^V_+$ be a nonnegative integer vector such that $b_i(v) \leq b(v)$ for every $v \in V$ and $b_i \neq b$. Then, D has disjoint b-branchings $B_1, \ldots, B_k$ such that $d^-_{B_i} = b_i$ for each $i \in [k]$ if and only if the following two conditions are satisfied:*

$$d^-_A(v) \geq \sum_{i=1}^{k} b_i(v) \quad (v \in V), \tag{21}$$

$$d^-_A(X) \geq g(X) \qquad (X \subseteq V). \tag{22}$$

We remark that Szegő's generalization of Theorem 1 for packing arc sets which cover some intersecting families [48] (see also [19, Theorem 10.3.2]) looks similar to Theorem 10, but it does not directly implies Theorem 10. In Theorem 10, every arc set should cover $\delta^-_A(v)$ multiple times (i.e., $\delta^-_A(v)$ should be covered $b_i(v)$ times by B_i), and this coverage is not immediately rephrased in the form of [48].

Below is a proof for Theorem 10, which extends the proof for Theorem 1 by Lovász [38].

Proof of Theorem 10. Necessity is clear. We prove sufficiency by induction on $\sum_{i=1}^{k} b_i(V)$. The case $\sum_{i=1}^{k} b_i(V) = 0$ is trivial; $B_i = \emptyset$ for each $i \in [k]$.

Without loss of generality, suppose $b_1(V) > 0$. Define a partition $\{V_0, V_1, V_2\}$ of V by $V_0 = \{v \in V \colon b_1(v) = 0\}$, $V_1 = \{v \in V \colon 0 < b_1(v) < b(v)\}$, and $V_2 = \{v \in V \colon b_1(v) = b(v)\}$. Then, it holds that

$$V_0 \cup V_1 \neq \emptyset, \tag{23}$$

$$V_0 \neq V, \tag{24}$$

which follow from $b_1 \neq b$ and $b_1(V) > 0$, respectively.

For $X \subseteq V$, define $g(X) = |\{i \in [k] \colon b_i(X) = b(X) \neq 0\}|$. Let $W \subseteq V$ be an inclusionwise minimal vertex subset satisfying

$$W \cap (V_0 \cup V_1) \neq \emptyset, \tag{25}$$

$$W \setminus V_0 \neq \emptyset, \tag{26}$$

$$d^-_A(W) = g(W). \tag{27}$$

Such W always exists, since $W = V$ satisfies (25)–(27): (25) follows from (23); (26) from (24); and (27) from $b_i \neq b$ ($i \in [k]$) and hence $g(V) = 0$. Let $W_j = W \cap V_j$ ($j = 0, 1, 2$).

▶ **Claim 1.** *There exists an arc $(u, v) \in A$ such that $u \in W_0 \cup W_1$ and $v \in W_1 \cup W_2$.*

Proof. First, suppose that $W_2 \neq \emptyset$. Then, it holds that $g(W_2) > g(W)$, since every $i \in [k]$ contributing to $g(W)$ also contributes to $g(W_2)$, and $i = 1$ does not contribute to $g(W)$ but to $g(W_2)$. Hence we obtain that

$$d_A^-(W_2) \geq g(W_2) > g(W) = d_A^-(W). \tag{28}$$

Now (28) implies that there exists an arc $(u, v) \in A$ such that $u \in W_0 \cup W_1$ and $v \in W_2$.

Next, suppose that $W_2 = \emptyset$. By (26), we have that $W_1 \neq \emptyset$. Then, it holds that

$$\begin{aligned}
\sum_{v \in W_1} d_A^-(v) &\geq \sum_{i=1}^{k} b_i(W_1) \quad (\because (21)) \\
&> \sum_{i=2}^{k} b_i(W_1) \quad (\because b_1(W_1) > 0) \\
&\geq |\{i \in [k] \colon b_i(W) = b(W) \neq 0\}| \quad (\because b_1(W) \neq b(W)) \\
&= g(W) = d_A^-(W),
\end{aligned}$$

implying that there exists an arc $(u, v) \in A$ such that $u \in W = W_0 \cup W_1$ and $v \in W_1$. ◀

Let $a = (u, v) \in A$ be an arc in Claim 1. We then show that resetting

$$A := A \setminus \{a\}, \tag{29}$$
$$b_1(v) := b_1(v) - 1 \tag{30}$$

maintains (21) and (22). (This resetting amounts to augmenting B_1 by adding a.)

It is straightforward to see that the resetting (29) and (30) maintain (21). To prove that it also maintain (22), suppose to the contrary that $X \subseteq V$ violates (22) after the resetting.

This violation implies that $d_A^-(X) = g(X)$ before the resetting, and $d_A^-(X)$ has decreased by one while $g(X)$ has remained unchanged by the resetting. It then follows that

$$u \in V \setminus X \quad \text{and} \quad v \in X. \tag{31}$$

It also follows that $i = 1$ does not contribute to $g(X)$, and hence before the resetting, it holds that

$$X \cap (V_0 \cup V_1) \neq \emptyset. \tag{32}$$

By (31), we have that $u \in W \setminus X$ and $v \in X \cap W$, and hence $\emptyset \neq X \cap W \subsetneqq W$. Here we show that $X \cap W$ satisfies (25)–(27), which contradicts the minimality of W.

Before the resetting, it holds that

$$\begin{aligned}
d_A^-(X \cap W) &\leq d_A^-(X) + d_A^-(W) - d_A^-(X \cup W) && (33) \\
&\leq g(X) + g(W) - g(X \cup W) && (34) \\
&\leq g(X \cap W). && (35)
\end{aligned}$$

Indeed, (33) follows from submodularity of d_A^-. The inequality (34) follows from $d_A^-(X) = g(X)$, $d_A^-(W) = g(W)$, and $d_A^-(X \cup W) \geq g(X \cup W)$. Finally, (35) follows from Lemma 9.

Since $d_A^-(X \cap W) \geq g(X \cap W)$ by (22), all inequalities (33)–(35) hold with equality, and hence $d_A^-(X \cap W) = g(X \cap W)$ holds before the resetting.

Equality in (35) implies that $(X \cap W) \cap (V_0 \cup V_1) \neq \emptyset$. Indeed, we have that $W \cap (V_0 \cup V_1) \neq \emptyset$ because $u \in W \cap (V_0 \cup V_1)$, and hence $i = 1$ does not contribute to $g(W)$. Combined with (32), $i = 1$ contributes to none of $g(X)$, $g(W)$, and $g(X \cup W)$. Thus, by the equality in (35), $i = 1$ does not contribute to $g(X \cap W)$ as well, and hence $(X \cap W) \cap (V_0 \cup V_1) \neq \emptyset$ must hold.

We also have $(X \cap W) \setminus V_0 \neq \emptyset$, because $v \in (X \cap W) \setminus V_0$. Therefore, $X \cap W$ satisfies (25)–(27), contradicting the minimality of W. Thus, we have finished proving that resetting of (29) and (30) maintains (22).

Now we can apply induction to obtain disjoint b-branchings $B_1, \ldots, B_k$ in the digraph $(V, A \setminus \{a\})$ such that $d^-_{B_1} = b_1 - \chi_v$ and $d^-_{B_i} = b_i$ for $i = 2, \ldots, k$, where $\chi_v \in \mathbb{Z}^V$ is a vector defined by $\chi_v(v) = 1$ and $\chi_v(u) = 0$ for e very $u \in V \setminus \{v\}$. We complete the proof by showing that $B_1 \cup \{a\}$ is a b-branching.

In resetting, we always have $u \in W_0 \cup W_1$, which implies that the construction of B_1 begins with a vertex r with $b_1(r) < b(r)$ and the component in (V, B_1) containing a includes r. Thus, no $X \subseteq V$ comes to satisfy $|B_1[X]| = b(X)$. ◀

4.2 Algorithm for finding disjoint b-branchings

Let us discuss the algorithmic aspect of Theorem 10. First, we can determine whether (21) and (22) hold in strongly polynomial time. Condition (21) is clear. For (22), we have that $d^-_A(X)$ is submodular and $g(X)$ is supermodular (Lemma 9), and hence $d^-_A(X) - g(X)$ is submodular. Thus, we can determine whether there exists X with $d^-_A(X) - g(X) < 0$ by submodular function minimization, in strongly polynomial time [28, 37, 46].

Finding b-branchings $B_1, \ldots, B_k$ can also be done in strongly polynomial time. By the proof for Theorem 10, it suffices to find an arc $a \in A$ such that resetting

$$A := A \setminus \{a\}, \quad b_1(v) := b_1(v) - 1 \tag{36}$$

maintains (22). This can be done by determining whether there exists X with $d^-_A(X) - g(X) < 0$ after resetting (36) for each $a \in A$, i.e., at most $|A|$ times of submodular function minimization [28, 37, 46].

▶ **Theorem 11.** *Conditions* (21) *and* (22) *can be checked in strongly polynomial time. Moreover, if* (21) *and* (22) *hold, then disjoint b-branchings* $B_1, \ldots, B_k$ *such that* $d^-_{B_i} = b_i$ *for each* $i \in [k]$ *can be found in strongly polynomial time.*

Further, if a weight vector $w \in \mathbb{R}^A_+$ is given, we can find disjoint b-branchings $B_1, \ldots, B_k$ minimizing $w(B_1) + \cdots + w(B_k)$ in strongly polynomial time. Indeed, conditions (21) and (22) derive a totally dual integral system which determines a *submodular flow polyhedron*. A set family $\mathcal{C} \subseteq 2^V$ is called a *crossing family* if, for each $X, Y \in \mathcal{C}$ with $X \cup Y \neq V$ and $X \cap Y \neq \emptyset$, it holds that $X \cup Y, X \cap Y \in \mathcal{C}$. A function $f : \mathcal{C} \to \mathbb{R}$ defined on a crossing family $\mathcal{C} \subseteq V$ is called *crossing submodular* if, for each $X, Y \in \mathcal{C}$ with $X \cup Y \neq V$ and $X \cap Y \neq \emptyset$, it holds that $f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$. A function f is *crossing supermodular* if $-f$ is crossing submodular. A *submodular flow polyhedron* is a polyhedron described as

$$\begin{aligned} &x(\delta^-_A(X)) - x(\delta^+_A(X)) \leq f(X) && (X \in \mathcal{C}), \\ &l(a) \leq x(a) \leq u(a) && (a \in A) \end{aligned}$$

by some digraph (V, A), crossing submodular function f on a crossing family $\mathcal{C} \subseteq 2^V$, and vectors $l, u \in \mathbb{R}^A$, where $\delta^+_A(X)$ denotes the set of arcs in A from X to $V \setminus X$.

▶ **Lemma 12** ([45]). *For a digraph $D = (V, A)$, let $f\colon 2^V \to \mathbb{R}$ be a crossing supermodular function on $\mathcal{C} \subseteq 2^V$ and $u \in \mathbb{R}^A$. Then, a polyhedron determined by*

$$x(\delta_A^-(X)) \geq f(X) \quad (X \in \mathcal{C}),$$
$$0 \leq x(a) \leq u(a) \quad (a \in A)$$

is a submodular flow polyhedron.

By Lemma 12, the linear inequality system (21) and (22) determines a submodular flow polyhedron. Indeed, we can define a crossing supermodular function $f\colon 2^V \to \mathbb{R}$ by

$$f(X) = \begin{cases} \sum_{i=1}^{k} b_i(v) & (X = \{v\} \text{ for some } v \in V), \\ g(X) & (\text{otherwise}). \end{cases}$$

Since a submodular flow polyherdron is totally dual integral [15], an arc set $B \subseteq A$ with (21) and (22) minimizing $w(B)$ can be found by optimization over a submodular flow polyhedron, which can be done in strongly polynomial time [16, 21, 29, 30]. After that, we can partition B into b-branchings $B_1, \ldots, B_k$ with $d^-_{B_i} = b_i$ $(i \in [k])$ in the same manner as above.

▶ **Theorem 13.** *If (21) and (22) hold, disjoint b-branchings $B_1, \ldots, B_k$ such that $d^-_{B_i} = b_i$ for each $i \in [k]$ minimizing $w(B_1) + \cdots + w(B_k)$ can be found in strongly polynomial time.*

4.3 Integer decomposition property of the b-branching polytope

In this subsection we show another consequence of Theorem 10: the integer decomposition property of the b-branching polytope. First, Theorem 10 leads to the following min-max relation on covering by b-branchings. This is an extension of Theorem 2, the theorem on covering by branchings [17, 41]. Its proof appears in the full version [31].

▶ **Corollary 14.** *Let $D = (V, A)$ be a digraph, $b \in \mathbb{Z}^V_{++}$ be a positive integer vector on V, and k be a positive integer. Then, the arc set A can be covered by k b-branchings if and only if*

$$d_A^-(v) \leq k \cdot b(v) \quad (v \in V), \tag{37}$$
$$|A[X]| \leq k(b(X) - 1) \quad (\emptyset \neq X \subseteq V). \tag{38}$$

The integer decomposition property of the b-branching polytope is a direct consequence of Corollary 14. Its proof is described in the full version [31].

▶ **Corollary 15.** *The b-branching polytope has the integer decomposition property.*

5 Matroid-restricted b-branchings

Our multi-phase greedy algorithm (Algorithm bB) can be extended to a more generalized problem of finding a maximum-weight *matroid-restricted b-branching*. Let $D = (V, A)$ be a digraph and $b \in \mathbb{Z}^V_{++}$ be a positive integer vector on V. For each vertex $v \in V$, a matroid $\mathbf{M}_v = (\delta^-(v), \mathcal{I}_v)$ with rank $b(v)$ is attached. We denote the direct sum of $\mathbf{M}_v$ for every $v \in V$ by $\mathbf{M}_V = (A, \mathcal{I}_V)$. Now an arc set $F \subseteq A$ is an *$\mathbf{M}_V$-restricted b-branching* if $F \in \mathcal{I}_V \cap \mathcal{I}_{\mathrm{sp}}$. Note that a b-branching is a special case where $\mathbf{M}_v$ is a uniform matroid for each $v \in V$.

A maximum-weight $\mathbf{M}_V$-restricted b-branching can be found by a slight extension of Algorithm bB. The extended algorithm is described in the full version [31].

6 Concluding remarks

In this paper, we have proposed b-branchings, a generalization of branchings. In a b-branching, a vertex v can have indegree at most $b(v)$, and thus b-branchings serve as a counterpart of b-matchings for matchings.

It is somewhat surprising that, to the best of our knowledge, such a fundamental generalization of branchings has never appeared in the literature. The reason might be that, in order to obtain a reasonable generalization, it is far from being trivial how the other matroid (graphic matroid) in branchings is generalized. We have succeeded in obtaining a generalization inheriting the multi-phase greedy algorithm [4, 6, 11, 23] and the packing theorem [13] for branchings by setting a sparsity matroid defined by (2) as the other matroid.

An important property of the two matroids is Lemma 6, which says that an independent set of one matroid is decomposed into an independent set and some circuits in the other matroid. This plays an important role in the design of a multi-phase greedy algorithm: find an optimal independent F set in one matroid; contract the circuits in F with respect to the other matroid; and the optimal common independent set can be found recursively. We remark that the definitions (1) and (2) are essential to attain this property. For example, the property fails if the vector b is not identical in (1) and (2). It also fails if the sparsity matroid is defined by $|F[X]| \leq b(X) - k$ for $k \neq 1$.

Another remark is on the similarity of our algorithm and the blossom algorithm for non-bipartite matchings [10], where a factor-critical component can be contracted and expanded. In our b-branching algorithm, for each strong component $X \in \mathcal{X}$ and each $v^* \in X$, there exists an arc set $F_X \subseteq A[X]$ such that $d^-_{F_X}(v^*) = b(v^*) - 1$ and $d^-_{F_X}(v) = b(v)$ for each $v \in X \setminus \{v^*\}$. In the blossom algorithm for nonbipatite matchings, for each factor-critical component X and each vertex $v^* \in X$, there exists a matching exactly covering $X \setminus \{v^*\}$.

We finally remark that the problem of finding a maximum-weight b-branching is a special case of a modest generalization of the framework of the $\mathcal{U}$-feasible t-matching problem in bipartite graphs [49]. In [49], it is proved that the $\mathcal{U}$-feasible t-matching problem in bipartite graphs is efficiently tractable under certain assumptions on the family of excluded structures $\mathcal{U}$. The b-branching problem can be regarded as a new problem which falls in this tractable class of the (generalized) $\mathcal{U}$-feasible t-matching problem.

References

1 S. Baum and L.E. Trotter, Jr.: Integer rounding for polymatroid and branching optimization problems, *SIAM Journal on Algebraic and Discrete Methods*, 2 (1981), 416–425.

2 Y. Benchetrit: Integer round-up property for the chromatic number of some h-perfect graphs, *Mathematical Programming*, 164 (2017), 245–262.

3 A. Bernáth and T. Király: Blocking optimal k-arborescences, in *Proc. 27th SODA*, 2016, 1682–1694.

4 F. Bock: An algorithm to construct a minimum directed spanning tree in a directed network, in *Developments in Operations Research*, Gordon and Breach, 1971, 29–44.

5 C. Brezovec, G. Cornuéjols and F. Glover: Two algorithms for weighted matroid intersection, *Mathematical Programming*, 36 (1986), 39–53.

6 Y.J. Chu and T.H. Liu: On the shortest arborescence of a directed graph, *Scientia Sinica*, 14 (1965), 1396–1400.

7 J. Davies and C. McDiarmid: Disjoint common transversals and exchange structures, *The Journal of the London Mathematical Society*, 14 (1976), 55–62.

8 R. Dougherty, C. Freiling and K. Zeger: Network coding and matroid theory, *Proceedings of the IEEE*, 99 (2011), 388–405.

9 O. Durand de Gevigney, V.-H. Nguyen and Z. Szigeti: Matroid-based packing of arborescences, *SIAM Journal on Discrete Mathematics*, 27 (2013), 567–574.

10 J. Edmonds: Paths, trees, and flowers, *Canadian Journal of Mathematics*, 17 (1965), 449–467.

11 J. Edmonds: Optimum branchings, *Journal of Research National Bureau of Standards, Section B*, 71 (1967), 233–240.

12 J. Edmonds: Submodular functions, matroids, and certain polyhedra, *Combinatorial Structures and Their Applications*, New York, 1970, Gordon and Breach, 69–87.

13 J. Edmonds: Edge-disjoint branchings, in R. Rustin, ed., *Combinatorial Algorithms*, Algorithmics Press, 1973, 285–301.

14 J. Edmonds: Matroid intersection, *Annals of Discrete Mathematics*, 4 (1979), 39–49.

15 J. Edmonds and R. Giles: A min-max relation for submodular functions on graphs, *Annals of Discrete Mathematics*, 1 (1977), 185–204.

16 L. Fleischer, S. Iwata and S.T. McCormick: A faster capacity scaling algorithm for minimum cost submodular flow, *Mathematical Programming*, 92 (2002), 119–139.

17 A. Frank: Covering branchings, *Acta Scientiarum Mathematicarum*, 41 (1979), 77–81.

18 A. Frank: A weighted matroid intersection algorithm, *Journal of Algorithms*, 2 (1981), 328–336.

19 A. Frank: Rooted k-connections in digraphs, *Discrete Applied Mathematics*, 157 (2009), 1242–1254.

20 A. Frank: *Connections in Combinatorial Optimization*, Oxford University Press, New York, 2011.

21 A. Frank and É. Tardos: An application of simultaneous Diophantine approximation in combinatorial optimization, *Combinatorica*, 7 (1987), 49–65.

22 S. Fujishige: A note on disjoint arborescences, *Combinatorica*, 30 (2010), 247–252.

23 D.R. Fulkerson: Packing rooted directed cuts in a weighted directed graph, *Mathematical Programming*, 6 (1974), 1–13.

24 F.R. Giles: *Submodular Functions, Graphs and Integer Polyhedra*, Ph.D. thesis, University of Waterloo, 1975.

25 N.J.A. Harvey, D.R. Karger and K. Murota: Deterministic network coding by matrix completion, in *Proc. 16th SODA*, 2005, 489–498.

26 C.-C. Huang, N. Kakimura and N. Kamiyama: Exact and approximation algorithms for weighted matroid intersection, *Mathematical Programming*, to appear.

27 M. Iri and N. Tomizawa: An algorithm for finding an optimal "independent assignment", *Journal of the Operations Research Society of Japan*, 19 (1976), 32–57.

28 S. Iwata, L. Fleischer and S. Fujishige: A combinatorial strongly polynomial algorithm for minimizing submodular functions, *Journal of the ACM*, 48 (2001), 761–777.

29 S. Iwata, S.T. McCormick and M. Shigeno: A fast cost scaling algorithm for submodular flow, *Information Processing Letters*, 74 (2000), 123–128.

30 S. Iwata, S.T. McCormick and M. Shigeno: Fast cycle canceling algorithms for minimum cost submodular flow, *Combinatorica*, 23 (2003), 503–525.

31 N. Kakimura, N. Kamiyama, and K. Takazawa: The b-branching problem in digraphs, *CoRR*, abs/1802.02381 (2018).

32 N. Kamiyama: Arborescence problems in directed graphs: Theorems and algorithms, *Interdisciplinary Information Sciences*, 20 (2014), 51–70.

33 N. Kamiyama, N. Katoh, and A. Takizawa: Arc-disjoint in-trees in directed graphs, *Combinatorica*, 29 (2009), 197–214.

34 C. Király: On maximal independent arborescence packing, *SIAM Journal on Discrete Mathematics*, 30 (2016), 2107–2114.

35 J. Kleinberg and É. Tardos: *Algorithm Design*, Addison Wesley, Boston, 2005.

36 B. Korte and J. Vygen: *Combinatorial Optimization – Theory and Algorithms*, Springer, Berlin, 5th ed., 2012.

37 Y. Lee, A. Sidford, and S.C.-W. Wong: A faster cutting plane method and its implications for combinatorial and convex optimization, in *Proc. 56th FOCS*, IEEE Computer Society, 2015, 1049–1065.

38 L. Lovász: On two minimax theorems in graph, *Journal of Combinatorial Theory, Series B*, 21 (1976), 96–103.

39 E.L. Lawler: Optimal matroid intersections, *Combinatorial Structures and Their Applications*, New York, 1970, Gordon and Breach, 233–234.

40 E.L. Lawler: Matroid intersection algorithms, *Mathematical Programming*, 9 (1975), 31–56.

41 S.E. Markosyan and G.S. Gasparyan: Optimal'noe razlozhenie orientirovannykh mul'tigrafov na orlesa [Russian; The optimal decomposition of directed multigraphs into a diforest], *Metody Diskretnogo Analiza v Teorii Grafov i Logicheskikh Funktsĭ*, 43 (1986), 75–86.

42 C. McDiarmid: On pairs of strongly-base-orderable matroids, Technical report, No. 283, School of Operations Research and Industrial Engineering, College of Engineering, Cornell University, 1976.

43 K. Murota: *Matrices and Matroids for Systems Analysis*, Springer, Berlin, 2nd ed., 2000.

44 A. Recski: *Matroid Theory and Its Applications in Electric Network Theory and in Statics*, Springer, Berlin, 1989.

45 A. Schrijver: Totally dual integral system from directed graphs, crossing families and sub- and supermodular functions, in W.R. Pulleyblank, ed., *Progress in Combinatorial Optmization*, Toronto, 1984, Academic Press, 315–361.

46 A. Schrijver: A combinatorial algorithm minimizing submodular functions in strongly polynomial time, *Journal of Combinatorial Theory, Series B*, 80 (2000), 346–355.

47 A. Schrijver: *Combinatorial Optimization – Polyhedra and Efficiency*, Springer, Heidelberg, 2003.

48 L. Szegő: On covering intersecting set-systems by digraphs, *Discrete Mathematics*, 234 (2001), 187–189.

49 K. Takazawa: Excluded t-factors: A unified framework for nonbipartite matchings and restricted 2-matchings, in *Proc. 19th IPCO, LNCS 10328*, Springer, 2017, 430–441.

Pairing heaps: the forward variant

Dani Dorfman
Blavatnik School of Computer Science, Tel Aviv University, Israel
dannatand@mail.tau.ac.il

Haim Kaplan[1]
Blavatnik School of Computer Science, Tel Aviv University, Israel
haimk@post.tau.ac.il

László Kozma[2]
Eindhoven University of Technology, The Netherlands
l.kozma@tue.nl

Uri Zwick[3]
Blavatnik School of Computer Science, Tel Aviv University, Israel
zwick@tau.ac.il

Abstract

The pairing heap is a classical heap data structure introduced in 1986 by Fredman, Sedgewick, Sleator, and Tarjan. It is remarkable both for its simplicity and for its excellent performance in practice. The "magic" of pairing heaps lies in the restructuring that happens after the deletion of the smallest item. The resulting collection of trees is consolidated in two rounds: a left-to-right pairing round, followed by a right-to-left accumulation round. Fredman et al. showed, via an elegant correspondence to splay trees, that in a pairing heap of size n all heap operations take $O(\log n)$ amortized time. They also proposed an arguably more natural variant, where both pairing and accumulation are performed in a combined left-to-right round (called the forward variant of pairing heaps). The analogy to splaying breaks down in this case, and the analysis of the forward variant was left open.

In this paper we show that inserting an item and deleting the minimum in a forward-variant pairing heap both take amortized time $O(\log n \cdot 4^{\sqrt{\log n}})$. This is the first improvement over the $O(\sqrt{n})$ bound showed by Fredman et al. three decades ago. Our analysis relies on a new potential function that tracks parent-child rank-differences in the heap.

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

Keywords and phrases data structure, priority queue, pairing heap

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.13

1 Introduction

A heap is an abstract data structure that stores a set of keys, supporting the usual operations of creating an empty heap, inserting a key, finding and deleting the smallest key, decreasing a key, and merging ("melding") two heaps. Heaps are ubiquitous in computer science, used,

[1] Research supported by The Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11), Israel Science Foundation grant no. 1841-14.
[2] Work done while at Tel Aviv University. Research supported by The Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11).
[3] Research supported by BSF grant no. 2012338 and by The Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11).

for instance, in Dijkstra's shortest path algorithm and in the Jarník-Prim minimum spanning tree algorithm (see e.g., [5]). Heaps are typically implemented as trees (binary or multiary), where each node stores a key, and no key may be smaller than its parent-key.

The pairing heap, introduced by Fredman, Sedgewick, Sleator, and Tarjan [9], is a popular heap implementation that supports all operations in $O(\log n)$ amortized time. It is intended to be a simpler, *self-adjusting* alternative to the Fibonacci heap [10], and has been observed to have excellent performance in practice [27, 23, 20, 22]. Despite significant research, the exact complexity of pairing heap operations is still not fully understood [9, 8, 14, 24, 6, 17, 18].

Self-adjusting data structures. Many of the fundamental heap- and tree- data structures in the literature are designed with a certain "good structure" in mind, which guarantees their efficiency. Binary search trees, for instance, need to be (more or less) balanced in order to support operations in $O(\log n)$ time. In the case of heaps, a reasonable goal is to keep node-degrees low, as it is costly to delete a node with many children. Maintaining a tree structure in such a favorable state at all times requires considerable book-keeping. Indeed, data structures of this flavor (AVL-trees [1], red-black trees [2, 11], Binomial heaps [29], Fibonacci heaps [10], etc.) store structural parameters in the nodes of the tree, and enforce strict rules on how the tree may evolve.

By contrast, pairing heaps and other *self-adjusting* data structures store no additional information besides the keys and the pointers that represent the tree itself. Instead, they perform local re-adjustments after each operation, seemingly ignoring global structure. Due to their simplicity and low memory footprint, self-adjusting data structures are appealing in practice. Moreover, self-adjustment allows data structures to *adapt* to various usage patterns. This has led, in the case of search trees, to a rich theory of instance-specific bounds (see e.g., [26, 15]). The price to pay for the simplicity and power of self-adjusting data structures is the complexity of their analysis. With no rigid structure, self-adjusting data structures are typically analysed via potential functions that measure, informally speaking, how far the data structure is from an ideal state.

The standard analysis of pairing heaps [9] borrows the potential function developed for splay trees by Sleator and Tarjan [26]. This technique is not directly applicable to other variants of pairing heaps. There is a large design space of self-adjusting heaps similar to pairing heaps, yet, we currently lack the tools to analyse their behavior (apart from isolated cases). We find it a worthy task to develop tools to remedy this situation.

Description of pairing heaps. A pairing heap is built as a single tree with nodes of arbitrary degree. Each node is identified with a key, with the usual min-heap condition: every (non-root) key is larger than its parent key.

The heap operations are implemented using the unit-cost *linking* primitive: given two nodes x and y (with their respective subtrees), `link`(x, y) compares x and y and lets the greater of the two become the leftmost child of the smaller.

The `delete-min` operation works as follows. We first delete (and return) the root r of the heap (i.e., the minimum), whose children are $x_1, \ldots, x_k$. We then perform a left-to-right pairing round, calling `link`(x_{2i-1}, x_{2i}) for all $i = 1, \ldots, \lfloor k/2 \rfloor$. The resulting roots are denoted $y_1, \ldots, y_{\lceil k/2 \rceil}$. (Observe that if k is odd, the last root is not linked.) Finally, we perform a right-to-left accumulate round, calling `link`(p_i, y_{i-1}) for all $i = \lceil k/2 \rceil, \ldots, 2$, where p_i is the minimum among $y_i, \ldots, y_{\lceil k/2 \rceil}$. We illustrate this process in Figure 1.

The other operations are straightforward. In `decrease-key`, we cut out the node whose key is decreased (together with its subtree) and link it with the root. In `insert`, we link the new node with the root. In `meld`, we link the two roots.

Fredman et al. [9] showed that the amortized time of all operations is $O(\log n)$. They also conjectured that, similarly to Fibonacci heaps, the amortized time of `decrease-key` is in fact constant. This conjecture was disproved by Fredman [8], who showed that in certain sequences, `decrease-key` requires $\Omega(\log \log n)$ time. In fact, the result of Fredman holds for a more general family of heap algorithms. Iacono and Özkan [17, 16] gave a similar lower bound for a different generalization[4] of pairing heaps. Assuming $O(\log n)$ time for `decrease-key`, Iacono [14] showed that `insert` takes constant amortized time. Pettie [24] proved that both `decrease-key` and `insert` take $O(4^{\sqrt{\log \log n}})$ time. Improving these trade-offs remains a challenging open problem.

Pairing heap variants. We refer to the pairing heap data structure described above as the *standard* pairing heap. Fredman et al. [9] also proposed a number of variants that differ in the way the heap is consolidated during a `delete-min` operation.

We describe first the *forward variant*, which is the main subject of this paper (in the original pairing heap paper this is called the *front-to-back variant*). The pairing round in the forward variant is identical to the standard variant, resulting in roots $y_1, \ldots, y_t$, where $t = \lceil k/2 \rceil$, and k is the number of children of the deleted root. In the forward variant, however, we perform the second round also from left to right: for all $i = 1, \ldots, t-1$, we call `link`(p_i, y_{i+1}), where p_i is the minimum among $y_1, \ldots, y_i$. Occasionally we refer to p_i as the current *left-to-right minimum*. See Figure 1 for an illustration. What may seem like a minor change causes the data structure to behave differently. Currently, the forward variant appears to be much more difficult to analyse than the standard variant.

The implementation of the forward variant is arguably simpler. The two passes can be performed together in a single pass, using only the standard *leftmost child* and *right sibling* pointers. In fact, it is possible to implement the two rounds in one pass also in the standard variant. To achieve this, we need to perform both the pairing and accumulation rounds *from right to left*. As shown by Fredman et al. [9], this can be done with a slightly more complicated link structure, and the original analysis goes through essentially unchanged. Regardless of the low-level details, it remains the case that of the two most natural pairing heap variants one is significantly better understood than the other.

Yet another variant described by Fredman et al. [8] is the *multipass* pairing heap. Here, instead of an accumulation round, repeated pairing rounds are executed, until a single root remains. For multipass pairing heaps, the bound of $O\big(\log n \cdot \log \log n / \log \log \log n\big)$ was shown [9] on the cost of a `delete-min`.[5]

Fredman et al. [9] also describe what we could call the *arbitrary pairing and linking* variant. Here, before performing the initial pairing round, the roots may be arbitrarily reordered. After the pairing round, arbitrary pairs of roots (not necessarily neighbors) are linked, until there is a single root left. For this general case (that subsumes all previous variants) Fredman et al. [9] show a tight[6] $\Theta(\sqrt{n})$ amortized bound for `delete-min`. Even in the special case of the forward variant, the $O(\sqrt{n})$ upper bound has never been improved.

Our main result is the following.

[4] Both results are rather subtle, for instance, it is not clear whether `decrease-key` remains costly in an implementation where we cut an affected node only if its decreased key is smaller than that of its parent.

[5] A recently claimed improvement by Pettie [25] would reduce this to an almost, but not quite, logarithmic bound. The result of Pettie has, in some sense, inspired our results.

[6] It is not clear how efficient this strategy is if we only allow `link` operations between neighboring siblings.

▶ **Theorem 1.** *In the forward variant of pairing heaps, the amortized costs of* `delete-min` *and* `insert` *are* $O(\log n \cdot 4^{\sqrt{\log n}})$*, where* n *is the number of items in the heap when the operation is performed.*

The result holds for sequences of `insert` and `delete-min` operations arbitrarily intermixed, starting from an empty heap. Note that an `insert` operation performs a single `link`, its worst-case cost is therefore constant.

The quantity in the running time is asymptotically smaller than n^ε, for all $\varepsilon > 0$. We remark that the $4^{\sqrt{\log n}}$ term grows much faster than the $\log n$ term, as for arbitrary constants $c, d > 1$ it holds that $c^{\sqrt{\log n}} = \omega(\log^d n)$.

Besides the concrete result (heaps with proven logarithmic cost are known, afterall), the contribution of our paper is in the development of a new, fairly general potential function, that may have further applications.

There has been significant further work in designing heap data structures, with the goal of finding a simpler alternative to Fibonacci heaps, matching, or almost matching their theoretical guarantees (e.g., [27, 19, 7, 12, 13, 4]). These heaps are typically more complicated than pairing heaps. Moreover, they are not self-adjusting, i.e., they store extra information at the nodes, are thus out of the scope of this paper.

Splaying and sorting. There is a standard representation of multi-way rooted trees as binary trees ([21, § 2.3.2], [9]), by renaming the *leftmost child* and *right sibling* pointers as *left child* and *right child*. (See Figure 1.) In this binary tree representation, the restructuring performed when we delete the minimum from a pairing heap closely resembles the restructuring performed by an access to an item in a splay tree. The correspondence is sufficiently close that the proof of logarithmic access cost in splay trees [26] also goes through for the pairing heap. (We note that this correspondence is far from trivial: one has to relabel some nodes and swap their left and right children in the analysis to make it work; we refer to the original pairing heap paper [9] for details.)

In case of the forward variant, the analogy with splay trees breaks down, with no apparent way to fix it. The reason is that the accumulation round of the forward variant may reverse the ordering of large groups of consecutive neighbors. In splay-view, this has the effect of turning long portions of the search path upside-down, something known to be notoriously hard to analyse, see [28, 3].

There is a close link between heaps and sorting. The ancestor-descendant relationship in a heap captures everything we know about the order of the keys at a certain time. A sequence of `delete-min`s in a pairing heap (or in any other heap, for that matter), can be seen as transitioning from the current partial order towards the total order, i.e., sorting. (More precisely, we have here a special kind of selection-sort, in which only candidate minima are ever compared.) It would then be natural for the potential function to capture some measure of "sortedness" of the heap, for example, the entropy of the partial order or some other quantity related to the number of linear extensions. In general, as observed by Pettie [24], this is far from being the case for the splay potential.

The potential function in the classical analysis of splay trees is equal to the sum of the logarithms of subtree-sizes of all nodes in the tree. A subtree in the splay-view corresponds, in the heap-view, to the set of subtrees of a node and all of its right siblings (Figure 1). It is far from intuitive why such a quantity would be useful in analysing pairing heaps. In particular, this potential function does not distinguish between left and right children in the splay-view, even though, in the heap-view, they play different roles: one is a provably larger key, and the other is (so far) incomparable. It may seem that, implicitly, the splay

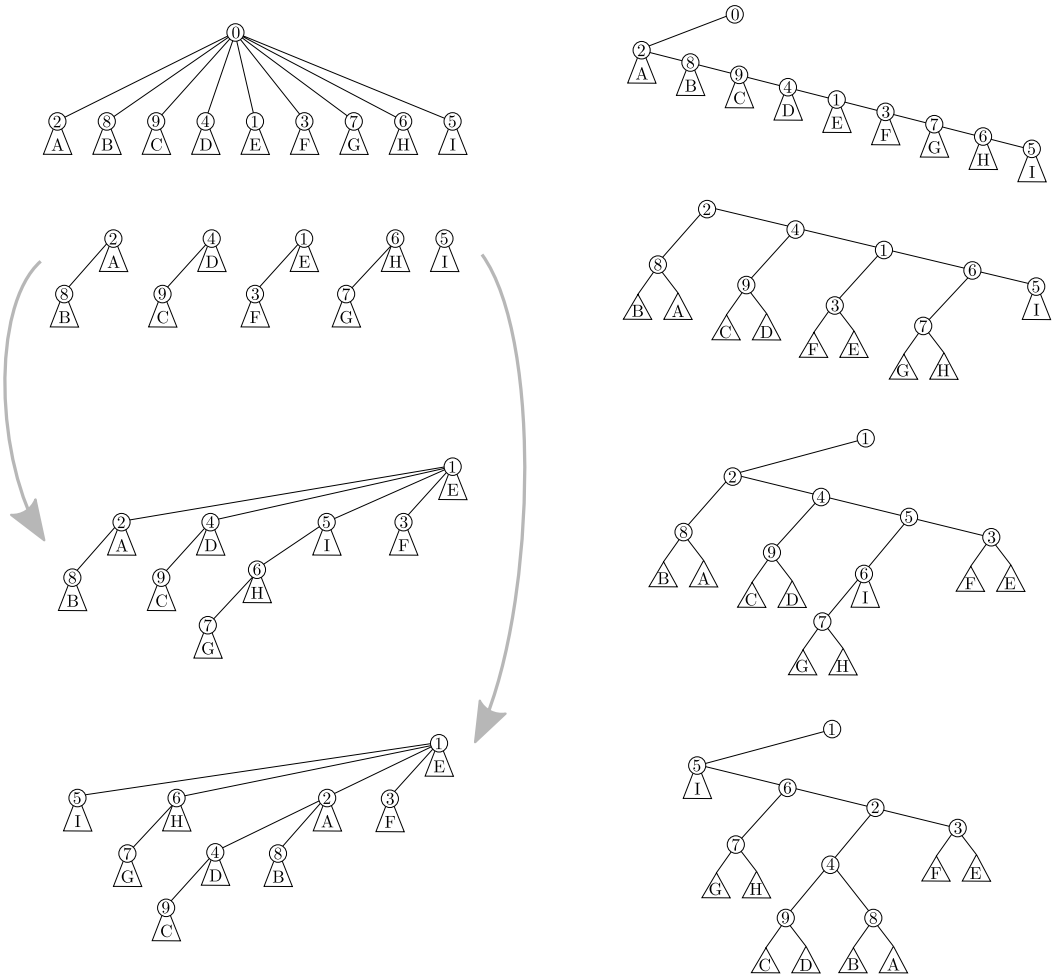


Figure 1 Pairing heap `delete-min` operation. Left: heap-view, right: binary tree-view (splay-view). From top to bottom: (i) initial heap, (ii) deleting the root and a left-to-right pairing round, (iii) standard variant, resulting heap after right-to-left accumulation round, (iv) forward-variant, resulting heap after left-to-right accumulation round. Circled numbers are keys, letters indicate arbitrary subtrees. Observe that the forward variant (iv) reverses the ordering of the siblings with keys 2,6,5.

potential assumes that the left-to-right ordering of siblings (in heap-view) is, to some extent, correlated with the sorted order. We argue that such an assumption is reasonable for the standard variant of pairing heaps, but not for the forward variant (since the accumulation round of the forward variant frequently reverses blocks of consecutive siblings).

As a simple example, consider either the standard or the forward variant, and look at a group of consecutive siblings $x_1, x_2, \ldots, x_k, y$ (arbitrarily shuffled), of which y is the smallest. Suppose, for simplicity, that these nodes interact with each other *only in pairing rounds*, i.e., the minimum of the accumulation round always comes from "outside the group", and does not change within the group. Assume also that, in each pairing round following a deletion of the parent of the group, y gains one node from the group as its new leftmost child.

If x_i and x_j both become children of y, and x_i is to the left of x_j, then x_i became the child of y in a later round than x_j, having "survived" more rounds (winning the links it took part in) at the same level as y. This fact is an indication that x_i may be smaller than x_j.

When y is eventually deleted, the standard variant will preserve the order of its children, whereas the forward variant will reverse it (assuming again, that there is no minimum-change within the group during the accumulation round).

There are limits to this intuition, and it is easy to construct examples that break it. Nonetheless, this interpretation suggests that we analyse the forward variant by somehow directly using the order-information, instead of trying to infer it from the tree structure. This motivates our potential function in § 2.

The intuition described earlier also hints at why the standard variant of pairing heaps may in fact be faster than the forward variant. If the left-to-right ordering of siblings is indeed the increasing order, then a right-to-left accumulate round is vastly more efficient then a left-to-right round. (The former immediately achieves the sorted order, whereas the latter merely finds the minimum.) On the other hand, in this case, the left-to-right round also reverses the order of siblings, making it optimal for the subsequent round. This intuition is consistent with our experiments that show the forward-variant to be somewhat slower[7] than the standard variant. However, the possibility that the forward-variant also has logarithmic cost has not been ruled out.

It remains an interesting open question to determine the exact complexity of the forward variant, and to characterize the types of instances on which it may outperform the standard variant. We also leave open the question whether `decrease-key` may take $o(\log n)$ time in this variant, noting that the lower bound of Fredman [8] applies to this case, whereas the upper bound of Pettie [24] does not.

More importantly, one can hope that new techniques for the analysis of pairing heaps will find their way to the analysis of splay trees and other dynamic search tree algorithms. Splaying and its variants pose some of the most intriguing and central open questions of the field, such as the dynamic optimality conjecture [26, 15].

2 Analysis of the forward variant of pairing heaps

Before proving our main result, as a warm-up, we look at the *arbitrary pairing and linking* variant of pairing heaps. First we introduce some terminology.

We define the *rank* of a node x as the number of nodes in the heap with a smaller key,[8] e.g., the rank of the root is 0. (For simplicity, we assume that the keys are unique.) We denote the rank of x by $r(x)$. The *rank-difference* $rd(x)$ of a node x is defined as $rd(x) = r(x) - r(p(x))$, where $p(x)$ is the parent node of x. It is clear that $rd(x)$ is positive for all non-root nodes x. For the root r, we define $rd(r) = 0$.

2.1 Arbitrary pairing and linking

We show that starting with an arbitrary initial heap of size n, the cost of n `delete-min`s using arbitrary pairing and linking is $O(n\sqrt{n})$. This was already known, as a corollary of Fredman et al.'s result in the original pairing heap paper [9]. Our proof is different (and arguably simpler), and is intended to illustrate the use of rank-differences in the analysis.

Consider a heap of size n. Let Φ denote the sum of rank-differences over all nodes of the heap. Observe that the ranks take all values $0, \ldots, n-1$, and the rank-difference of a node is not more than its rank. It follows that $0 \leq \Phi \leq n(n-1)/2$.

[7] Experiments also suggest that multipass is somewhat slower than the standard variant [27].
[8] Contrary to many other *rank*s in the data structures literature, we use the word in its "original" meaning.

For the purpose of the analysis, we slightly change the implementation of `delete-min`. Rather than deleting the root first, we first do the pairing and accumulation rounds on the children of the root, until the root has only one child. Only then, we actually delete the root. This modified algorithm is equivalent to the original.

In this implementation, all `link` operations take place between children of the root. Consider a `link` between nodes x and y, whose rank-differences are a and b, respectively. Suppose $x < y$, and consequently $a < b$. After the operation, y becomes the leftmost child of x. The new rank-difference of y is $b - a$. Observe that the rank-differences of nodes other than y remain unchanged, therefore the potential Φ decreases by a.

▶ **Theorem 2.** *The cost of n* `delete-min` *operations from an arbitrary initial heap of size n, using arbitrary pairing and linking, is $O(n\sqrt{n})$.*

Proof. Consider a `delete-min` in which the root has k children. In the pairing round we perform $\lfloor k/2 \rfloor$ `link` operations. By the earlier observation, the potential Φ goes down by $a_1 + \cdots + a_{\lfloor k/2 \rfloor}$, where a_i is the rank-difference of the "winner" (i.e., smaller node) in the i-th `link`. The values a_i are distinct (since all respective nodes have different ranks, and the same parent), and $a_i > 0$ for all i. Thus, the entire pairing round reduces Φ by at least $1 + 2 + \cdots + \lfloor k/2 \rfloor \geq (k^2 - 1)/8 \geq (k-1)^2/8$. The remaining operations can only further decrease the potential.

Let k_i be the number of children of the root before the i-th `delete-min` operation. (This is also our estimate for the cost of the operation.) The total reduction in potential due to this operation is at least $\sum_{i=1}^{n} (k_i - 1)^2/8$. On the other hand, the total reduction in potential over all n operations is not more than $n^2/2$. It follows that $\sum_i (k_i - 1)^2 \leq 4n^2$. Under this condition, setting $k_1 = \cdots = k_n = 2\sqrt{n} + 1$ maximizes the total real cost $\sum_i k_i$. ◀

In its current form, the potential function is not well suited for analysing operations other than `delete-min`; a single `insert`, for example, may increase Φ by a linear term. We sketch a new analysis for sequences of `insert` and `delete-min` operations that foretells the technique used in § 2.2.

Define the potential of a node x, denoted by $\phi(x)$, to be $rd(x)$, if $rd(x) \leq \sqrt{n} - 1$, and $\sqrt{n} - 1 + rd(x)/\sqrt{n}$ otherwise. Call nodes of the first kind *light*, and nodes of the second kind *heavy*. The potential function Φ is now defined as the sum of $\phi(x)$ over all nodes x. Observe that for a heap of size n, the total potential is $\Phi < 2n\sqrt{n}$.

Consider the pairing round in a `delete-min` operation. Since there are less than $\sqrt{n}$ light nodes among the children of the root, all but $\sqrt{n}$ of the link operations are between heavy nodes. It is easy to verify that a `link` between two heavy nodes reduces the potential by at least 1. The amortized $O(\sqrt{n})$ cost of `delete-min` follows.

Next we bound the amortized cost of an `insert` operation, considering the increase in potential that it may cause.

First, we have to add the potential of the newly inserted key, which is at most $2\sqrt{n}$. (In case the newly inserted key is the new minimum, it does not contribute to the potential.)

The second, more subtle effect of `insert` is that it may cause a change in the rank-differences of other nodes. A given node y is affected by a newly inserted node x if the rank of x falls between the rank of y and that of its parent $p(y)$. If the affected node y is a light node (before the `insert`), then its rank $r(y)$ (before the `insert`) can be larger than $r(x)$ by at most $\sqrt{n} - 2$ (otherwise, the rank-difference of y would have been too large). There can be at most $\sqrt{n} - 1$ such nodes, and the potential of each may go up by 1. Otherwise, if y is a heavy node, then $\phi(y)$ may go up only by $1/\sqrt{n}$. Overall, the potential Φ increases by at most $2\sqrt{n}$. The amortized $O(\sqrt{n})$ cost of `insert` follows.

The reader may observe that we ignored the change in potential due to the change in the value of n. We can deal with this technicality by keeping n unchanged, as long as it does not get *too far* away from the true number of elements. When that happens, n can be updated by a standard doubling-halving strategy, without affecting the claimed amortized costs. We discuss this issue in more detail in the context of our main result.

2.2 The main result

▶ **Theorem 1.** *In the forward variant of pairing heaps, the amortized costs of* `delete-min` *and* `insert` *are* $O(\log n \cdot 4^{\sqrt{\log n}})$*, where* n *is the number of items in the heap when the operation is performed.*

To prove Theorem 1, we replace the simple potential function used in § 2.1 by one with a more fine-grained scaling. Again, first we present the tools necessary to analyse the heap in a *sorting mode*, i.e., we compute the cost of n `delete-min` operations on an arbitrary initial heap of size n, then we make the necessary changes to analyse both `insert` and `delete-min`.

In the following, for the purpose of analysis, we assume that n is an upper bound on the number of nodes in the heap, not greater than four times the true value. (Except for the beginning, when the heap is empty, and we set n to a small constant value, say $n = 4$.) At the end, we describe how we update n, if, after a certain number of operations, the true value reaches n, or falls far below n.

Let $q = q(n)$ be the *scaling factor* for n, an integer that we optimize later; assume for now that $1 < q < n$. The *category* of a (non-root) node x, denoted $c(x)$ is defined as $c(x) = \lfloor \log_q rd(x) \rfloor$. Observe that $c = c(x)$ is the unique integer for which $q^c \leq rd(x) < q^{c+1}$. For all nodes x we have $0 \leq c(x) < t$, where $t = t(n) = \lfloor \log_q (n-1) \rfloor + 1$, and $t \geq 2$.

For all non-root nodes x, we define the *node potential* of x as

$$\phi(x) = \frac{rd(x) - q^c}{q^c - q^{c-1}} + c \cdot q, \text{ where } c = c(x).$$

For the root r, we let $\phi(r) = 0$. The total node potential is $\Phi_N = \sum \phi(x)$, where the sum ranges over all nodes x. Some observations about the node potential are in order.

▶ **Lemma 3.** *With the previous definitions, we have:*

(i) *If* $rd(y) = rd(x) + 1$*, and* $rd(x) \geq 1$*, then* $\phi(y) = \phi(x) + 1/(q^c - q^{c-1})$*, where* $c = c(x)$*.*

(ii) *For every node* x*, it holds that* $0 \leq \phi(x) \leq t \cdot q$*, and therefore* $\Phi_N \leq n \cdot t \cdot q$*.*

(iii) *If two nodes of the same category are linked, then* Φ_N *decreases by at least 1.*

Proof.

(i) Suppose $c(y) = c(x) = c$. Then, $\phi(y) - \phi(x) = (rd(y) - rd(x))/(q^c - q^{c-1}) = 1/(q^c - q^{c-1})$. Otherwise, suppose $c(y) = c(x) + 1$. Let $c = c(x)$. Then, $rd(y) = q^{c+1}$, and $rd(x) = q^{c+1} - 1$. Thus, $\phi(y) - \phi(x) = (c+1) \cdot q - (q^{c+1} - 1 - q^c)/(q^c - q^{c-1}) - c \cdot q = 1/(q^c - q^{c-1})$.

(ii) From (i) it follows that $\phi(x)$ is maximal if $rd(x) = n - 1$. Then, $c(x) = t - 1$, and $\phi(x) \leq (q^t - q^{t-1})/(q^{t-1} - q^{t-2}) + (t-1) \cdot q = t \cdot q$.

(iii) Let us define the function $f(\cdot)$ that maps $rd(x)$ to $\phi(x)$ for all x. It is easy to verify that $f(\cdot)$ is strictly increasing for $rd(x) \geq 1$.
Suppose that $x < y$, and $c(x) = c(y) = c$. Then, $q^c \leq rd(x) < rd(y) < q^{c+1}$. Only the potential of y changes after the link operation, from $f(rd(y))$ to $f(rd(y) - rd(x))$.
We want to show $f(rd(y)) - f(rd(y) - rd(x)) \geq 1$. This quantity is minimized if $rd(x) = q^c$. By (i), $f(rd(y) - q^c) \leq f(rd(y)) - q^c/(q^c - q^{c-1}) \leq f(rd(y)) - 1$. The result follows. ◀

Suppose that $x_1, \ldots, x_k$ are children of the same node in the heap, indexed in left-to-right order. A subset $\{x_i, x_{i+1}, \ldots, x_j\}$ of these nodes, for $1 \leq i < j \leq k$, is referred to as a *contiguous group* of siblings. Within the heap, we consider certain contiguous groups of siblings to be *in a box*; this is only for the purpose of the analysis, with no effect on the implementation.

We define a second kind of potential to capture the current state of the boxes. Throughout the lifetime of the heap we maintain a number of invariants about the boxes, as follows.

(A) The size of a box (i.e., the number of nodes in a box) is of the form $2^b - 1$, for some $2 \leq b \leq t$.

(B) Boxes are pairwise disjoint.

(C) A box of size $2^b - 1$ can only contain nodes of category $b - 2$ or smaller.

The *box potential*, denoted Φ_B is a sum over all boxes of $(b - t - 1)/t$, where $2^b - 1$ is the size of the box. In particular, the largest possible box of size $2^t - 1$ contributes $-1/t$ to the box potential, and the smallest possible box of size 3 contributes $(1/t - 1)$. Observe that $-n < \Phi_B \leq 0$. At the beginning of the operations there are no boxes, therefore $\Phi_B = 0$.

The *total potential* Φ combines the node potential and the box potential, as $\Phi = \Phi_N + \Phi_B$.

We are ready to analyse the individual operations and their effect on the potential.

Delete-min only. Again, assume that the `delete-min` operation performs the pairing and accumulation rounds first, and deleting the root afterwards. We need to account for the change of node and box potential in all steps.

Consider an operation `link`(x, y), and assume w.l.o.g. that $x < y$, and therefore x becomes the parent of y. We say that the link is a *good link* if one of the following holds:

(1) before the link x and y were of the same category,

(2) the link occurs in the accumulation round, is not of type (1), and y is the left-to-right minimum before the link (being replaced by x in this role).

At a high level, our strategy is to show that for every $t \cdot 2^t$ link operations, one good link occurs. A good link of the type specified in case (1) decreases the node potential by at least 1. Therefore, by Lemma 3(ii), there are at most $n \cdot t \cdot q$ such links. (In our current setting no action increases the node potential.) A good link of the type specified in case (2) can occur at most $t - 1$ times during a `delete-min`. To see that there are no more than $t - 1$ type-(2) good links in a `delete-min`, observe that such an event necessarily leads to a decrease in category of the current left-to-right minimum, and the number of possible categories is t.

The link operations that we charge to a single good link may be scattered through multiple `delete-min` operations. The purpose of the boxes is to keep track of link operations that we have not yet accounted for.

Let k denote the number of children of the deleted root, i.e., the real cost of the operation, up to a constant factor, and let us look in turn at the pairing round, accumulation round, and the deletion of the root.

Pairing round. Consider the links involving nodes from a box of size $2^b - 1$. If at least one of these is a good link of type (1), we simply remove the box. The result is an increase in box potential by $(t - b + 1)/t$ due to the deletion of the box, and a decrease in node potential by at least 1 due to the good link. This amounts to a decrease in total potential of $(b - 1)/t \geq 1/t$. Observe that if the box size before the operation is $2^2 - 1 = 3$ (i.e., minimal), then the link operation within the box is *always* a good link. This is because, by invariant (C), all nodes in the box are of category 0.

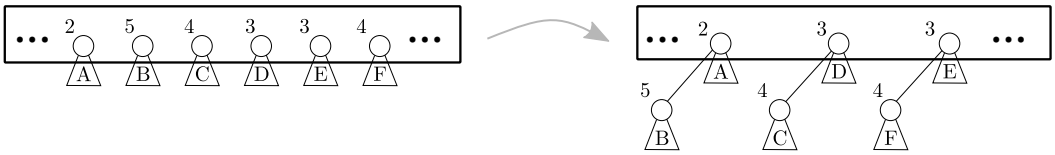


Figure 2 The evolution of a box. Numbers indicate categories of nodes. The losers of the comparisons drop out of the box.

Suppose that we process a box of size $2^b - 1$ for $b > 2$, and no good link occurs within the box. Then, the box continues to exist around the winners of the link operations where both nodes were in the same box, i.e., the *losers* of the link operations, and possibly the nodes "on the margin" leave the box. Observe that the size of the box goes from $2^b - 1$ to $2^{b-1} - 1$. Due to the decrease of the box size, we have a decrease in box potential by $1/t$, and consequently a decrease by $1/t$ in total potential. (See Figure 2 for illustration.)

Now suppose we execute $T = 2^t - 1$ consecutive non-good link operations without encountering a box. Then, we form a box around the $2^t - 1$ winners of these links. Due to the newly created box of maximum size, the total potential decreases by $1/t$.

Consider a sequence of T consecutive links. Then, either (i) one of the links is good, or (ii) we create a new box containing the T winners, or (iii) we encounter an existing box. In the latter case, we process the box until its end. Thus, in all cases, for at most $2T - 1$ consecutive link operations, we achieve a saving in potential of at least $1/t$. This yields (for the entire pairing round) a decrease in potential of at least

$$\left\lfloor \frac{\lfloor k/2 \rfloor}{2 \cdot 2^t - 3} \right\rfloor \cdot \frac{1}{t} \geq \frac{k}{4 \cdot t \cdot 2^t} - \frac{2}{t}.$$

We need to argue that in the cases described above, the box invariants are maintained. Condition (A) clearly holds in every case. When we create a new box, we only use nodes that are not in a box, therefore condition (B) holds. We never add new nodes to an existing box, thus (B) continues to hold. We next discuss the validity of invariant (C), which is the crucial ingredient of the proof.

When a new box of size $2^t - 1$ is created, we claim that there cannot be any node of category $t - 1$ within the box. This is because, if any node in the box was of category $t - 1$, its "winning" of the linking would have happened against another node of category $t - 1$, a type-(1) good link that contradicts our condition for creating the box.

Similarly, when shrinking an existing box from size $2^b - 1$ to $2^{b-1} - 1$, if invariant (C) was true before the operation (i.e., there were only nodes of category $0, \ldots, b - 2$ in the box), then the nodes in the box after the operation can only be of category $0, \ldots, b - 3$. A node of category $b - 2$ could only have "won" against another node of category $b - 2$, a type-(1) good link that contradicts our condition for maintaining the box. This establishes the invariants.

Accumulation round. By Lemma 3(i), the node potential can only further decrease. We need to argue that the box-structure is not destroyed by this round.

Recall that we start with the leftmost node (the current minimum), and keep linking against the nodes from left to right. As long as there is no type-(2) good link, the nodes that we encounter left-to-right become the children of the current minimum in right-to-left order. The box invariants are clearly not affected by this reversal of order, the box, together with its parent node simply moves further down the tree. (Categories of nodes in the box can only decrease.)

If there is a change in the minimum, i.e., a type-(2) event, while we are processing the nodes in a box, then we delete the box. By deleting the box, we may increase the box potential by $1 - 1/t < 1$. This is the only kind of potential-change we need to consider in this round, yielding an increase in total potential of at most $t - 1$ (since there are at most $t - 1$ type-(2) events).

Deleting the root. The actual deletion of the root leaves the node potential unchanged, since at that time the root has only one child, its successor, which has rank-difference 1, and consequently, node potential 0. The box potential is unaffected, since the root cannot be in a box.

To summarize, if the actual cost of `delete-min` is k, then, collecting the terms from the different rounds, the decrease in potential is at least $k/(4t \cdot 2^t) - 2/t - (t-1) \geq k/(4t \cdot 2^t) - t$.

Denoting the actual costs of the n `delete-min` oparations by $k_1, \ldots, k_n$, for the total potential decrease $\Delta\Phi$ we have:

$$\Delta\Phi \geq \left(\frac{1}{4t \cdot 2^t} \sum_{i=1}^{n} k_i \right) - n \cdot t.$$

From Lemma 3(i) it follows that $\Phi \leq n \cdot t \cdot q$, for an arbitrary heap. For the empty heap, we have $\Phi = 0$. Thus, $\Delta\Phi \leq n \cdot t \cdot q$. It follows that the total cost is $\sum k_i \leq (n \cdot t \cdot q + n \cdot t) \cdot (4t \cdot 2^t)$.

Let us now choose the scaling factor used in the analysis. Recall that $t = \lfloor \log_q (n-1) \rfloor$. Fixing $t = \lfloor \sqrt{\log n} \rfloor$, we have $q = O(n^{1/\sqrt{\log n}}) = O(2^{\sqrt{\log n}})$. Thus, we obtain that the amortized cost of `delete-min` is $O(\log n \cdot 4^{\sqrt{\log n}})$.

Insert and Delete-min. Next, we consider sequences of `insert` and `delete-min` operations, arbitrarily intermixed. The amortised cost of an operation is defined as the true cost *plus* the increase in potential due to the operation.

The analysis of `delete-min` is the same as before, yielding a decrease in total potential due to a `delete-min` of at least $k/(4t \cdot 2^t) - t$, where k is the actual cost of the operation.

The true cost of `insert` is $O(1)$, but we also need to consider the increase in potential. The box potential is not affected by `insert`, as we do not create any new boxes and neither the root, nor the newly inserted node are in a box.

When a key x is inserted, its first contribution is its own node potential $\phi(x)$, in case it becomes the child of the root. (If x is smaller than the root, then, after the linking, both x and the old root have node potential zero.) By Lemma 3(ii), the resulting increase in the node potential is at most $t \cdot q$.

As in § 2.1, we need to deal with the fact that an `insert` operation may change the ranks of existing nodes, possibly increasing their rank-difference. We could proceed with an argument similar to the one in § 2.1, estimating the change in potential for nodes in each category. The argument gets complicated, however, because the increase in rank-difference may also cause an increase in category, invalidating the box invariants.[9]

To avoid the issue of updating ranks during an `insert` operation, we redefine *rank* to take into account not just the items currently in the heap, but also the items that will be inserted into the heap in the future. (Since we use ranks only in the analysis, we can assume that future operations are known to us.) In this way, an `insert` operation has no effect on

[9] We observe that our problem of maintaining the ranks under insertions is reminiscent of the well-studied *ordered list maintenance* problem, for which efficient (although quite involved) solutions exist. Luckily, we can get away with a simpler solution.

ranks, rank-differences, or categories of existing items, since the rank of the newly inserted item has already been taken into consideration. Therefore, there is no further change in potential that we need to consider.

There remains the issue, that we cannot afford to look *too far* into the future, as we want the value n to be, at all times, close to the current size of the heap. Therefore, we split the operations into epochs, and keep the value of n fixed throughout an epoch. If the heap is empty, e.g., in the beginning, we let $n = 4$.

An epoch ends when the true size of the heap increases to n (after an `insert`), or decreases to $\lfloor n/4 \rfloor$ (after a `delete-min`), or if, within the epoch, n distinct keys have already been encountered (including those that were in the heap at the beginning of the epoch). In all three cases, we reset n to be twice the true size of the heap, remove all boxes, and start a new epoch. (We stress that epochs and boxes are used only in the analysis, with no effect on the actual operation of the data structure.) As mentioned, the ranks of the items are computed with respect to *all items* encountered during an epoch, they are therefore, fixed within the epoch. Clearly, all ranks and rank-differences are still upper bounded by n.

We make four observations, all of which are easy to verify based on the preceding discussion: (1) The true size of the heap is between $\lfloor n/4 \rfloor$ and n. (2) Within a finished epoch there must have been at least $\lceil n/4 \rceil$ operations. (3) The increase in node potential due to the resetting of n is not more than $n \cdot t \cdot q$. (4) Since the boxes are disjoint, there are less than n boxes that we are removing, increasing the total potential by at most n.

We distribute the increase in potential of at most $n \cdot t \cdot q + n$ among the $\lceil n/4 \rceil$ operations in the finished epoch, obtaining that the *increase* in potential during an `insert` is at most $5 \cdot t \cdot q + 4$, and the increase in potential during a `delete-min` is at most $t - k/(4t \cdot 2^t) + 4 \cdot t \cdot q + 4$. Scaling the potential by $t \cdot 2^t$, we obtain that the amortized cost of both `insert` and `delete-min` is $O(t^2 \cdot q \cdot 2^t)$.

Again, choosing $t = \lfloor \sqrt{\log n} \rfloor$, we obtain $q = O(n^{1/\sqrt{\log n}}) = O(2^{\sqrt{\log n}})$, and the amortized costs of $O(\log n \cdot 4^{\sqrt{\log n}})$ follow.

Meld. A `meld` operation is similar to an `insert`, in that only one node changes its rank-difference, and thereby its node potential (the root with the larger key). Furthermore, `meld` also leaves the box potential unaffected, so its analysis goes through similarly to the analysis of `insert`, yielding the same amortized cost for `meld` as for `insert` and `delete-min`. Observe, however, that if we also include `meld` operations in a sequence of operations, then the value n that appears in the cost no longer denotes just the size of the affected heap, it is instead, the number of items in *all heaps* that we are currently working with.

Other variants. The presented analysis is fairly general. It is not difficult to adapt it to the *standard* and *multipass* variants of pairing heaps, yielding similar upper bounds. For the standard variant the analysis is essentially the same as for the forward variant. For multipass, the analysis becomes simpler, since boxes need to be maintained only during individual `delete-min` operations. However, for these variants, stronger upper bounds are already known, as discussed in § 1.

Recall that during a `link` operation, the larger of the two items is linked as the leftmost child of the smaller. The reader may observe that the different behaviors of the standard and forward variants depend on this particular way of implementing `link`. If we were to link by making the larger item the *rightmost* child of the smaller, then the situation would reverse, with the forward variant being easy, and the standard variant hard to analyse.

We may modify the implementation of `link`, such as to link *arbitrarily* as the leftmost or rightmost child (i.e., deciding independently for every `link` operation whether to link at the left or at the right). Our analysis also extends to this more general class of pairing heap algorithms (that use the modified `link` implementation) with minimal changes. The only difficulty that arises in the new setting is that during the accumulation round, an existing box may split into two, one part going to the left, the other to the right side of the current minimum. To account for this loss, we need to start with a larger initial box (4^t instead of 2^t). The resulting bounds are still of the form $2^{O(\sqrt{\log n})}$.

Improving (significantly) the bounds presented in the paper and extending the analysis to other operations should be possible but will likely require new ideas.

References

1. G. M. Adelson-Velskiĭ and E. M. Landis. An algorithm for organization of information. *Dokl. Akad. Nauk SSSR*, 146:263–266, 1962.
2. Rudolf Bayer. Symmetric binary b-trees: Data structure and maintenance algorithms. *Acta Informatica*, 1(4):290–306, Dec 1972. doi:10.1007/BF00289509.
3. Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Self-adjusting binary search trees: What makes them tick? In *ESA 2015*, pages 300–312, 2015. doi:10.1007/978-3-662-48350-3_26.
4. Timothy M. Chan. *Quake Heaps: A Simple Alternative to Fibonacci Heaps*, pages 27–32. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. doi:10.1007/978-3-642-40273-9_3.
5. Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.
6. Amr Elmasry. Pairing heaps with $O(\log \log n)$ decrease cost. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 471–476, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496822.
7. Amr Elmasry. The violation heap: a relaxed fibonacci-like heap. *Discrete Math., Alg. and Appl.*, 2(4):493–504, 2010. doi:10.1142/S1793830910000838.
8. Michael L. Fredman. On the efficiency of pairing heaps and related data structures. *J. ACM*, 46(4):473–501, 1999. doi:10.1145/320211.320214.
9. Michael L. Fredman, Robert Sedgewick, Daniel Dominic Sleator, and Robert Endre Tarjan. The pairing heap: A new form of self-adjusting heap. *Algorithmica*, 1(1):111–129, 1986. doi:10.1007/BF01840439.
10. Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*, pages 338–346, 1984. doi:10.1109/SFCS.1984.715934.
11. Leonidas J. Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16-18 October 1978*, pages 8–21, 1978. doi:10.1109/SFCS.1978.3.
12. Bernhard Haeupler, Siddhartha Sen, and Robert Endre Tarjan. Rank-pairing heaps. *SIAM J. Comput.*, 40(6):1463–1485, 2011. doi:10.1137/100785351.
13. Thomas Dueholm Hansen, Haim Kaplan, Robert Endre Tarjan, and Uri Zwick. Hollow heaps. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 689–700, 2015. doi:10.1007/978-3-662-47672-7_56.

14 John Iacono. Improved upper bounds for pairing heaps. In *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings*, pages 32–45, 2000. doi:10.1007/3-540-44985-X_5.

15 John Iacono. In pursuit of the dynamic optimality conjecture. In *Space-Efficient Data Structures, Streams, and Algorithms*, volume 8066 of *Lecture Notes in Computer Science*, pages 236–250. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-40273-9_16.

16 John Iacono and Özgür Özkan. A tight lower bound for decrease-key in the pure heap model. *CoRR*, abs/1407.6665, 2014. URL: http://arxiv.org/abs/1407.6665.

17 John Iacono and Özgür Özkan. Why some heaps support constant-amortized-time decrease-key operations, and others do not. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 637–649, 2014. doi:10.1007/978-3-662-43948-7_53.

18 John Iacono and Mark Yagnatinsky. *A Linear Potential Function for Pairing Heaps*, pages 489–504. Springer International Publishing, Cham, 2016. doi:10.1007/978-3-319-48749-6_36.

19 Haim Kaplan and Robert Endre Tarjan. Thin heaps, thick heaps. *ACM Trans. Algorithms*, 4(1):3:1–3:14, 2008. doi:10.1145/1328911.1328914.

20 Irit Katriel, Peter Sanders, and Jesper Larsson Träff. A practical minimum spanning tree algorithm using the cycle property. In *Algorithms - ESA 2003, 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003, Proceedings*, pages 679–690, 2003. doi:10.1007/978-3-540-39658-1_61.

21 Donald E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1997.

22 Daniel H. Larkin, Siddhartha Sen, and Robert Endre Tarjan. A back-to-basics empirical study of priority queues. In *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2014, Portland, Oregon, USA, January 5, 2014*, pages 61–72, 2014. doi:10.1137/1.9781611973198.7.

23 Bernard M. E. Moret and Henry D. Shapiro. *An empirical analysis of algorithms for constructing a minimum spanning tree*, pages 400–411. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991. doi:10.1007/BFb0028279.

24 Seth Pettie. Towards a final analysis of pairing heaps. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 174–183, 2005. doi:10.1109/SFCS.2005.75.

25 Seth Pettie. Thirteen ways of looking at a splay tree. Unpublished lecture, 2014.

26 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985. doi:10.1145/3828.3835.

27 John T. Stasko and Jeffrey Scott Vitter. Pairing heaps: Experiments and analysis. *Commun. ACM*, 30(3):234–249, 1987. doi:10.1145/214748.214759.

28 Ashok Subramanian. An explanation of splaying. *J. Algorithms*, 20(3):512–525, 1996. doi:10.1006/jagm.1996.0025.

29 Jean Vuillemin. A data structure for manipulating priority queues. *Commun. ACM*, 21(4):309–315, 1978. doi:10.1145/359460.359478.

Simultaneous Multiparty Communication Protocols for Composed Functions

Yassine Hamoudi
IRIF, Université Paris Diderot, France
hamoudi@irif.fr
https://orcid.org/0000-0002-3762-0612

Abstract

The Number On the Forehead (NOF) model is a multiparty communication game between k players that collaboratively want to evaluate a given function $F : \mathcal{X}_1 \times \cdots \times \mathcal{X}_k \to \mathcal{Y}$ on some input $(x_1, \ldots, x_k)$ by broadcasting bits according to a predetermined protocol. The input is distributed in such a way that each player i sees all of it except x_i (as if x_i is written on the forehead of player i). In the Simultaneous Message Passing (SMP) model, the players have the extra condition that they cannot speak to each other, but instead send information to a referee. The referee does not know the players' inputs, and cannot give any information back. At the end, the referee must be able to recover $F(x_1, \ldots, x_k)$ from what she obtained from the players.

A central open question in the simultaneous NOF model, called the $\log n$ *barrier*, is to find a function which is hard to compute when the number of players is polylog(n) or more (where the x_i's have size poly(n)). This has an important application in circuit complexity, as it could help to separate ACC^0 from other complexity classes [22, 3]. One of the candidates for breaking the $\log n$ barrier belongs to the family of *composed functions*. The input to these functions in the k-party NOF model is represented by a $k \times (t \cdot n)$ boolean matrix M, whose row i is the number x_i on the forehead of player i and t is a block-width parameter. A symmetric composed function acting on M is specified by two symmetric n- and kt-variate functions f and g (respectively), that output $f \circ g(M) = f(g(B_1), \ldots, g(B_n))$ where B_j is the j-th block of width t of M. As the majority function $\textsc{Maj}$ is conjectured to be outside of ACC^0, Babai *et. al.* [5, 3] suggested to study the composed function $\textsc{Maj} \circ \textsc{Maj}_t$, with t large enough, for breaking the $\log n$ barrier (where $\textsc{Maj}_t$ outputs 1 if at least $kt/2$ bits of the input block are set to 1).

So far, it was only known that block-width $t = 1$ is not enough for $\textsc{Maj} \circ \textsc{Maj}_t$ to break the $\log n$ barrier in the simultaneous NOF model [3] (Chattopadhyay and Saks [17] found an efficient protocol for $t \leq$ polyloglog(n), but it requires randomness to be simultaneous). In this paper, we extend this result to any constant block-width $t > 1$ by giving a *deterministic simultaneous* protocol of cost $2^{\mathcal{O}(2^t)} \log^{2^{t+1}}(n)$ for *any* symmetric composed function $f \circ g$ (which includes $\textsc{Maj} \circ \textsc{Maj}_t$) when there are more than $2^{\Omega(2^t)} \log n$ players.

2012 ACM Subject Classification Theory of computation → Communication complexity

Keywords and phrases Communication complexity, Number On the Forehead model, Simultaneous Message Passing, Log n barrier, Symmetric Composed functions

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.14

Acknowledgements This work was initiated during a visit to Carnegie Mellon University. The author is very grateful to Anil Ada, who introduced him to the $\log n$ barrier problem and the $\textsc{Maj} \circ \textsc{Maj}_t$ conjecture for composed functions. He also thanks him for helpful discussions on this subject, as well as the anonymous referees for their valuable comments and suggestions which helped to improve this paper.

43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018).
Editors: Igor Potapov, Paul Spirakis, and James Worrell; Article No. 14; pp. 14:1–14:15

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

1 Introduction

1.1 Number On the Forehead and Simultaneous models

The *Number On the Forehead* (NOF) model is a multiparty communication model introduced by Chandra, Furst and Lipton [15] that generalizes the two player communication game of Yao [36]. In this model, k players are given k inputs $x_1 \in \mathcal{X}_1, \ldots, x_k \in \mathcal{X}_k$ on which they want to compute some function $F : \mathcal{X}_1 \times \cdots \times \mathcal{X}_k \to \mathcal{Y}$. Each player i sees all of the input $(x_1, \ldots, x_k)$, except x_i. The situation is as if input x_i is written on the forehead of player i.

In order to collaboratively evaluate $F(x_1, \ldots, x_k)$, the players communicate by *broadcasting* bits according to a predetermined *protocol*. This protocol specifies whose turn it is to speak, and which bit is to be sent given the information exchanged so far and the input seen by the speaking player. It also determines when communication stops. At the end, all the players must be able to recover $F(x_1, \ldots, x_k)$ from the input they see and the transcript of the exchange. The cost of the protocol on input $(x_1, \ldots, x_k)$ is the number of exchanged bits, and the total cost is the worst case cost on all inputs. The k-party deterministic communication complexity of F, denoted $\mathsf{D}_k(F)$, is the cost of the most efficient protocol computing F.

In most of the settings, the x_i's are poly n-bits long (for some parameter n) and $\mathcal{Y} = \{0, 1\}$. In this case, the naive protocol is to broadcast first the entire input x_1 (this can be done by player 2), and then player 1 computes $F(x_1, \ldots, x_k)$ and sends the result to the other players. This protocol has cost $m + 1$ (where $m = \text{poly}(n)$ is the number of bits required for sending x_1), which proves $\mathsf{D}_k(F) = \mathcal{O}(\text{poly}\, n)$. Consequently, a protocol will be said to be *efficient* if it has cost $\mathcal{O}(\text{polylog}\, n)$ (i.e. we seek for exponential speed-up over the naive protocol).

Among the many variants of the previous framework (randomized, quantum, etc.), we will be interested in the *simultaneous* (or *Simultaneous Message Passing - SMP*) model [36, 25, 5, 28] in which the players cannot speak to each other but instead send information to a referee. The referee does not know the players' inputs, and cannot give any information back. At the end, the referee must be able to recover $F(x_1, \ldots, x_k)$ from what she obtained from the players. The simultaneous deterministic communication complexity is denoted $\mathsf{D}^{||}_k(F)$, and it always satisfies $\mathsf{D}_k(F) \leq \mathsf{D}^{||}_k(F)$. It has often been easier to reason first in this weaker model for proving lower bounds [3, 28, 10, 7]. It is also more suitable and fruitful for studying certain functions, such as Equality in the two party setting [36, 2, 24, 4, 14, 19, 13]. We will show in the next section that the simultaneous deterministic communication model is also closely connected to lower bound results in the complexity class ACC^0.

1.2 The log n barrier problem and ACC^0 lower bounds

The NOF model has proved to be of value in the study of many areas of computer science, such as branching programs [15], Ramsey theory [15], circuit complexity [22, 12], quasirandom graphs [18], proof complexity [9], etc. One of the most interesting connections, pointed out by Håstad and Goldmann [22] and refined in [3], is a way to derive lower bounds for the complexity class[1] ACC^0 from lower bounds in the *simultaneous* NOF model. More precisely, according to a result from Yao, Beigel and Tarui [37, 12], any function $f \in \mathsf{ACC}^0$ can be expressed as a depth-2 circuit whose top gate is a symmetric gate of fan-in $2^{\log^c n}$, and each bottom gate is an And gate of fan-in $\log^d n$ (for some constants c, d). Consequently, for

[1] ACC_0 refers to the functions computable by constant-depth poly-size circuits with unbounded fan-in And, OR, Not and Mod_m gates (where Mod_m outputs 0 iff the sum of its inputs is divisible by m).

any partition of the input of f between $k = \log^d n - 1$ players in the *simultaneous* NOF model, there exists a partition of the AND gates between the players such that each of them sees all the input bits she needs to evaluate the gates she received. The players can then send to the referee the number of gates that evaluate to 1, which enables the referee to compute f. The total cost of this protocol is $\mathcal{O}\left(k \log\left(2^{\log^c n}\right)\right) = \mathcal{O}\left(\log^{c+d} n\right)$. Conversely, any super-polylogarithmic lower bound in the *simultaneous* NOF model for a function f and a partition of its input between polylog(n) players would imply $f \notin \mathsf{ACC}^0$.

Separating ACC^0 from other complexity classes is a central question in complexity theory. It is conjectured that ACC^0 does not contain the majority function MAJ, but the only result known so far is $\mathsf{NEXP} \not\subset \mathsf{ACC}^0$ [35]. The aforementioned connection with communication complexity has motivated the search for a function which is hard to compute for $k \geq \log n$ players in the simultaneous NOF model. This problem is called the $\log n$ *barrier*.

Obtaining lower bounds in the NOF model is a challenging task, as the current methods become very weak when $k \geq \log n$. The only general lower bound technique known so far is the discrepancy method and its variants [6, 18, 30, 31]. One of the early application of it was an $\Omega(n/4^k)$ lower bound on the randomized complexity of the *Generalized Inner Product* (GIP) function [6]. A long series of generalizations and improvements of the discrepancy method subsequently led to an $\Omega\left(\frac{\sqrt{n}}{k2^k}\right)$ (resp. $\Omega(n/4^k)$) lower bound on the randomized (resp. deterministic) complexity of the *Disjointness* (DISJ) function [34, 11, 16, 23, 8, 33, 32, 29]. It might seem like other lower bound arguments could prove that GIP and DISJ remain hard for $k \geq \log n$ players. However, surprising non-simultaneous [21, 1] and simultaneous [3, 1] protocols proved that the aforementioned lower bounds are nearly optimal, and that these two functions cannot break the $\log n$ barrier. Very recently, Podolskii and Sherstov [27] showed that the randomized complexity of GIP and DISJ is exactly $\Theta\left(\frac{\log n}{\lceil 1+k/\log n \rceil} + 1\right)$ when $k \geq \log n$, and built a function having complexity $\Omega(\log n)$ independently of k. Although these last results do not break the $\log n$ barrier, they are the first superconstant lower bounds proved for explicit functions when $k \geq \log n$.

1.3 Composed Functions

An input $x_1, \ldots, x_k \in \{0,1\}^n$ to k players in the NOF model can be visualized as a $k \times n$ boolean matrix M where row i is the number x_i on the forehead of player i. The protocols known so far for GIP and DISJ strongly rely on the particular way these functions act on matrix M. They both consist in applying the $g = \text{AND}$ function on each of the n columns of M, followed by the $f = \text{MOD}_2$ (for GIP) or $f = \text{NOR}$ (for DISJ) function on the n resulting bits. Since GIP and DISJ do not break the $\log n$ barrier, a natural move has been to try other f and g functions, and to increase the number t of columns on which each g function applies. These are called the *composed functions*, formally defined below and depicted in Figure 1.

▶ **Definition 1** (Boolean input version)**.** Fix a block-width parameter $t \geq 1$, and consider functions $f : \{0,1\}^n \to \{0,1\}$ and $\vec{g} = (g_1, \ldots, g_n)$ where $g_j : (\{0,1\}^t)^k \to \{0,1\}$. Given $x_1, \ldots, x_k \in \{0,1\}^{t \cdot n}$, the *composed function* $f \circ \vec{g}$ for k players outputs $f \circ \vec{g}(x_1, \ldots, x_k) = f(g_1(B_1), \ldots, g_n(B_n))$ where $B_j \in (\{0,1\}^t)^k$ is the j^{th} block of width t in the matrix representation M of the input. When $g = g_1 = \cdots = g_n$, we denote $f \circ \vec{g}$ by $f \circ g$.

Both $\text{GIP} = \text{MOD}_2 \circ \text{AND}$ and $\text{DISJ} = \text{NOR} \circ \text{AND}$ are composed functions for $t = 1$, with the additional property that MOD_2, NOR and AND are *symmetric* functions (i.e. invariant under any permutation of their input). Since the majority function MAJ is conjectured to be outside of ACC^0, Babai *et. al.* [5, 3] suggested to look at $\text{MAJ} \circ \text{MAJ}_t$ and $\text{MAJ} \circ \text{THR}_t^s$ for

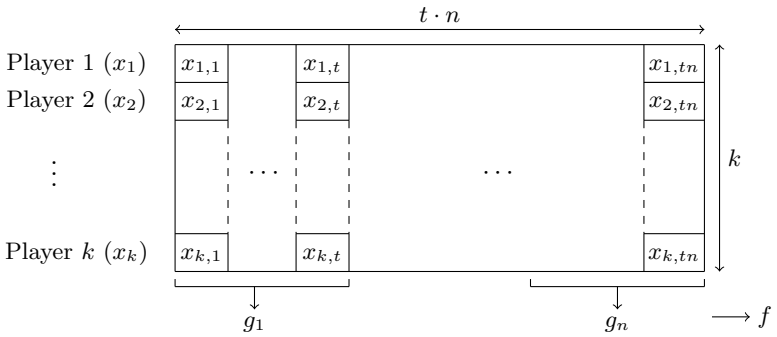


Figure 1 Matrix structure of a composed function $f \circ \vec{g}$ of block-width t.

breaking the $\log n$ barrier (where $\textsc{Maj}_t$ outputs 1 if at least $kt/2$ bits of the input block are set to 1, and $\textsc{Thr}_t^s(r_1, \ldots, r_k) = 1$ if $r_1 + \cdots + r_k \geq s$ for $r_1, \ldots, r_k$ seen as t-bits numbers).

Another way to look at composed functions of block-width t is to interpret each sub-row $r \in \{0,1\}^t$ of each block as a number in $\mathbb{Z}_d$, where $d = 2^t$. This representation of the input as a $k \times n$ matrix M over some set $\mathbb{Z}_d$ is sometimes more convenient to use. Below, we reformulate Definition 1 using this point of view.

▶ **Definition 2** (Integer input version)**.** Fix an integer $d \geq 2$ and consider functions $f : \{0,1\}^n \to \{0,1\}$ and $\vec{g} = (g_1, \ldots, g_n)$ where $g_j : \mathbb{Z}_d^k \to \{0,1\}$. Given $x_1, \ldots, x_k \in \mathbb{Z}_d^n$, the *composed function* $f \circ \vec{g}$ for k players outputs $f \circ \vec{g}(x_1, \ldots, x_k) = f(g_1(C_1), \ldots, g_n(C_n))$ where $C_j \in \mathbb{Z}_d^k$ is the j^{th} column in the matrix representation M of the input. When $g = g_1 = \cdots = g_n$, we denote $f \circ \vec{g}$ by $f \circ g$.

The set of all composed functions $f \circ \vec{g}$ (resp. $f \circ g$) over $\mathbb{Z}_d$ is denoted $\mathsf{ANY} \circ \overrightarrow{\mathsf{ANY}}_{\mathbb{Z}_d}$ (resp. $\mathsf{ANY} \circ \mathsf{ANY}_{\mathbb{Z}_d}$). Similarly, $\mathsf{SYM} \circ \mathsf{SYM}_{\mathbb{Z}_d}$ is the set of $f \circ g$ for symmetric f and symmetric g functions, $\mathsf{SYM} \circ \overrightarrow{\mathsf{ANY}}_{\mathbb{Z}_d}$ is the set of $f \circ \vec{g}$ for symmetric f and any $\vec{g}$, etc. If $d = 2$ (which corresponds to block-width $t = 1$), we will drop the subscript and write $\mathsf{ANY} \circ \overrightarrow{\mathsf{ANY}}$, $\mathsf{SYM} \circ \mathsf{SYM}$, etc. We have for instance $\textsc{Gip}, \textsc{Disj} \in \mathsf{SYM} \circ \mathsf{SYM}$ and $\textsc{Maj} \circ \textsc{Maj}_t, \textsc{Maj} \circ \textsc{Thr}_t^s \in \mathsf{SYM} \circ \mathsf{SYM}_{\mathbb{Z}_{2^t}}$.

The first efficient protocol for composed functions with polylog(n) or more players was given by Grolmusz [21]. It is a *non-simultaneous* protocol of cost $\mathcal{O}\left(\log^2 n\right)$ for any composed function in $\mathsf{SYM} \circ \textsc{And}$ (the inner function is fixed to be $\textsc{And}$) when $k \geq \log n$. The study of composed functions with symmetric outer function f was subsequently continued, as it captures many other interesting cases in communication complexity. Babai *et. al.* [5] proposed first $\textsc{Maj} \circ \textsc{Maj}_1$ as a candidate to break the $\log n$ barrier. However, in a subsequent work [3], they found a *simultaneous* protocol that applies to $\mathsf{SYM} \circ \mathsf{COMP}^c$ (where COMP^c holds for *c-compressible symmetric* functions[2], a subset of SYM that contains $\textsc{Maj}$ and $\textsc{And}$). It has cost $\mathcal{O}\left(\log^{2+c} n\right)$ when $k > 1 + \log n$. Later, Ada *et. al.* [1] generalized this result to $\mathsf{SYM} \circ \overrightarrow{\mathsf{ANY}}$, with a *simultaneous* protocol of cost $\mathcal{O}\left(\log^3 n\right)$ for $k > 1 + 2\log n$ players. The only protocol known so far for block-width $t > 1$ has been discovered by Chattopadhyay and Saks [17]. It has cost $\mathcal{O}\left(d \log n \log(dn)\right)$ for $\mathsf{SYM} \circ \overrightarrow{\mathsf{ANY}}_{\mathbb{Z}_d}$ when $k > 1 + d\log(3n)$ (which is

[2] A class $\mathcal{G}$ (parameterized by k) of symmetric functions $g : \{0,1\}^k \to \{0,1\}$ is c-compressible if for any function $g \in \mathcal{G}$, set $S \subsetneq \{1, \ldots, k\}$ and input $(x_i)_{i \in S} \in \{0,1\}^{|S|}$ there is a message m_S of size $\mathcal{O}(1) + c\log(k - |S|)$ such that $g(x_1, \ldots, x_k)$ can be computed for all $(x_i)_{i \in \{1,\ldots,k\} \setminus S} \in \{0,1\}^{k-|S|}$ from knowledge of m_S and $(x_i)_{i \in \{1,\ldots,k\} \setminus S}$ only. The $\textsc{Maj}_1$ and $\textsc{Thr}_1^s$ functions are 1-compressible [3].

Table 1 Deterministic protocols for different families of composed functions. The top three results apply only to block-width 1 (i.e. $d = 2$), whereas the last two results work for any d. Note that the protocol of [17] can be made simultaneous using shared randomness between the players.

	Supported functions	Complexity of the protocol	Simultaneous	Number of players required
Grolmusz [21]	$\mathsf{SYM} \circ \textsc{And}$	$\mathcal{O}\left(\log^2 n\right)$	No	$k \geq \log n$
Babai *et. al.* [3]	$\mathsf{SYM} \circ \mathsf{COMP}^c$	$\mathcal{O}\left(\log^{2+c} n\right)$	Yes	$k > 1 + \log n$
Ada *et. al.* [1]	$\mathsf{SYM} \circ \overrightarrow{\mathsf{ANY}}$	$\mathcal{O}\left(\log^3 n\right)$	Yes	$k > 1 + 2\log n$
C. and Saks [17]	$\mathsf{SYM} \circ \overrightarrow{\mathsf{ANY}}_{\mathbb{Z}_d}$	$\mathcal{O}\left(d \log n \log(dn)\right)$	No	$k > 1 + d\log(3n)$
This work	$\mathsf{SYM} \circ \overrightarrow{\mathsf{SYM}}_{\mathbb{Z}_d}$	$2^{\mathcal{O}(d)} \log^{4d}(n)$	Yes	$k \geq 4^{2d} \log n$

efficient for $d \leq \text{polylog}\, n$). However, it is *not simultaneous* in the deterministic setting (the authors showed how to make it simultaneous using shared randomness between the players). Thus, none of these previous results prevents from breaking the $\log n$ barrier in the SMP model with composed functions of block-width as small as $t = 2$. The goal of this paper is to rule out this possibility for all symmetric composed functions of constant block-width $t > 1$.

1.4 Summary of Results and Comparison to Previous Protocols

Below, we describe our main results, and summarize in Table 1 the complexity of all the known protocols for composed functions. Then, we review the main ideas used in the previous literature, and we explain how we differ from them.

Our results. In this paper, we describe the first *deterministic simultaneous* protocol for symmetric composed functions of block-width $t > 1$. Our result is divided into two parts. We first give (Section 3.1) a protocol of cost $\mathcal{O}\left(k(k+d)^{d-1} \log n\right)$ for $\mathsf{SYM} \circ \mathsf{SYM}_{\mathbb{Z}_d}$ when the number of players is $k \geq 4^{d-1} \log n$. In a second time (Section 3.2), we build upon this result to give a simultaneous protocol of cost $2^{\mathcal{O}(d)} \log^{2 \cdot 2^{\lceil \log d \rceil}}(n)$ for $\mathsf{SYM} \circ \overrightarrow{\mathsf{SYM}}_{\mathbb{Z}_d}$ when $k \geq 4^{2d} \log n$. Unlike the first protocol, this last result also works with different inner functions $g_1, \ldots, g_n$ and it is efficient even if k is super-polylogarithmic.

Adjacent vertices of the $\{0,1\}^n$ hypercube. For block-width $t = 1$ and an input matrix $M \in \{0,1\}^{k \times n}$, denote n_c the number of times column $c \in \{0,1\}^k$ occurs in M. Grolsmusz [21] noticed that if $c_1, \ldots, c_m$ is a sequence of adjacent vertices of the $\{0,1\}^k$ hypercube (i.e. c_{l+1} differs from c_l by exactly one coordinate) then $n_{c_1} = \left(\sum_{l=1}^{m-1} (-1)^{l+1} (n_{c_l} + n_{c_{l+1}})\right) + (-1)^{m+1} n_{c_m}$. Moreover, if position i is the coordinate at which c_l and c_{l+1} differ, then the quantity $n_{c_l} + n_{c_{l+1}}$ is known by player i. This leads to a straightforward simultaneous protocol of cost $\mathcal{O}(k \log n)$ for computing n_{c_1}, provided that n_{c_m} is known by the referee. In his initial work, Grolsmusz [21] gave a non-simultaneous way to find some initial n_{c_m}. Ada *et. al.* [1] noticed later that this step can be made simultaneous using the protocol of Babai *et. al.* [3], and that the idea of Grolsmusz (initially designed for $\mathsf{SYM} \circ \textsc{And}$) easily adapts to $\mathsf{SYM} \circ \overrightarrow{\mathsf{ANY}}$. Unfortunately, this "hypercube view" does not generalize to block-width $t > 1$: for each i and $c \in (\{0,1\}^t)^k$, the number of vertices that differ from c only at position i is now $2^t - 1 > 1$. It is easy to see that writing a similar telescoping sum as above, in which each term would be known by a player, is no longer possible.

Counting up to symmetry. Given a $k \times n$ matrix M over $\mathbb{Z}_d$, for all $0 \leq e_1 + \cdots + e_{d-1} \leq k$

denote $y_{e_1,\ldots,e_{d-1}}$ the number of columns of M with exactly e_s occurrences of each $s \in \mathbb{Z}_d \backslash \{0\}$ (we do not put e_0 since it is always equal to $k - (e_1 + \cdots + e_{d-1})$). These numbers provide less information than the n_c's defined above, but they still unable us to compute $f \circ g(M)$ for all $f \circ g \in \mathsf{SYM} \circ \mathsf{SYM}_{\mathbb{Z}_d}$. If M is distributed between k players in the NOF model (player i does not see row i), a naive *simultaneous* protocol is to have each player i send the number of columns $a^i_{e_1,\ldots,e_{d-1}}$ which contain, *from her point of view*, exactly e_s occurrences of each element $s \in \{1, \ldots, d-1\}$ (for all $e_1 + \cdots + e_{d-1} \leq k-1$). Babai *et. al.* [3] analyzed this protocol in the case $d = 2$, and showed that it gives the referee enough information to recover the $y_{e_1,\ldots,e_{d-1}}$'s, provided that $k > 1 + \log n$. In Section 3.1, we extend this analysis to any $d > 2$. The core of the proof, as in [3], is to define a specific equation (using the $a^i_{e_1,\ldots,e_{d-1}}$'s) whose only integral solution is the $y_{e_1,\ldots,e_{d-1}}$'s.

The shifted basis technique. The only protocol [17] known prior to this work for block-width $t > 1$ is based on the following observation: given polynomial representations of the inner functions g_j (over variables $x_{1,j}, \ldots, x_{k,j}$), each term involving strictly less than k variables can be evaluated on input matrix M by at least one player (in fact, by all the players that have one of the missing variables on their foreheads). The key idea of [17] is to get rid of the remaining terms by expressing the g_j in a s-shifted basis where all terms of degree k will evaluate to 0 on M (shifting for instance monomial $x_{1,j} \cdots x_{k,j}$ by $s = (s_1, \ldots, s_k)$ means to replace it with $(x_{1,j} - s_1) \cdots (x_{k,j} - s_k)$). To this end, it would suffice to find some s that shares at least one coordinate in common with each column of M. Provided that k is large enough, [17] showed that a randomly picked s has this property with high probability. This gives rise to a simultaneous protocol for $\mathsf{SYM} \circ \overrightarrow{\mathsf{ANY}}_{\mathbb{Z}_d}$ *if* the players have access to a shared random string. In the deterministic setting (no shared randomness), it is not known how to make this protocol simultaneous.

Different inner functions, and reducing the number of players. The communication complexity is expected to decrease as k grows up (since the overlap of information among the players increases). However, this fact is not reflected in the cost of our first protocol (Section 3.1). This issue is closely related to that of having different inner functions $g_1, \ldots, g_n$. Indeed, the problem of computing $f \circ g \in \mathsf{SYM} \circ \mathsf{SYM}_{\mathbb{Z}_d}$ with k players on a matrix $M \in \mathbb{Z}_d^{k \times n}$ can be changed into computing $f \circ (\widetilde{g_1}, \ldots, \widetilde{g_n}) \in \mathsf{SYM} \circ \overrightarrow{\mathsf{SYM}}_{\mathbb{Z}_d}$ with the first $\ell < k$ players on the submatrix $\widetilde{M} \in \mathbb{Z}_d^{\ell \times n}$ (first ℓ rows of M), where $\widetilde{g_j} : \mathbb{Z}_d^\ell \to \{0,1\}$ is defined as $\widetilde{g_j}(u) = g(u \cdot v_j)$ and v_j is the values occurring from row $\ell + 1$ to k in the j-th column of M (note that the new $\widetilde{g_j}$ functions are still symmetric, but unknown to the referee). Our first protocol cannot handle different inner functions, but this issue will be solved in Section 3.2 where we describe a protocol for $\mathsf{SYM} \circ \overrightarrow{\mathsf{SYM}}_{\mathbb{Z}_d}$ based on a new use of the *polynomial representations* (different than [17]). We will show that each inner function $\widetilde{g_j}$ can be represented into a (small) basis of symmetric functions $\{m_a\}_a$ (Section 2), which will allow us to split the problem of computing $f \circ (\widetilde{g_1}, \ldots, \widetilde{g_n})$ on $\widetilde{M}$ into computing each $f \circ m_a \in \mathsf{SYM} \circ \mathsf{SYM}_{\mathbb{Z}_d}$ on a well-chosen matrix $\widetilde{M_a}$. This last step can be done with the initial protocol of Section 3.1.

2 Polynomial Representations for Symmetric Functions

Throughout this paper, $\mathbb{Z}_d$ will denote the set of integers $\{0, \ldots, d-1\}$ and $\mathbb{F}_p$ is the finite field with p elements. Furthermore, a function $f : \mathcal{X}^m \to \mathcal{Y}$ is said to be *m-symmetric* (or *symmetric*) if it is invariant under any permutation of the input variables (i.e. for any input $(x_1, \ldots, x_m)$ and permutation $\sigma \in S_m$, we have $f(x_1, \ldots, x_m) = f(x_{\sigma(1)}, \ldots, x_{\sigma(m)})$).

The protocol designed in Section 3.2 for composed functions $f \circ \vec{g}$ requires a concise polynomial representation of the inner functions $g_1, \ldots, g_n : (\{0,1\}^t)^k \to \{0,1\}$. Informally,

we look for a field K and polynomials $G_j \in K[X]$ with variables $X = (x_{u,v})_{1\leq u\leq k, 1\leq v\leq t}$, such that:

- **(a)** for all $x \in (\{0,1\}^t)^k$, $g_j(x) = G_j(x)$
- **(b)** the order of K is at least $n+1$ (so that the set $\{0,\dots,n\}$ of values taken by $\sum_j g_j(x^{(j)})$ for $x^{(1)},\dots,x^{(n)} \in (\{0,1\}^t)^k$ can be embedded into K)
- **(c)** the G_j polynomials can be represented in a basis of size $\mathcal{O}(\operatorname{poly} k)$ when t is constant
- **(d)** the values of the coefficients of the G_j polynomials in this basis are less than n^c, for some absolute constant c independent of k and t.

The first step towards this end is to look at the usual $\mathbb{R}$-*multilinear representation* (also called *Fourier expansion* [26]) of a function $g : (\{0,1\}^t)^k \to \{0,1\}$. For each $a = (a_{u,v})_{1\leq u\leq k, 1\leq v\leq t} \in (\{0,1\}^t)^k$ we define the *indicator polynomial* $1_{\{a\}}(x)$ to be $1_{\{a\}}(x) = \prod_{1\leq u\leq k, 1\leq v\leq t}(1 - a_{u,v} + (2a_{u,v}-1)x_{u,v})$. It is easy to see that it takes value 1 when $x = a$ and value 0 when $x \in (\{0,1\}^t)^k \setminus \{a\}$. Consequently, we have $g(x) = \sum_{a\in(\{0,1\}^t)^k} g(a)1_{\{a\}}(x)$ for all $x \in (\{0,1\}^t)^k$. If we let x^a be the monomial $\prod_{(u,v):a_{u,v}=1} x_{u,v}$, then there exist real coefficients $\widehat{g}(a)$ such that it can be rewritten as the following multilinear polynomial

$$g(x) = \sum_{a\in(\{0,1\}^t)^k} \widehat{g}(a)x^a \tag{1}$$

Moreover, the $\widehat{g}(a)$ coefficients are given by the Möbius inversion formula

$$\widehat{g}(a) = \sum_{a'\subseteq a} (-1)^{|a|-|a'|} g(a') \tag{2}$$

where $|a|$ is the number of 1 in $a \in (\{0,1\}^t)^k$, and $a' \subseteq a$ means $a'_{u,v} = 0$ whenever $a_{u,v} = 0$.

Polynomial (1) is called the $\mathbb{R}$-*multilinear representation* of function g. It satisfies requirements (a) and (b) above, but not requirement (c). Indeed, these polynomials are expressed in the basis of monomials $\{x^a\}_{a\in(\{0,1\}^t)^k}$ which has size $2^{t\cdot k}$.

In order to reduce the size of the basis, we restrict ourselves to the k-symmetric functions $g : (\{0,1\}^t)^k \to \{0,1\}$ (as will be the case in Section 3.2). This condition leads to the following equalities between coefficients.

▶ **Lemma 3.** *For any $a = (a_1,\dots,a_k) \in (\{0,1\}^t)^k$ and any permutation $\sigma \in S_k$, if $g : (\{0,1\}^t)^k \to \{0,1\}$ is a k-symmetric function then the coefficients $\widehat{g}(a)$ and $\widehat{g}(\sigma(a))$ in the $\mathbb{R}$-multilinear representation of g are equal (where $\sigma(a) = (a_{\sigma(1)},\dots,a_{\sigma(k)})$).*

Proof. The proof is direct from Equation (2). ◀

This lemma motivates the definition of the following polynomials, that will be used to obtain a basis for the k-symmetric functions over $(\{0,1\}^t)^k$.

▶ **Definition 4.** Given $a \in (\{0,1\}^t)^k$, the *monomial k-symmetric polynomial* $m_a(x)$ over variables $(x_{u,v})_{1\leq u\leq k, 1\leq v\leq t}$ is defined to be the sum of all the *distinct* monomials $x^{\sigma(a)}$ where $\sigma \in S_k$ ranges over all the permutations.

▶ **Example 5.** If $(t,k) = (2,3)$ and $a = ((1,1),(0,1),(0,1))$ then $m_a(x) = x_{1,1}x_{1,2}x_{2,2}x_{3,2} + x_{1,2}x_{2,1}x_{2,2}x_{3,2} + x_{1,2}x_{2,2}x_{3,1}x_{3,2}$.

According to Lemma 3, any k-symmetric function $g : (\{0,1\}^t)^k \to \{0,1\}$ can be expressed as a linear combination of monomial k-symmetric polynomials. From this observation, we can derive a basis for the k-symmetric functions by taking all the *distinct* monomial k-symmetric polynomials. We specify a subset of elements $a \in (\{0,1\}^t)^k$ that corresponds to this basis.

▶ **Definition 6.** We define a tuple $a = (a_1, \ldots, a_k) \in (\{0,1\}^t)^k$ to be *sorted*, if $|a_u| \leq |a_{u'}|$ for all $1 \leq u \leq u' \leq k$, and $a_u \leq_{lex} a_{u'}$ whenever $|a_u| = |a_{u'}|$ (where $|a_u|$ is the Hamming weight of a_u, and $\leq_{lex}$ is the lexicographic order over $\{0,1\}^t$). The set of all the sorted tuples over $(\{0,1\}^t)^k$ is denoted $\mathcal{S}(t,k)$.

▶ **Lemma 7.** *The set $\{m_a(x) : a \in \mathcal{S}(t,k)\}$ is a basis for the k-symmetric functions $g : (\{0,1\}^t)^k \to \{0,1\}$. Moreover, it has size $\binom{k+2^t-1}{2^t-1}$.*

Proof. It is straightforward to see that all the possible monomial k-symmetric polynomials belong to $\{m_a(x) : a \in \mathcal{S}(t,k)\}$, and that no two elements in this set have a monomial in common. Thus, it is a basis for the k-symmetric functions.

Consider the total order $\prec$ over $\{0,1\}^t$ defined as $a_u \prec a_{u'}$ if and only if $|a_u| \leq |a_{u'}|$, or $|a_u| = |a_{u'}|$ and $a_u \leq_{lex} a_{u'}$. Each $a \in \mathcal{S}(t,k)$ can be seen as a (distinct) non-decreasing sequence of length k from the totally ordered set $(\{0,1\}^t, \prec)$ of size 2^t. The total number of such sequences is known to be $\binom{k+2^t-1}{2^t-1}$. ◀

Finally, given a parameter n, we want the coefficients of the k-symmetric functions in the chosen basis to be less than n^c for some constant c independent of k and t (requirement (d)). To this end, it suffices to reformulate the previous results over a field $\mathbb{F}_p$, for some prime $p \in (n, 2n)$. We obtain the following polynomial representation for k-symmetric functions:

▶ **Proposition 8.** *Any k-symmetric function $g : (\{0,1\}^t)^k \to \{0,1\}$ can be written as*

$$g(x) = \sum_{a \in \mathcal{S}(t,k)} c_a(g) \cdot m_a(x) \mod p$$

where $p \in (n, 2n)$ is prime, $c_a(g) \in \mathbb{F}_p$ and m_a is the monomial k-symmetric polynomial corresponding to the sorted tuple a. Moreover, $\mathcal{S}(t,k)$ has size $\binom{k+2^t-1}{2^t-1}$.

3 Simultaneous Protocol for $\mathsf{SYM} \circ \overrightarrow{\mathsf{SYM}}_{\mathbb{Z}_d}$

We now describe in detail our simultaneous protocol for symmetric composed functions. The result is divided into two parts. We first give in Section 3.1 a protocol of cost $\mathcal{O}\left(k(k+d)^{d-1} \log n\right)$ for $\mathsf{SYM} \circ \mathsf{SYM}_{\mathbb{Z}_d}$ when $k \geq 4^{d-1} \log n$. This is a generalization of the idea of [3], which was based on solving a particular equation. We build upon this result in Section 3.2 to give an efficient protocol of cost $\mathcal{O}\left(\log^{4d}(n)\right)$ for $\mathsf{SYM} \circ \overrightarrow{\mathsf{SYM}}_{\mathbb{Z}_d}$ when $k \geq 4^{2d} \log n$ and d is constant. This last result uses the protocol of Theorem 9 as a subroutine, and the polynomial representations described in Section 2.

3.1 The Equation Solving part

We extend the protocol for $\mathsf{SYM} \circ \mathsf{SYM}_{\mathbb{Z}_2}$ from [3] to any $d > 1$. It applies to all functions in $\mathsf{SYM} \circ \mathsf{SYM}_{\mathbb{Z}_d}$ as long as $k \geq 4^{d-1} \log n$, but it is not efficient if d is nonconstant or if the number k of players is super-polylogarithmic (we will remove this last condition in the next section). For convenience in the proof, we state the result over $\mathbb{Z}_{d+1}$ instead of $\mathbb{Z}_d$:

▶ **Theorem 9.** *Let M be a $k \times n$ matrix over $\mathbb{Z}_{d+1}$, where $n \geq 2$ and $d \geq 1$. For $0 \leq e_1 + \cdots + e_d \leq k$, denote $y_{e_1,\ldots,e_d}$ the number of columns of M with exactly e_s occurrences of each $s \in \mathbb{Z}_{d+1} \backslash \{0\}$. For each $i = 1, \ldots, k$, let player i see all of M except row i. If $k \geq 4^d \log n$ then there exists a deterministic simultaneous NOF protocol of cost $k\binom{k+d}{d}\lceil \log n \rceil$, at the end of which the referee knows all the $y_{e_1,\ldots,e_d}$'s.*

Proof. The communication part of the protocol is pretty simple: each player i sends to the referee the number of columns $a^i_{e_1,\dots,e_d}$ which contain, *from her point of view* (i.e. without taking row i into account), exactly e_s occurrences of each element $s \in \{1,\dots,d\}$ (for all $e_1+\cdots+e_d \leq k-1$).

The referee computes then $b_{e_1,\dots,e_d} = \sum_{i=1}^{k} a^i_{e_1,\dots,e_d}$ (for all $e_1+\cdots+e_d \leq k-1$). The important thing to note is that these numbers must verify the following equalities:

$$\begin{cases} (k-(e_1+\cdots+e_d))y_{e_1,\dots,e_d} + \sum_{s=1}^{d}(e_s+1)y_{e_1,\dots,e_{s-1},e_s+1,e_{s+1},\dots,e_d} = b_{e_1,\dots,e_d} \\ 0 \leq e_1+\cdots+e_d \leq k-1 \end{cases} \tag{3}$$

To see why it is true, consider a column C of M that contributes to a given $b_{e_1,\dots,e_d}$. Either C contains exactly e_s occurrences of each element $s \in \{1,\dots,d\}$, or there is one $s' \in \{1,\dots,d\}$ that occurs $e_{s'}+1$ times in C (the other s having exactly e_s occurrences in C). In the first case, C contributes to $y_{e_1,\dots,e_d}$ and to the quantity $a_i(e_1,\dots,e_d)$ of each player i having a 0 entry of C on her forehead (there are $k-(e_1+\cdots+e_d)$ such players). In the second case, C contributes to $y_{e_1,\dots,e_{s'-1},e_{s'}+1,e_{s'+1},\dots,e_d}$ and to the quantity $a_i(e_1,\dots,e_d)$ of each player i having a s' entry of C on her forehead (there are $e_{s'}+1$ such players). Thus, the total contribution for $b_{e_1,\dots,e_d}$ is $(k-(e_1+\cdots+e_d))y_{e_1,\dots,e_d}+\sum_{s'=1}^{d}(e_{s'}+1)y_{e_1,\dots,e_{s'-1},e_{s'}+1,e_{s'+1},\dots,e_d}$.

Equalities (3) can be seen as a system of equations whose unknowns are the $y_{e_1,\dots,e_d}$'s. Since the referee is not computationally restricted she can enumerate all the integral solutions, but she does not know which one corresponds to matrix M. The key lemma is to show that Equations (3), under mild constraints

$$y_{e_1,\dots,e_d} \geq 0,\ 0 \leq e_1+\cdots+e_d \leq k \quad \textit{and} \quad \sum_{e_1+\cdots+e_d \leq k} y_{e_1,\dots,e_d} \leq n \tag{4}$$

have at most one integral solution when $k \geq 4^d \log n$. We prove it by induction on d (the base case $d=1$ corresponds to the work of [3], the induction step is more involved and is given in Appendix A). Consequently, the referee is able to know unambiguously the correct $y_{e_1,\dots,e_d}$'s that correspond to M. This protocol is clearly simultaneous since the players do not need to talk to each other. Each of the k players sends $\binom{k+d}{d}$ numbers $a_i(e_1,\dots,e_d) \leq n$. Thus the total communication cost is at most $k\binom{k+d}{d}\lceil \log n \rceil$. ◀

▶ **Corollary 10.** *Let $n \geq 2$, $d \geq 2$ and suppose $k \geq 4^{d-1}\log n$. There is a deterministic simultaneous NOF protocol of cost $k\binom{k+d-1}{d-1}\lceil \log n \rceil$, at the end of which the referee can compute* all *composed functions $f \circ g \in \mathsf{SYM} \circ \mathsf{SYM}_{\mathbb{Z}_d}$ of her choice.*

This result can also be adapted to the case of $k < 4^d \log n$ players by splitting the initial matrix into sufficiently many parts. Previously, Ada *et. al.* [1] also generalized their work to any number k of players, by giving a protocol of cost $\mathcal{O}\left(n/2^k \cdot \log n + k \log n\right)$ for $\mathsf{SYM} \circ \overrightarrow{\mathsf{ANY}}$. However, it was not simultaneous and it did not apply to $t > 1$.

▶ **Proposition 11.** *Let M be a $k \times n$ matrix over $\mathbb{Z}_{d+1}$, where $n \geq 2$ and $d \geq 1$. For $0 \leq e_1+\cdots+e_d \leq k$, denote $y_{e_1,\dots,e_d}$ the number of columns of M with exactly e_s occurrences of each $s \in \mathbb{Z}_{d+1}\backslash\{0\}$. For each $i = 1,\dots,k$, let player i see all of M except row i. If $4^d \leq k < 4^d \log n$ then there exists a deterministic simultaneous NOF protocol of cost at most $\mathcal{O}\left(\frac{n}{2^{k/4^d}} \cdot (k+d)^{d+2}\right)$, at the end of which the referee knows all the $y_{e_1,\dots,e_d}$'s.*

Proof. We split M into $\left\lceil \frac{n}{\lfloor 2^{k/4^d} \rfloor} \right\rceil$ matrices, each of size $k \times \left\lfloor 2^{k/4^d} \right\rfloor$ (except one matrix that can have less columns). These matrices have few enough columns to apply (separately) the

protocol of Theorem 9 on them. The $y_{e_1,\dots,e_d}$'s for the original matrix M are computed by recombining all the obtained results. The total cost is $\mathcal{O}\left(\frac{n}{2^{k/4^d}} \cdot k\binom{k+d}{d} \log\left(2^{k/4^d}\right)\right)$. ◀

3.2 The Polynomial Representation part

Using the polynomial representation of Proposition 8, we give a protocol that improves upon Corollary 10 in two ways: it is still efficient when k is super-polylogarithmic, and the inner functions $g_1, \dots, g_n$ can be different (i.e. it applies to $\mathsf{SYM} \circ \overrightarrow{\mathsf{SYM}}_{\mathbb{Z}_d}$ instead of $\mathsf{SYM} \circ \mathsf{SYM}_{\mathbb{Z}_d}$).

▶ **Theorem 12.** *Let $n \geq 2$, $d \geq 2$ and suppose $k \geq 4^{2^{\lceil \log d \rceil}} \log n$. For any composed function $f \circ \vec{g} \in \mathsf{SYM} \circ \overrightarrow{\mathsf{SYM}}_{\mathbb{Z}_d}$ there exists a deterministic simultaneous NOF protocol that computes it with cost $4^{2^{\lceil \log d \rceil + 2}} \log^{2 \cdot 2^{\lceil \log d \rceil}}(n)$.*

Proof. Let $\vec{g} = (g_1, \dots, g_n)$. In order to use the polynomial representation of Section 2, we change the range of the g_j functions as $g_j : (\{0,1\}^t)^k \to \{0,1\}$, where $t = \lceil \log d \rceil$. This requires to encode each number $x \in \mathbb{Z}_d$ as an element $\bar{x} \in \{0,1\}^t$. If d is not a power of two then some $y \in \{0,1\}^t$ will not correspond to any $x \in \mathbb{Z}_d$. We extend each g_j as the zero function on inputs that contain such numbers (note that the functions are still k-symmetric).

The input is now a $k \times (t \cdot n)$ boolean matrix M. Each function g_j acts on the j^{th} block of M, which will be denoted $B_j \in (\{0,1\}^t)^k$. Let $\ell = 4^{2^t} \log n$, so that only the first ℓ players are going to speak. For each block B_j, if we let $v_j \in (\{0,1\}^t)^{(k-\ell)}$ be the sub-block occurring from row $\ell+1$ to k, then $g_j : (\{0,1\}^t)^k \to \{0,1\}$ induces a new function $\widetilde{g_j} : (\{0,1\}^t)^\ell \to \{0,1\}$ such that $\widetilde{g_j}(u) = g_j(u \cdot v_j)$. Moreover, $\widetilde{g_j}$ is still a symmetric function. Thus, our task reduces to find an efficient simultaneous protocol for $f \circ (\widetilde{g_1}, \dots, \widetilde{g_n})$ with $\ell = 4^{2^t} \log n$ players. We denote $\widetilde{M}$ the $\ell \times (t \cdot n)$ submatrix of M on which we now work, and $\widetilde{B_j} \in (\{0,1\}^t)^\ell$ is the sub-block of B_j corresponding to $\widetilde{M}$.

We cannot apply directly the protocol of Theorem 9, since it only works for equal inner functions $\widetilde{g_1} = \dots = \widetilde{g_n}$. Instead, we use first Proposition 8 on the $\widetilde{g_j}$ functions: for each $j \in \{1, \dots, n\}$ there exist coefficients $(c_a(\widetilde{g_j}))_{a \in \mathcal{S}(t,\ell)}$ over $\mathbb{F}_p$ such that $\widetilde{g_j}(x) = \sum_{a \in \mathcal{S}(t,\ell)} c_a(\widetilde{g_j}) \cdot m_a(x) \mod p$ where $p \in (n, 2n)$, m_a is the monomial k-symmetric polynomial corresponding to the sorted tuple a and $|\mathcal{S}(t,\ell)| = \binom{\ell + 2^t - 1}{2^t - 1}$. The coefficients $c_a(\widetilde{g_j})$ are known by the first ℓ players, but not by the referee (since they depend on rows $\ell+1$ to k of M).

For each $a \in \mathcal{S}(t,\ell)$, the players build a new matrix $\widetilde{M_a}$ of size $\ell \times (c_a(\widetilde{g_1}) + \dots + c_a(\widetilde{g_n}))$ where block $\widetilde{B_j}$ from $\widetilde{M}$ is copied $c_a(\widetilde{g_j}) \in [0, 2n)$ times. Note that $\widetilde{M_a}$ has at most $2n^2$ blocks, and there are enough players $\ell = 4^{2^t} \log n$ for applying (the boolean input version of) the simultaneous protocol of Theorem 9. It allows the referee to know the number of blocks of $\widetilde{M_a}$ which are equal – up to row permutation – to any $\widetilde{B} \in (\{0,1\}^t)^\ell$. This information is sufficient to compute $\sum_{j=1}^n c_a(\widetilde{g_j}) \cdot m_a(\widetilde{B_j})$ since the m_a functions are k-symmetric.

Finally, the referee sums these quantities modulo p over all a. It gives her $\sum_{a \in \mathcal{S}(t,\ell)} \sum_{j=1}^n c_a(\widetilde{g_j}) \cdot m_a(\widetilde{B_j}) \mod p = \sum_{j=1}^n \widetilde{g_j}(\widetilde{B_j}) \mod p$. Since $\sum_{j=1}^n \widetilde{g_j}(\widetilde{B_j}) \leq n$ and $p > n$, it equals $\sum_{j=1}^n \widetilde{g_j}(\widetilde{B_j}) = \sum_{j=1}^n g_j(B_j)$. Knowing this, the referee can compute $f \circ (g_1, \dots, g_n)(M)$ since f is symmetric.

Regarding the cost of the protocol, we applied $|\mathcal{S}(t,\ell)| = \binom{\ell + 2^t - 1}{2^t - 1}$ times the protocol of Theorem 9, with ℓ players and inputs of size at most $2n^2$. Thus the total cost is at most $\binom{\ell+2^t-1}{2^t-1} \cdot \ell \binom{\ell+2^t-1}{2^t-1} \lceil \log 2n^2 \rceil \leq \ell(\ell + 2^t)^{2^{t+1}-2} \log n$. Since $\ell = 4^{2^t} \log n$ and $t = \lceil \log d \rceil$, this is less than $4^{2^t + (2^t+1)(2^{t+1}-2)} \log^{2^{t+1}}(n) \leq 4^{2^{\lceil \log d \rceil + 2}} \log^{2 \cdot 2^{\lceil \log d \rceil}}(n)$. ◀

4 Conclusion and Open Problems

One of the main open problems in communication complexity remains to find a function which is hard to compute for $k \geq \log n$ players in the simultaneous Number On the Forehead model. We discarded this possibility for the composed functions in $\mathsf{SYM} \circ \overrightarrow{\mathsf{SYM}}_{\mathbb{Z}_d}$ (for constant d) by giving the first efficient *deterministic simultaneous* protocol for composed functions of block-width $t > 1$. In the non-simultaneous setting, the best result so far applies to $\mathsf{SYM} \circ \overrightarrow{\mathsf{ANY}}_{\mathbb{Z}_d}$ and $d = \mathcal{O}(\text{polylog}\, n)$ [17]. Extending these protocols to larger d, bigger families of composed functions or to the simultaneous setting (for [17]) would give a better insight on composed functions. Indeed, it is conjectured that the $\log n$ barrier can be broken by such functions for large d, two of the candidates being $\textsc{Maj} \circ \textsc{Maj}_t$ and $\textsc{Maj} \circ \textsc{Thr}_t^s$.

Note that both the Equation Solving and the Polynomial Representation parts of our protocol are bottleneck for handling non-constant d in our result. It could be interesting to restrict to smaller families than symmetric functions (or to choose specific inner or outer functions, such as threshold functions), or to find other relevant equations that could be solved by the referee with fewer information than in our protocol.

Apart from composed functions, there are a few other candidates for breaking the $\log n$ barrier. Some of them are matrix related problems, such as deciding the top-left entry of the multiplication of k matrices in $\mathbb{F}_2^{n \times n}$ (an $\Omega(n/2^k)$ lower bound has been obtained by Raz [30]). More recently, Gowers and Viola [20] studied the interleaved group products, where each player receives a tuple $(x_{i,1}, \ldots, x_{i,n})$ in $G = SL(2, q)$, with the promise that $\prod_{i=1}^{n} x_{1,i} \cdots x_{k,i} = g$ *or* h. Finding which is the case has cost $\Omega(n \log |G|)$ when $k = 2$, and it is conjectured to remain hard for larger k.

References

1 A. Ada, A. Chattopadhyay, O. Fawzi, and P. Nguyen. The NOF multiparty communication complexity of composed functions. *Computational Complexity*, 24(3):645–694, 2015.

2 A. Ambainis. Communication complexity in a 3-computer model. *Algorithmica*, 16(3):298–301, 1996.

3 L. Babai, A. Gál, P. G. Kimmel, and S. V. Lokam. Communication complexity of simultaneous messages. *SIAM J. Comput.*, 33(1):137–166, 2004.

4 L. Babai and P. G. Kimmel. Randomized simultaneous messages: solution of a problem of Yao in communication complexity. In *Proceedings of Computational Complexity. Twelfth Annual IEEE Conference*, pages 239–246, 1997.

5 L. Babai, P. G. Kimmel, and S. V. Lokam. Simultaneous messages vs. communication. In *12th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 361–372. Springer, 1995.

6 L. Babai, N. Nisan, and M. Szegedy. Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs. *J. Comput. Syst. Sci.*, 45(2):204–232, 1992.

7 Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. Information theory methods in communication complexity. In *Proceedings 17th IEEE Annual Conference on Computational Complexity*, pages 72–81, 2002.

8 P. Beame and T. Huynh. Multiparty communication complexity and threshold circuit size of AC0. *SIAM Journal on Computing*, 41(3):484–518, 2012.

9 P. Beame, T. Pitassi, and N. Segerlind. Lower bounds for Lovász–Schrijver systems and beyond follow from multiparty communication complexity. *SIAM J. Comput.*, 37(3):845–869, 2007.

10 P. Beame, T. Pitassi, N. Segerlind, and A. Wigderson. A direct sum theorem for corruption and the multiparty NOF communication complexity of set disjointness. In *Proceedings of*

the 20th Annual IEEE Conference on Computational Complexity, CCC '05, pages 52–66. IEEE Computer Society, 2005.

11 P. Beame, T. Pitassi, N. Segerlind, and A. Wigderson. A strong direct product theorem for corruption and the multiparty communication complexity of disjointness. *Comput. Complex.*, 15(4):391–432, 2006.

12 R. Beigel and J. Tarui. On ACC. *Computational Complexity*, 4(4):350–366, 1994.

13 R. C. Bottesch, D. Gavinsky, and H. Klauck. Equality, revisited. *CoRR*, abs/1511.01211, 2015.

14 H. Buhrman, R. Cleve, J. Watrous, and R. de Wolf. Quantum fingerprinting. *Phys. Rev. Lett.*, 87:167902, 2001.

15 A. K. Chandra, M. L. Furst, and R. J. Lipton. Multi-party protocols. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 94–99. ACM, 1983.

16 A. Chattopadhyay and A. Ada. Multiparty communication complexity of disjointness. *arXiv preprint arXiv:0801.3624*, 2008.

17 A. Chattopadhyay and M. E. Saks. The power of super-logarithmic number of players. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, 2014.

18 F. R. K. Chung and P. Tetali. Communication complexity and quasi randomness. *SIAMJDiscreteMath*, 6(1):110–123, 1993.

19 D. Gavinsky, O. Regev, and R. de Wolf. Simultaneous communication protocols with quantum and classical messages. *Chicago Journal of Theoretical Computer Science*, 7, 2008.

20 T. Gowers and E. Viola. The communication complexity of interleaved group products. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 351–360. ACM, 2015.

21 V. Grolmusz. The BNS lower bound for multi-party protocols is nearly optimal. *Information and Computation*, 112:51–54, 1994.

22 J. Håstad and M. Goldmann. On the power of small-depth threshold circuits. *Computational Complexity*, 1(2):113–129, 1991.

23 T. Lee and A. Shraibman. Disjointness is hard in the multiparty number-on-the-forehead model. *Computational Complexity*, 18(2):309–336, 2009.

24 I. Newman and M. Szegedy. Public vs. private coin flips in one round communication games (extended abstract). In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 561–570. ACM, 1996.

25 N. Nisan and A. Wigderson. Rounds in communication complexity revisited. *SIAM J. Comput.*, 22(1):211–219, 1993.

26 R. O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

27 V. V. Podolskii and A. A. Sherstov. Inner product and set disjointness: Beyond logarithmically many parties. *CoRR*, abs/1711.10661, 2017.

28 P. Pudlák, V. Rödl, and J. Sgall. Boolean circuits, tensor ranks, and communication complexity. *SIAM J. Comput.*, 26(3):605–633, 1997.

29 A. Rao and A. Yehudayoff. Simplified lower bounds on the multiparty communication complexity of disjointness. In *Proceedings of the 30th Conference on Computational Complexity*, CCC '15, pages 88–101, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

30 R. Raz. The BNS-Chung criterion for multi-party communication complexity. *Computational Complexity*, 9:2000, 2000.

31 A. A. Sherstov. The pattern matrix method. *SIAM Journal on Computing*, 40(6):1969–2000, 2011.

32 A. A. Sherstov. Communication lower bounds using directional derivatives. *J. ACM*, 61(6):34:1–34:71, 2014.

33 A. A. Sherstov. The multiparty communication complexity of set disjointness. *SIAM Journal on Computing*, 45(4):1450–1489, 2016.

34 P. Tesson. *Computational Complexity Questions Related to Finite Monoids and Semigroups*. PhD thesis, McGill University, Montreal, Canada, 2003.

35 R. Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2:1–2:32, 2014.

36 A. Yao. Some complexity questions related to distributive computing. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 209–213. ACM, 1979.

37 A. Yao. On ACC and threshold circuits. In *Proceedings 31st Annual Symposium on Foundations of Computer Science*, pages 619–627 vol.2, 1990.

A Lemma for the Equation Solving part

In this section, we prove the following lemma:

▶ **Lemma 13.** *Let $n \geq 2$, $d \geq 1$ and $k \geq 4^d \log n$. Let $(b_{e_1,\dots,e_d})_{0 \leq e_1+\cdots+e_d \leq k-1}$ be integers. Consider the following system of equations:*

$$\begin{cases} (k - (e_1 + \cdots + e_d))y_{e_1,\dots,e_d} + \sum_{s=1}^{d} (e_s + 1) y_{e_1,\dots,e_{s-1},e_s+1,e_{s+1},\dots,e_d} = b_{e_1,\dots,e_d} \\ 0 \leq e_1 + \cdots + e_d \leq k - 1 \end{cases} \tag{5}$$

Assume further that

$$y_{e_1,\dots,e_d} \geq 0, \ 0 \leq e_1 + \cdots + e_d \leq k \quad \text{and} \quad \sum_{e_1+\cdots+e_d \leq k} y_{e_1,\dots,e_d} \leq n \tag{6}$$

Then, under constraints (6), the system of equations (5) has at most one integral solution.

We will show a stronger result:

▶ **Lemma 14.** *Let $n \geq 2$, $d \geq 1$ and $k > 4^d \log n - d$. Let $(b_{e_1,\dots,e_d})_{0 \leq e_1+\cdots+e_d \leq k-1}$ be integers. Consider the following system of equations:*

$$\begin{cases} (k - (e_1 + \cdots + e_d))z_{e_1,\dots,e_d} + \sum_{s=1}^{d} (e_s + 1) z_{e_1,\dots,e_{s-1},e_s+1,e_{s+1},\dots,e_d} = 0 \\ 0 \leq e_1 + \cdots + e_d \leq k - 1 \end{cases} \tag{7}$$

Assume further that

$$\sum_{e_1+\cdots+e_d \leq k} |z_{e_1,\dots,e_d}| \leq 2n \tag{8}$$

Then, under constraints (8), the system of equations (7) cannot have a non-zero integral solution.

Proof that Lemma 14 implies Lemma 13. Assume by contradiction that Equations (5) under Constraints (6) have two different integral solutions $y = (y_{e_1,\dots,e_d})_{0 \leq e_1+\cdots+e_d \leq k}$ and $y' = (y'_{e_1,\dots,e_d})_{0 \leq e_1+\cdots+e_d \leq k}$ for $k \geq 4^d \log n$. Define $z_{e_1,\dots,e_d} = y_{e_1,\dots,e_d} - y'_{e_1,\dots,e_d}$. It is easy

to see that it must verify (7), and since $y \neq y'$ there is at least one $z_{e_1,\dots,e_d} \neq 0$. Finally, since $z_{e_1,\dots,e_d} = |y_{e_1,\dots,e_d} - y'_{e_1,\dots,e_d}| \leq y_{e_1,\dots,e_d} + y'_{e_1,\dots,e_d}$, we have

$$\sum_{e_1+\dots+e_d\leq k} |z_{e_1,\dots,e_d}| \leq \sum_{e_1+\dots+e_d\leq k} (y_{e_1,\dots,e_d} + y'_{e_1,\dots,e_d}) \leq 2n$$

◀

Proof of Lemma 14. We prove the result by induction on d. The base case has already been established in [3], we recall it for completeness.

Base case (d = 1). We denote $(z_i)_{0\leq i\leq k}$ the variables. Equations (7) under Constraints (8) become

$$\begin{cases} (k-i)z_i + (i+1)z_{i+1} = 0, \ i = 0, 1, \dots, k-1 \\ \sum_{i=0}^{k} |z_i| \leq 2n \end{cases}$$

Thus, $z_1 = -kz_0 = -\binom{k}{1}z_0$, $z_2 = -\frac{k-1}{2}z_1 = \binom{k}{2}z_0$, and more generally $z_i = (-1)^i\binom{k}{i}z_0$. Consequently, if $(z_i)_{0\leq i\leq k}$ is a nonzero integral solution, then $z_0 \neq 0$ and $|z_i| = \binom{k}{i}|z_0| \geq \binom{k}{i}$ for all i. We obtain a contradiction: $2n \geq \sum_{i=0}^{k}|z_i| \geq \sum_{i=0}^{k}\binom{k}{i} = 2^k > 2^{4\log n-1} > 2n$. Thus, Lemma 14 is true for $d = 1$.

Induction step. Assuming that Lemma 14 is true for $d-1$, we prove that it is also the case for $d \geq 2$. Suppose by contradiction that Equations (7) under Constraints (8) have a non-zero integral solution $z = (z_{e_1,\dots,e_d})_{0\leq e_1+\dots+e_d\leq k}$ for $k > 4^d \log n - d$. As in the proof of the base case, we want to show $\sum_{e_1+\dots+e_d\leq k}|z_{e_1,\dots,e_d}| > 2n$, which would be a contradiction.

To this end, we are going to focus for each $0 \leq i \leq k$ on the largest element of $\{|z_{e_1,\dots,e_d}| : e_1 + \dots + e_d = i\}$. We define

$$Z_i = \max_{e_1+\dots+e_d=i} |z_{e_1,\dots,e_d}| \quad \text{and} \quad k^+ = \min\{i : Z_i \neq 0\}$$

Since z is a nonzero solution, k^+ is well defined. We conduct the proof as follows:

(a) Using the induction hypothesis, we show that the first nonzero Z_i must occur for $i = k^+ \leq 4^{d-1}\log n - (d-1)$.

(b) The sequence $(Z_i)_i$ verifies $\frac{k-i}{i+d}Z_i \leq Z_{i+1}$.

(c) Using the two previous results, we prove $\sum_{i=0}^{k} Z_i > 2n$.

The contradiction comes then from $\sum_{i=0}^{k} Z_i \leq \sum_{e_1+\dots+e_d\leq k}|z_{e_1,\dots,e_d}| \leq 2n$

Proof of (a). Assume $k^+ > 0$ (otherwise the result is trivial). According to Equations (7), and knowing that $z_{e_1,\dots,e_d} = 0$ whenever $e_1 + \dots + e_d < k^+$, we have

$$\sum_{s=1}^{d}(e_s+1)z_{e_1,\dots,e_s-1,e_s+1,e_{s+1},\dots,e_d} = 0$$

for all $e_1 + \dots + e_d = k^+ - 1$. If we set apart the last term $z_{e_1,\dots,e_{d-1},e_d+1}$, we obtain

$$(k^+ - (e_1 + \dots + e_{d-1}))z_{e_1,\dots,e_{d-1},e_d+1} + \sum_{s=1}^{d-1}(e_s+1)z_{e_1,\dots,e_s-1,e_s+1,e_{s+1},\dots,e_d} = 0$$

Let $z'_{e_1,\dots,e_{d-1}} = z_{e_1,\dots,e_{d-1},k^+-(e_1+\cdots+e_{d-1})}$ for all $0 \leq e_1+\cdots+e_{d-1} \leq k^+$. We can change the variables in the previous equation as follows

$$\begin{cases} (k^+ - (e_1+\cdots+e_{d-1}))z'_{e_1,\dots,e_{d-1}} + \sum_{s=1}^{d-1}(e_s+1)z'_{e_1,\dots,e_{s-1},e_s+1,e_{s+1},\dots,e_{d-1}} = 0 \\ 0 \leq e_1+\cdots+e_{d-1} \leq k^+ - 1 \end{cases}$$

This is equivalent to Equations (7) at rank $d-1$. Moreover, $\sum_{e_1+\cdots+e_{d-1}\leq k^+} |z'_{e_1,\dots,e_{d-1}}| \leq 2n$, and there exists $e_1+\cdots+e_d = k^+$ such that $z_{e_1,\dots,e_d} \neq 0$ (by definition of k^+), i.e. $z'_{e_1,\dots,e_{d-1}} \neq 0$. Consequently, it corresponds to a nonzero integral solution to Equations (7) under Constraints (8) at rank $d-1$ with parameter k^+. According to our induction hypothesis it implies $k^+ \leq 4^{d-1}\log n - (d-1)$.

Proof of (b). Setting apart $z_{e_1,\dots,e_d}$ in Equations (7), and using the triangle inequality, we obtain

$$(k-(e_1+\cdots+e_d))|z_{e_1,\dots,e_d}| \leq \sum_{s=1}^{d}(e_s+1)|z_{e_1,\dots,e_{s-1},e_s+1,e_{s+1},\dots,e_d}|$$

for all $e_1+\cdots+e_d \leq k$. In particular, if we choose $e_1+\cdots+e_d$ such that $Z_{e_1+\cdots+e_d} = |z_{e_1,\dots,e_d}|$ then

$$\begin{aligned}(k-(e_1+\cdots+e_d))Z_{e_1+\cdots+e_d} &\leq \sum_{s=1}^{d}(e_s+1)|z_{e_1,\dots,e_{s-1},e_s+1,e_{s+1},\dots,e_d}| \\ &\leq \sum_{s=1}^{d}(e_s+1)Z_{e_1+\cdots+e_d+1} \\ &\leq (e_1+\cdots+e_d+d)Z_{e_1+\cdots+e_d+1}\end{aligned}$$

Thus $(k-i)Z_i \leq (i+d)Z_{i+1}$, where $i = e_1+\cdots+e_d$.

Proof of (c). Using (b), first note for $i > k^+$ that

$$\begin{aligned}Z_i &\geq \frac{k-(i-1)}{(i-1)+d}\cdot\frac{k-(i-2)}{(i-2)+d}\cdots\frac{k-k^+}{k^++d}\cdot Z_{k^+} \\ &= \frac{(k-k^+)!}{(k-i)!}\cdot\frac{(k^++d-1)!}{(i+d-1)!}\cdot Z_{k^+} \\ &= \frac{(k+d-1)!}{(k-i)!(i+d-1)!}\cdot\frac{(k-k^+)!(k^++d-1)!}{(k+d-1)!}\cdot Z_{k^+} \\ &= \binom{k+d-1}{i+d-1}\binom{k+d-1}{k^++d-1}^{-1}\cdot Z_{k^+} \\ &\geq \binom{k+d-1}{i+d-1}\binom{k+d-1}{k^++d-1}^{-1} \qquad \text{since } Z_{k^+}\geq 1\end{aligned}$$

According to (a) and our initial hypothesis on k, we have $k^++d-1 \leq 4^{d-1}\log n \leq (k+d-1)/4$. Thus $\sum_{i=k^+}^{k}\binom{k+d-1}{i+d-1} \geq \frac{1}{2}\cdot\sum_{i=0}^{k+d-1}\binom{k+d-1}{i} = 2^{k+d-2}$ and $\binom{k+d-1}{k^++d-1}^{-1} \geq 2^{-(k+d-1)H(1/4)}$ (using the well-known bound $\binom{m}{\alpha m} \leq 2^{mH(\alpha)}$ where $H(\alpha) = -\log(\alpha^\alpha(1-\alpha)^{1-\alpha})$). Consequently, since $d \geq 2$ and $n \geq 2$, we obtain $\sum_{i=k^+}^{k} Z_i \geq 2^{(1-H(1/4))(k+d-1)-1}$ $\geq 2^{(1-H(1/4))4^d\log n - 1} > 2n$. ◀

Sliding Windows over Context-Free Languages

Moses Ganardi
Universität Siegen, Germany
ganardi@eti.uni-siegen.de

Artur Jeż
University of Wrocław, Poland
aje@cs.uni.wroc.pl

Markus Lohrey
Universität Siegen, Germany
lohrey@eti.uni-siegen.de

Abstract

We study the space complexity of sliding window streaming algorithms that check membership of the window content in a fixed context-free language. For regular languages, this complexity is either constant, logarithmic or linear [4]. We prove that every context-free language whose sliding window space complexity is $\log_2(n) - \omega(1)$ must be regular and has constant space complexity. Moreover, for every $c \in \mathbb{N}$, $c \geq 1$ we construct a (nondeterministic) context-free language whose sliding window space complexity is $\mathcal{O}(n^{1/c}) \setminus o(n^{1/c})$. Finally, we give an example of a deterministic one-counter language whose sliding window space complexity is $\Theta((\log n)^2)$.

2012 ACM Subject Classification Theory of computation → Streaming models

Keywords and phrases sliding windows, streaming algorithms, context-free languages

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.15

1 Introduction

In many streaming applications, data items are outdated after a certain time and the sliding window model is a simple way to model this: *Sliding window algorithms* process an input sequence $a_1 a_2 \cdots a_m$ from left to right and have at time t only direct access to the current symbol a_t. Moreover, at each time instant t the algorithm is required to compute a value that depends on the last n symbols. The value n is called the *window size* and the last n symbols form the *active window* at time t. A general goal in the area of sliding window algorithms is to avoid the explicit storage of the window content (which requires $\Omega(n)$ bits), and, instead, to work in considerably smaller space, e.g., polylogarithmic space in the window size n. An introduction into the sliding window model can be found in [1, Chapter 8].

In our recent papers [3, 4] we initiated the study of sliding window algorithms for regular languages. In general, a sliding window algorithm for a language $L \subseteq \Sigma^*$ decides, at every time instant, whether the word in the active window belongs to L. In [4] we proved that for every regular language L the optimal space bound for a sliding window algorithm for L is either constant, logarithmic or linear in the window size. In [3] we also gave several characterizations for the three space classes: A regular language has a sliding window algorithm with space bound $\mathcal{O}(\log n)$ (resp., $\mathcal{O}(1)$) if and only if it belongs to the Boolean closure of regular left ideals and regular length languages (resp., the Boolean closure of suffix-testable languages and regular length languages); see [3] for the formal definition of these language classes.

In this paper we investigate to which extent the results from [3, 4] can be generalized to context-free languages. Our first main result (Theorem 2) states that if L is a context-free language that has a sliding window algorithm with space bound $\log_2(n) - \omega(1)$ (recall that $f(n) \in \omega(1)$ iff $\forall c > 0\, \exists m\, \forall n \geq m : f(n) \geq c$) then L must be regular. By the results from [3, 4] this implies that L has a constant space sliding window algorithm and is a Boolean combination of suffix-testable languages and regular length languages. Our proof uses a variant of the classical pumping lemma. The crucial observation is that taking a reversed Greibach normal form grammar for G, we can ensure that pumping in a word of length n does not affect a suffix of length $o(n)$.

Theorem 2 shows that, analogously to regular languages, there is a gap between $\mathcal{O}(1)$ and $\mathcal{O}(\log n)$ in the space complexity spectrum for context-free languages. This leads to the question whether there is also a gap between $\mathcal{O}(\log n)$ and $\mathcal{O}(n)$ (as it is the case for regular languages). We answer this question negatively. For this we construct from a linear bounded automaton (LBA) a context-free language, whose sliding window space complexity is related to the time complexity of the LBA in a certain way. The precise technical statement can be found in Theorem 9. From this result we obtain for every $c \in \mathbb{N}$ a context-free language, whose optimal sliding window algorithm uses space $\mathcal{O}(n^{1/c})$ (Theorem 10).

The context-free languages from the proof of Theorem 9 are non-deterministic. They are obtained by taking the complement of all accepting computations of an LBA on an input from a^* (as usual, a computation is encoded by a sequence of configuration words). These complements are context-free since one can guess errors, but they are not deterministic context-free. This leads to the question whether there exist deterministic context-free languages for which the optimal sliding window algorithm has space complexity $o(n) \setminus \mathcal{O}(\log n)$. We answer this question positively by constructing a deterministic one-counter language whose optimal sliding window algorithm uses space $\mathcal{O}((\log n)^2)$ (Theorem 15).

The results from Theorem 10 and 15 are also shown for a more general sliding window model, which is known as the variable-size model in the literature. In the sliding window model discussed so far, the window size is fixed and for every window size there exists a streaming algorithm. In contrast, in the variable-size model, there is a single streaming algorithm and the window can grow and shrink. In other words, the arrival of new symbols and expiration of old symbols can happen independently. A formal definition can be found in Section 2. The space complexity of a variable-size streaming algorithm is measured with respect to the maximal window size seen in the past. In [4] it was shown that analogously to the fixed-size model, the space complexity of a regular language with respect to the variable-size model is either constant, logarithmic, or linear. Moreover, a regular language has space complexity $\mathcal{O}(\log n)$ in the variable-size model if and only if it has space complexity $\mathcal{O}(\log n)$ in the fixed-size model (on the other hand only trivial languages have constant space complexity in the variable-size model). Corollary 13 states that there exists a deterministic one-counter language whose optimal variable-size sliding window algorithm uses space $\Theta((\log n)^2)$.

Finally, we prove that our results for deterministic one-counter languages can be also shown for the reversals of the latter (i.e., for languages that can be accepted by a deterministic one-counter automaton that works from right to left). This is not obvious, since the reversal of a deterministic context-free language is in general not deterministic context-free. Moreover, the arguments for our space trichotomy result for regular languages [3, 4] mainly use a DFA for the reverse language, hence one might think that these arguments extend to reversals of deterministic context-free languages.

2 Preliminaries

For a function $f : \mathbb{N} \to \mathbb{N}$, we use the standard Landau notations $\mathcal{O}(f)$, $\Omega(f)$, $o(f)$ and $\Theta(f)$.

We assume that the reader is familiar with the basic notions of formal languages, in particular regular languages, see e.g. [7] for more details. Let Σ be a finite alphabet of symbols. With ε we denote the empty word. For a word $w = a_1 \cdots a_m \in \Sigma^*$ of length $|w| = m$ we define $w[i] = a_i$ and $w[i : j] = a_i \cdots a_j$ if $i \leq j$ and $w[i : j] = \varepsilon$ if $i > j$. We define $w[i :] = w[i : m]$ and $w[: j] = [1 : j]$. Let $\Sigma^n = \{w \in \Sigma^* : |w| = n\}$, $\Sigma^{\leq n} = \{w \in \Sigma^* : |w| \leq n\}$, and $\Sigma^{\geq n} = \{w \in \Sigma^* : |w| \geq n\}$. A word $v \in \Sigma^*$ is a *prefix* (resp., *suffix*) of the word w if there exists a word $u \in \Sigma^*$ such that $w = vu$ (resp., $w = uv$). With $\mathrm{prefix}(w)$ we denote the set of all prefixes of w. For a word $w = a_1 a_2 \cdots a_m$ let $\mathrm{rev}(w) = a_m \cdots a_2 a_1$ denotes the word w read from right to left.

2.1 Automata and streaming algorithms

We use standard definitions from automata theory. A *deterministic automaton* is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, where Q is a possibly infinite set of states, Σ is an alphabet, $q_0 \in Q$ is the initial states, $\delta \colon Q \times \Sigma \to Q$ is the transition relation, and F is the set of final states. The transition function δ is extended to a function $\delta \colon Q \times \Sigma^* \to Q$ in the usual way and we set $\mathcal{A}(x) = \delta(q_0, x)$ for all $x \in \Sigma^*$. The language accepted from a state $q \in Q$ is denoted by $L(\mathcal{A}, q) = \{x \in \Sigma^* \mid \delta(q, x) \in F\}$ and the language accepted by $\mathcal{A}$ is defined by $L(\mathcal{A}) = L(\mathcal{A}, q_0)$. If Q is finite, then A is a *deterministic finite automaton* (DFA).

A data stream is a finite sequence of data values. We make the assumption that these data values are from a finite set Σ. Thus, a data stream is a finite word $w = a_1 a_2 \cdots a_m \in \Sigma^*$. A streaming algorithm reads the symbols of a data stream from left to right. At time instant t the algorithm has only access to the symbol a_t and the internal storage, which is encoded by a bit string. The goal of the streaming algorithm is to compute a certain function $f \colon \Sigma^* \to A$ into some domain A, which means that at time instant t the streaming algorithm outputs the value $f(a_1 a_2 \cdots a_t)$. In this paper, we only consider the Boolean case $A = \{0, 1\}$; in other words, the streaming algorithm tests membership in a fixed language. Thus, a *streaming algorithm* over Σ can be seen as a deterministic (possibly infinite) automaton $\mathcal{A} = (S, \Sigma, s_0, \delta, F)$. Furthermore, we abstract away from the actual computation and only analyze the space requirement, which in particular means that we encode the states of $\mathcal{A}$ by bit strings. We describe this encoding by an injective function $\mathrm{enc} \colon S \to \{0, 1\}^*$. The *space function* $\mathrm{space}(\mathcal{A}, \cdot) \colon \Sigma^* \to \mathbb{N}$ specifies the space used by $\mathcal{A}$ on a certain input: For $w \in \Sigma^*$ let $\mathrm{space}(\mathcal{A}, w) = \max\{|\mathrm{enc}(\mathcal{A}(u))| : u \in \mathrm{prefix}(w)\}$. We also say that $\mathcal{A}$ is a *streaming algorithm for* the accepted language $L(\mathcal{A})$.

2.2 Sliding window streaming models

In the above streaming model, the output value of the streaming algorithm at time t depends on the whole past $a_1 a_2 \cdots a_t$ of the data stream. However, in many practical applications one is only interested in the relevant part of the past. Two formalizations of "relevant past" can be found in the literature:

- Only the suffix of $a_1 a_2 \cdots a_t$ of length n is relevant. Here, n is a fixed constant. This streaming model is called the *fixed-size sliding window model*.
- The relevant suffix of $a_1 a_2 \cdots a_t$ is determined by an adversary. In this model, at every time instant the adversary can either remove the first symbol from the active window (expiration of a data value), or add a new symbol at the right end (arrival of a new data value). This streaming model is also called the *variable-size sliding window model*.

In the following we formally define these two models.

Fixed-size sliding windows. Given a word $w \in \Sigma^*$ of length m and a window size $n \geq 0$, we define $\mathrm{last}_n(w) \in \Sigma^n$ by

$$\mathrm{last}_n(w) = \begin{cases} w[m-n+1:] & \text{if } n \leq m, \\ a^{n-m}w, & \text{if } n > m, \end{cases}$$

which is called the *active window*. Here $a \in \Sigma$ is an arbitrary, but fixed, symbol, which fills the initial window. For a language L and $n \geq 0$ let $L_n = \{w \in \Sigma^* : \mathrm{last}_n(w) \in L\}$. A sequence $\mathcal{A} = (\mathcal{A}_n)_{n\geq 0}$ is a *fixed-size sliding window algorithm* for a language $L \subseteq \Sigma^*$ if for each n the $\mathcal{A}_n$ is a streaming algorithm for L_n. Its *space complexity* is the function $f_{\mathcal{A}} \colon \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ where $f_{\mathcal{A}}(n)$ is the maximum encoding length of a state in $\mathcal{A}_n$.

Note that for every language L and every n the language L_n is regular, which ensures that $\mathcal{A}_n$ can be chosen to be a DFA and hence $f_{\mathcal{A}}(n) < \infty$ for all $n \geq 0$. The trivial fixed-size sliding window algorithm for L is the sequence $\mathcal{B} = (\mathcal{B}_n)_{n\geq 0}$, where $\mathcal{B}_n$ is the DFA with state set Σ^n and transitions $au \xrightarrow{b} ub$ for $a, b \in \Sigma$, $u \in \Sigma^{n-1}$. States of $\mathcal{B}_n$ can be encoded with $\mathcal{O}(\log|\Sigma| \cdot n)$ bits. Let $\mathcal{A}_n$ be the minimal DFA for L_n and encode each state of $\mathcal{A}_n$ with at most $\lfloor \log_2(a_n) \rfloor$ bits, where a_n is the number of states of $\mathcal{A}_n$. Then $\mathcal{A} = (\mathcal{A}_n)_{n\geq 0}$ is an *optimal fixed-size sliding window algorithm* $\mathcal{A}$ for L. Finally, we define $F_L(n) = f_{\mathcal{A}}(n) = \lfloor \log_2(a_n) \rfloor$. Thus, F_L is the space complexity of an optimal fixed-size sliding window algorithm for L. Notice that F_L is not necessarily monotonic. For instance, for $L = \{au \colon u \in \{a,b\}^*, |u| \text{ odd}\}$ we have $F_L(2n) \in \Theta(n)$ and $F_L(2n+1) \in \mathcal{O}(1)$. The above trivial algorithm $\mathcal{B}$ yields $F_L(n) \in \mathcal{O}(n)$ for every language L.

Note that the fixed-size sliding window is a *non-uniform* model: for every window size we have a separate streaming algorithm and these algorithms do not have to follow a common pattern. Working with a non-uniform model makes lower bounds stronger. In contrast, the variable-size sliding window model that we discuss next is a uniform model in the sense that there is a single streaming algorithm that works for every window size.

Variable-size sliding windows. For an alphabet Σ we define the extended alphabet $\overline{\Sigma} = \Sigma \cup \{\downarrow\}$. In the variable-size model the *active window* $\mathrm{wnd}(u) \in \Sigma^*$ for a stream $u \in \overline{\Sigma}^*$ is defined by

- $\mathrm{wnd}(\varepsilon) = \varepsilon$
- $\mathrm{wnd}(ua) = \mathrm{wnd}(u)a$ for $a \in \Sigma$
- $\mathrm{wnd}(u\downarrow) = \varepsilon$ if $\mathrm{wnd}(u) = \varepsilon$
- $\mathrm{wnd}(u\downarrow) = v$ if $\mathrm{wnd}(u) = av$ for $a \in \Sigma$

A *variable-size sliding window algorithm* for a language $L \subseteq \Sigma^*$ is a streaming algorithm $\mathcal{A}$ for $\{w \in \overline{\Sigma}^* : \mathrm{wnd}(w) \in L\}$. Following [3], we define its *space complexity* as the function $v_{\mathcal{A}} \colon \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ mapping each window size n to the maximum number of bits used by $\mathcal{A}$ on inputs producing an active window of size at most n. Formally, it is the monotonic function $v_{\mathcal{A}}(n) = \max\{\mathrm{space}(\mathcal{A}, u) : u \in \overline{\Sigma}^*, |\mathrm{wnd}(v)| \leq n \text{ for all } v \in \mathrm{prefix}(u)\}$. By taking the minimal (possibly infinite) deterministic automaton for $\{w \in \overline{\Sigma}^* : \mathrm{wnd}(w) \in L\}$ and encoding states appropriately one can prove that there exists an optimal space bound:

▶ **Lemma 1** ([3]). *For every language $L \subseteq \Sigma^*$ there exists a variable-size sliding window algorithm $\mathcal{A}$ for L such that $v_{\mathcal{A}}(n) \leq v_{\mathcal{B}}(n)$ for every variable-size sliding window algorithm $\mathcal{B}$ for L and every n.*

We define $V_L(n) = v_{\mathcal{A}}(n)$, where $\mathcal{A}$ is a *space optimal variable-size sliding window algorithm* for L from Lemma 1. Since any algorithm in the variable-size model yields an algorithm in the fixed-size model, we have $F_L(n) \leq V_L(n)$.

3 Sliding windows over context-free languages: below logspace

The goal of this section is to prove the following result:

▶ **Theorem 2.** *If L is a context-free language with $F_L(n) \in \log_2(n) - \omega(1)$, then L is regular and $F_L(n) \in \mathcal{O}(1)$.*

We start with some definitions. A language $L \subseteq \Sigma^*$ is *k-suffix testable* if it is a finite Boolean combination of languages of the form $\Sigma^* w$ where $w \in \Sigma^{\leq k}$. An equivalent condition is: for all $x, y, z \in \Sigma^*$ with $|z| = k$ we have $xz \in L$ if and only if $yz \in L$. We call L *suffix testable* if it is k-suffix testable for some k. Note that every suffix testable language is regular. Let $f\colon \mathbb{N} \to \mathbb{N}$ be a function. A language $L \subseteq \Sigma^*$ is *f-suffix definable* if for all $n \in \mathbb{N}$ and words $u, v, w \in \Sigma^*$ such that $|uw| = |vw| = n$ and $|w| = f(n)$ we have $uw \in L$ if and only if $vw \in L$. Similarly, one defines prefix testable and f-prefix definable languages. A *length language* is a language $L \subseteq \Sigma^*$ such that for every $n \geq 0$, either $\Sigma^n \subseteq L$ or $L \cap \Sigma^n = \emptyset$. We prove Theorem 2 in two steps:

▶ **Theorem 3.** *Every language $L \subseteq \Sigma^*$ is $(2^{F_L(n)+1} - 1)$-suffix definable.*

▶ **Theorem 4.** *If a context-free language L is f-suffix definable for a function $f(n) \in o(n)$, then L is a finite Boolean combination of suffix testable languages and regular length languages.*

Combining Theorem 3 and 4 yields Theorem 2: If a context-free language L satisfies $F_L(n) \in \log_2(n) - \omega(1)$ then L is f-suffix definable for a function $f(n) \in o(n)$ by Theorem 3. Theorem 4 implies that L is a finite Boolean combination of suffix testable languages and regular length languages. Hence L is regular and $F_L(n) \in \mathcal{O}(1)$. The rest of this section is devoted to the proofs of Theorem 3 and 4.

3.1 Proof of Theorem 3

For two languages L_1 and L_2 we define their distance $d(L_1, L_2)$ as follows: If $L_1 = L_2$, then we set $d(L_1, L_2) = 0$, and otherwise $d(L_1, L_2) = \sup\{|u| : u \in L_1 \triangle L_2\} + 1$ where $L_1 \triangle L_2 = (L_1 \setminus L_2) \cup (L_2 \setminus L_1)$ denotes the symmetric difference of L_1 and L_2. Notice that $d(L_1, L_2) < \infty$ if and only if $L_1 \triangle L_2$ is finite. If $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ is a DFA, we define the distance between two states $p, q \in Q$ by $d(p, q) = d(L(\mathcal{A}, p), L(\mathcal{A}, q))$. We will use a result from [5, Lemma 1] stating that $d(p, q) < \infty$ implies that $d(p, q) \leq |Q|$.

▶ **Lemma 5.** *Let $L \subseteq \Sigma^*$ be regular and $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ be its minimal DFA. We have:*
- **(i)** *$d(p, q) \leq k$ if and only if $\delta(p, z) = \delta(q, z)$ for all $p, q \in Q$ and $z \in \Sigma^k$.*
- **(ii)** *L is k-suffix testable if and only if $d(p, q) \leq k$ for all $p, q \in Q$.*
- **(iii)** *If there exists $k \geq 0$ such that L is k-suffix testable, then L is $|Q|$-suffix testable.*

Proof. The proof of (i) is an easy induction: If $k = 0$, the statement is $d(p, q) = 0$ iff $p = q$, which is true because $\mathcal{A}$ is minimal. For the induction step, we have $d(p, q) \leq k + 1$ iff $d(\delta(p, a), \delta(q, a)) \leq k$ for all $a \in \Sigma$ iff $\delta(p, z) = \delta(q, z)$ for all $z \in \Sigma^{k+1}$.

For (ii), assume that L is k-suffix testable and consider two states $p = \mathcal{A}(x)$ and $q = \mathcal{A}(y)$. If $z \in L(\mathcal{A}, p) \triangle L(\mathcal{A}, q)$, then $|z| < k$ because $xz \in L$ iff $yz \notin L$ and L is k-suffix testable. Now assume that $d(p, q) \leq k$ for all $p, q \in Q$ and consider $x, y \in \Sigma^*$, $z \in \Sigma^k$. Since

$d(\mathcal{A}(x), \mathcal{A}(y)) \leq k$, (i) implies $\mathcal{A}(xz) = \mathcal{A}(yz)$, and in particular $xz \in L$ iff $yz \in L$. Therefore, L is k-suffix testable.

Point (iii) follows from (ii) and the above mentioned result from [5, Lemma 1]. ◀

Proof of Theorem 3. Let $n \geq 0$ and $L_n = \{w \in \Sigma^* : \text{last}_n(w) \in L\}$. Let $\mathcal{A}_n$ be the minimal DFA for L_n, which has at most $f(n) := 2^{F_L(n)+1} - 1$ states. Since $\text{last}_n(xy) = y$ for all $x \in \Sigma^*$ and $y \in \Sigma^n$, the language L_n is n-suffix testable. Therefore L_n is $f(n)$-suffix testable by Lemma 5(iii). This implies that L is f-suffix definable because for all words $u, v, w \in \Sigma^*$ such that $|uw| = |vw| = n$ and $|w| = f(n)$ we have $uw \in L$ iff $uw \in L_n$ iff $vw \in L_n$ iff $vw \in L$. ◀

3.2 Proof of Theorem 4

We prove the variant of Theorem 4 that talks about prefix-definability. This makes no difference, since the reversal of a context-free languages is again context-free. Also note that the requirement $f(n) \in o(n)$ in Theorem 4 cannot be relaxed: For every $k \geq 1$, the language $\{xay : x, y \in \{a, b\}^*, |x| = k|ay|\}$ is context-free and $\lceil n/(k+1) \rceil$-suffix definable but not even regular.

First, we show that in the proof of Theorem 4 we can restrict ourselves to functions f with the following property: A monotonic function $f\colon \mathbb{N} \to \mathbb{N}$ has the *increasing plateau property* if for every $k \geq 1$ there exists an n_0 such that for all $n \geq n_0$ we have: $f(n+k) - f(n) \leq 1$. Clearly, if f has the increasing plateau property then $f \in o(n)$.

▶ **Lemma 6.** *Let $f\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$. If $f(n) \in o(n)$ then there exists a monotonic function $g\colon \mathbb{N} \to \mathbb{N}$ with the increasing plateau property and such that $f(n) \leq g(n)$ for all $n \in \mathbb{N}$.*

Proof. For a linear function $g : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ of the form $g(x) = \alpha \cdot x + \beta$ we call α the *slope* of g. We will first define a sequence of natural numbers $n_1 < n_2 < n_3 \cdots$ such that f is bounded by a continuous piecewise linear function $h : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ that has slope $1/i$ on the interval $[n_i, n_{i+1}]$ and slope 0 on the interval $[0, n_1]$. Then we show that $g : \mathbb{N} \to \mathbb{N}$ with $g(n) = \lceil h(n) \rceil$ has the properties from the lemma.

First, for every $i \geq 1$ we define $n_i \in \mathbb{N}$ and a linear function $h_i : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ of slope $1/i$ such that: (i) $n_{i+1} > n_i$, (ii) for all natural numbers $n \geq n_i$ we have $f(n) \leq h_i(n)$, and (iii) $h_i(n_{i+1}) = h_{i+1}(n_{i+1})$.

Let $n_1 \geq 0$ be the smallest natural number such that $f(n) \leq n$ for $n \geq n_1$ and $f(n) \leq n_1$ for $n < n_1$. Clearly such an n_1 exists, as $f(n) \in o(n)$. Define h_1 by $h_1(x) = x$ for all $x \in \mathbb{R}_{\geq 0}$. Hence, we have $f(n) \leq h_1(n)$ for all $n \geq n_1$.

For the induction step, assume that n_i and the linear function $h_i : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ (of slope $1/i$) are defined such that $f(n) \leq h_i(n)$ for all $n \geq n_i$. Define the linear function $u_{i+1}(x) = h_i(n_i) + (x - n_i)/(i+1)$, which has a slope $1/(i+1)$ and $u_{i+1}(n_i) = h_i(n_i)$. Then there is a smallest natural n_{i+1} such that $n_{i+1} > n_i$ and $u_{i+1}(n) \geq f(n)$ for each $n \geq n_{i+1}$. This holds because $f(n) \in o(n)$, and hence for any constants $\alpha > 0, \beta \in \mathbb{R}$ we have $f(n) \leq \alpha \cdot n + \beta$ for large enough n. Take this n_{i+1} and define the function h_{i+1} by $h_{i+1}(x) = h_i(n_{i+1}) + (x - n_{i+1})/(i+1)$. It has slope $1/(i+1)$ and satisfies $h_{i+1}(n_{i+1}) = h_i(n_{i+1})$. Finally, for all $n \geq n_{i+1}$ we have

$$\begin{aligned} h_{i+1}(n) &= h_i(n_{i+1}) + (n - n_{i+1})/(i+1) \\ &= h_i(n_i) + (n_{i+1} - n_i)/i + (n - n_{i+1})/(i+1) \\ &\geq h_i(n_i) + (n_{i+1} - n_i)/(i+1) + (n - n_{i+1})/(i+1) \\ &= h_i(n_i) + (n - n_i)/(i+1) = u_{i+1}(n) \geq f(n). \end{aligned}$$

Hence, n_{i+1} and h_{i+1} have all the desired properties.

We now define the function $h : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$:

$$h(x) = \begin{cases} n_1 & \text{if } x \in [0, n_1] \\ h_i(x) & \text{if } x \in [n_i, n_{i+1}] \text{ for some } i \geq 1. \end{cases}$$

Since $h_i(n_{i+1}) = h_{i+1}(n_{i+1})$ and $h_1(n_1) = n_1$, h is uniquely defined. Finally, let $g(n) = \lceil h(n) \rceil$ for all $n \in \mathbb{N}$.

Since $f(n) \leq h_i(n)$ for all $n \geq n_i$ and $f(n) \leq f(n_1) \leq n_1$ for all $n \leq n_1$, we have $f(n) \leq h(n) \leq g(n)$ for all $n \in \mathbb{N}$. Moreover, h is clearly monotonic, which implies that g is monotonic, too. It remains to show that g has the increasing plateau property.

Let $k \geq 1$ and $n \geq n_k$. Since h is continuous and piecewise linear with slopes $\leq 1/k$ on $[n_k, \infty)$, we have $h(n+k) - h(n) \leq (n+k-n)/k = 1$. This implies $g(n+k) - g(n) \leq 1$. ◀

▶ **Lemma 7.** *Let L be a context-free language and $f : \mathbb{N} \to \mathbb{N} \setminus \{0\}$ be monotonic with $f(n) \in o(n)$. There are constants n_0 and $c > 0$ (only depending on L and f) such that the following hold for every $n \geq n_0$:*

- *$n \geq f(n) + c$ and*
- *for all words u, v with $uv \in L$, $|uv| = n$, $|u| = f(n)$, and $|v| = n - f(n)$, there exist words v', v'' with $|v'| = |v| - c$, $|v''| = |v| + c$, and $uv', uv'' \in L$.*

Proof. Consider the following variant of the pumping lemma for context-free languages (see also [7, Chapter 6.1]), which simultaneously considers all languages defined by various nonterminals of the grammar; it can be shown in the same way as the standard variant:

Given a context-free grammar G with set of nonterminals N and productions P, let L_A denote the language generated by the grammar G_A with productions P and the start nonterminal A. Then there exists a natural number c_1 depending only on G and not on A, such that if $w \in L_A$ and $|w| \geq c_1$, then w can be written as $w = w_1 w_2 w_3 w_4 w_5$ with: $w_1 w_2^k w_3 w_4^k w_5 \in L_A$ for every $k \geq 0$, $|w_2 w_3 w_4| \leq c_1$ and $|w_2 w_4| > 0$. In particular, the word $w_1 w_2^{1+c_1!/|w_2 w_4|} w_3 w_4^{1+c_1!/|w_2 w_4|} w_5$ of length $|w| + c_1!$ belongs to L_A.

Let G be a grammar for L in Greibach normal form, i.e., all productions are of the form $A \to aA_1 \cdots A_k$ for $k \geq 0$, nonterminals $A, A_1, \ldots, A_k$ and a terminal a (such a grammar exists for every context-free language); see also [7, Chapter 4.6]. Let r be the maximal length of the productions' right-hand sides in G, let N be the set of nonterminals, and let c_1 be the constant from the above pumping lemma for G. We can assume that $r \geq 2$, otherwise L is finite and the lemma holds. Define $c = c_1!$ and choose an n_0 such that for all $n \geq n_0$ the following three inequalities hold:

$$\frac{n}{f(n)} > 1 + (r-1) r^{2|N| \cdot (r c_1 |N| + 1)!} \tag{1}$$

$$\frac{n}{f(n)} > 1 + c_1 (r-1) \tag{2}$$

$$n > f(n) + c$$

As the right-hand sides are constant and $f(n) \in o(n)$, such an n_0 exists. Hence, for all $n \geq n_0$ the following two inequalities hold ((1) is equivalent to (3) and (2) is equivalent to (4)):

$$\log_r \left(\frac{n - f(n)}{f(n)(r-1)} \right) > 2|N|(r c_1 |N| + 1)! \tag{3}$$

$$\frac{n - f(n)}{f(n)(r-1)} > c_1 \tag{4}$$

Consider a string uv of length $n \geq n_0$ generated by G, where $|u| = f(n)$. Fix a leftmost derivation of uv and consider the first moment, at which the current sentential form has u as a prefix. This happens after $|u| = f(n)$ derivation steps since G is in Greibach normal form. Apart from the prefix u, the rest of the sentential form has length at most $1 + f(n)(r-2) \leq f(n)(r-1)$ and it derives the word v of length $n - f(n)$. So one of the nonterminals in the sentential form, say A, generates a word x with

$$|x| \geq \frac{n - f(n)}{f(n)(r-1)} \overset{(4)}{>} c_1 \ . \tag{5}$$

The further analysis splits into several cases depending on the claim we want to prove.

We first show the second claim of the lemma, that there exists v'' such that $|v''| = |v| + c$ and $uv'' \in L(G)$. Since $|x| \geq c_1$, we can apply the pumping lemma and replace in the derivation of uv the word x by a word of length $|x| + c_1! = |x| + c$. The resulting derivation yields a word uv'' with $|v''| = |v| + c$, as claimed.

So let us now prove that there is v' such that $uv' \in L(G)$, where $|v'| = |v| - c$. Again, consider the nonterminal A that generates a string x satisfying (5). Since the length of each right-hand side is at most r, there is a path Π in the derivation tree of length at least

$$\log_r \left(\frac{n - f(n)}{f(n)(r-1)} \right) > 2|N| \cdot (rc_1|N| + 1)! \ ,$$

where the estimation follows from (3). We are going to color some nodes on the path Π black or grey: if a node v on Π has a child that does not belong to Π and derives a string of length at least c_1, then we color v black. Then, as long as there are two uncolored nodes v, v' on Π (v above v') such that (i) v and v' are labelled with the same nonterminal, (ii) the path from v to v' has length at most $|N|$, and (iii) does not contain a black node, then we color v black and v' grey.

There can be at most $|N|$ consecutive nodes on the path that are not colored and there are at least as many black nodes as grey nodes. Thus, the number of black nodes is at least

$$\left\lfloor \frac{1}{2} \left\lfloor \frac{2|N| \cdot (rc_1|N| + 1)!}{|N|} \right\rfloor \right\rfloor = (rc_1|N| + 1)!$$

For each black node we can shorten the derivation such that the derived word is shorter by at least 1 and at most $rc_1|N|$ without affecting other colored nodes:

- For the first type of black nodes this follows directly from the pumping lemma. Note that we can apply the pumping lemma to a subtree that is disjoint from Π.
- For the second kind of black nodes, let v and v' be the corresponding nodes colored black and grey, respectively. We can delete the subtree rooted in v and replace it with the one rooted in v'. The length of the path is $\leq |N|$, the arity of the rules $\leq r$ and each deleted nonterminal derived a string of length $\leq c_1$ (otherwise it would be black).

So for some $m \in \{1, 2, \ldots, rc_1|N|\}$ there are $\frac{(rc_1|N|+1)!}{rc_1|N|} > (rc_1|N|)!$ different possibilities to shorten the derived word by m letters. We choose $(rc_1|N|)!/m$ of them so that the word is shortened by $(rc_1|N|)!$ letters. Thus we showed that there exists v' such that $uv' \in L(G)$ and $|uv'| = n - (rc_1|N|)!$. As $c = c_1!$ divides $(rc_1|N|)!$, by applying $((rc_1|N|)!/c - 1)$ times the already proved second claim of the lemma we can first obtain a word $uz \in L(G)$ such that $|uz| = n + (rc_1|N|)! - c$ and then use the argument above to obtain a word $uv' \in L(G)$ such that $|uv'| = |uz| - (rc_1|N|)! = |uv| - c$. Here, we use monotonicity of f, which ensures that the prefix u is not touched when using the above argument for the longer word uz. ◀

▶ **Lemma 8.** *Let L be a context-free language that is f-prefix definable for a function $f(n) \in o(n)$. Then there exists a constant α such that for every word u of length α and all words v, w with $|v| = |w|$ we have $uv \in L$ if and only if $uw \in L$.*

Proof. By Lemma 6 there exists a monotonic function $g(n) \in o(n)$ having the increasing plateau property and such that $f(n) \leq g(n)$ for all $n \geq 0$. Hence, L is still g-prefix definable. Moreover, let $f' \in o(n)$ be defined by $f'(n) = g(n) + 1$ for all n. Take the constants n_0 and c from Lemma 7 for L and f' (instead of f). Choose m such that (i) $m \geq n_0 + c$ and (ii) $g(n) - g(n-c) \leq 1$ for all $n \geq m$, which is possible by the increasing plateau property. We take $\alpha = g(m)$. Heading for a contradiction, let us take words u, v, w such that $|u| = \alpha$, $|v| = |w|$, $uv \in L$ and $uw \notin L$. We can assume that $|v| = |w|$ is minimal with these properties. Let $n = |uv| = |uw|$ in the following. We now distinguish two cases.

Case 1. $n \leq m$, which implies $g(n) \leq g(m) = |u|$. Hence, uv and uw have the same prefix of length $g(n)$. Since L is g-prefix definable, we have $uv \in L$ iff $uw \in L$, which is a contradiction.

Case 2. $n > m$, and thus $n > n_0 + c$ and $g(n) \geq g(m) = |u|$. Since $n - g(m) \geq n - g(n) = n - f'(n) + 1 > c > 0$, we can write $v = v_1 a v_2$ and $w = w_1 b w_2$ such that $a, b \in \Sigma$ and $|uv_1| = |uw_1| = g(n)$. Thus, $|uv_1 a| = |uw_1 b| = f'(n)$. By Lemma 7 there exists a word v_2' with $|v_2'| = |v_2| - c$ and $uv_1 a v_2' \in L$. Take any word w_2' of length $|w_2'| = |w_2| - c$. By the length-minimality of v and w we must have $uw_1 b w_2' \in L$ (note that $c > 0$). Note that $|uw_1 b w_2'| = |uw| - c = n - c > n_0$. Therefore, we can apply Lemma 7 to the word $uw_1 b w_2'$. Note that $g(n) - g(n-c) \leq 1$ since $n \geq m$. Thus, $f'(n-c) = g(n-c) + 1 \geq g(n)$ and the prefix of $uw_1 b w_2'$ of length $f'(n-c)$ starts with uw_1. We can conclude with Lemma 7 that there is a word w_2'' such that $uw_1 w_2'' \in L$ and $|uw_1 w_2''| = n$. But since $|uw_1| = g(n)$ and $|uw_1 w_2''| = n$, the g-prefix definability of L implies that $uw_1 y \in L$ for all words y of length $n - g(n)$. In particular, we get $uw_1 b w_2 = uw \in L$, which is a contradiction. ◀

We can now prove (the prefix version of) Theorem 4:

Proof of Theorem 4. Let L be a f-prefix definable context-free language with $f(n) \in o(n)$. Let α be the constant from Lemma 8. For every word u of length α let $L_u = \{w \colon uw \in L\}$. Each of these finitely many languages is context-free and by Lemma 8 it is a length language. It is a direct consequence of Parikh's theorem [8] (or the fact that every unary context-free language is regular) that a context-free length language is regular. Hence, every L_u (for $|u| = \alpha$) is a regular length language. We can now decompose L as follows:

$$L = (L \cap \Sigma^{<\alpha}) \cup \bigcup_{u \in \Sigma^\alpha} uL_u = (L \cap \Sigma^{<\alpha}) \cup \bigcup_{u \in \Sigma^\alpha} (u\Sigma^* \cap \Sigma^\alpha L_u).$$

Since L_u is a regular length language, also $\Sigma^\alpha L_u$ is a regular length language. Moreover, $u\Sigma^*$ is prefix testable. Finally, every finite language (and hence $L \cap \Sigma^{<\alpha}$) is a finite Boolean combination of prefix testable languages. This shows the theorem. ◀

4 Sliding windows over context-free languages: above logspace

In this section, we show that the trichotomy result for regular languages [4] does not carry over the context-free languages. More precisely, we show that for every natural number $c \geq 1$ there exists a one-counter language L_c such that $F_{L_c}(n) \in \mathcal{O}(n^{1/c}) \setminus o(n^{1/c})$ and $V_{L_c}(n) \in \Theta(n^{1/c})$. Recall that a one-counter language is a language that can be accepted by a nondeterministic pushdown automaton with a singleton pushdown alphabet (a so called

one-counter automaton). Also recall that a linear bounded automaton (LBA for short) is a Turing machine that only uses the space that is occupied by the input word; see also [7, Chapter 9.3]. We first show the following technical result:

▶ **Theorem 9.** *Let $t(k)$ be a monotonically increasing function and M be an LBA which halts on input a^k after exactly $t(k)$ steps. Let $f(n)$ be the function with*[1]

$$f(n) = \begin{cases} k & \text{if } n = k(t(k)+3) \text{ for some } k \\ 0 & \text{else} \end{cases}$$

and let $g(n) = \max\{f(m) \colon m \leq n\}$. There is a one-counter language L such that $F_L(n) \in \Theta(f(n))$ and $V_L(n) \in \Theta(g(n))$.

Proof. Let Γ be the tape alphabet of M and Q the set of states of M. A configuration of M is encoded by a word from $\Gamma^*(Q \times \Gamma)\Gamma^*$ over the alphabet $\Delta := \Gamma \cup (Q \times \Gamma)$. A computation of M on an input a^k ($k \geq 1$) is a sequence of configurations $c_0 \vdash_M \cdots \vdash_M c_{t(k)}$ where $|c_i| = k$ for all $1 \leq i \leq t(k)$, $c_0 = (q_0, a)a^{k-1}$ is the start configuration on input a^k, every c_{i+1} is obtained from c_i by applying a transition of M for $0 \leq i \leq t(k)-1$, and $c_{t(k)}$ is an accepting computation. Let $\Delta' = \{x' \mid x \in \Delta\}$ be a disjoint copy of Δ and define w' for a word $w \in \Delta^*$ by applying the homomorphism $x \mapsto x'$ ($x \in \Delta$) to w. Finally, let K be the set of all words

$$c_0 \operatorname{rev}(c_1)' \, c_2 \operatorname{rev}(c_3)' \cdots c_{t(k)} \, s \operatorname{rev}(s) \text{ or} \tag{6}$$

$$c_0 \operatorname{rev}(c_1)' \, c_2 \operatorname{rev}(c_3)' \cdots \operatorname{rev}(c_{t(k)})' \, s \operatorname{rev}(s) \tag{7}$$

such that $k \geq 1$, $c_0 \vdash_M \cdots \vdash_M c_{t(k)}$ is a computation on input a^k, $s \in \{0,1\}^*$ is an arbitrary word of length k (0 and 1 are arbitrary symbols not in $\Delta \cup \Delta'$), and $t(k)$ even (resp., odd) in case (6) (resp., (7)). Notice that the words in (6) and (7) have length $k(t(k)+3)$. We can assume that M never goes back to the initial state q_0. This ensures that every word has at most one non-empty suffix that is a prefix of a word from K.

For the language L from the theorem, we take the complement of K. It is not hard to see that L can be recognized by a nondeterministic one-counter automaton by guessing an error in the input word w. Possible errors are the following, where we call a block of w a maximal factor from $\Delta^+ \cup (\Delta')^+ \cup \{0,1\}^+$ in w, m is the number of blocks of w and u_i denotes the i-th block of w for $1 \leq i \leq m$:

1. $m < 2$,
2. u_1 is not an initial configuration, i.e., of the form $(q_0, a)a^{k-1}$ for some $k \geq 1$,
3. for some odd $i < m$, u_i is not a configuration,
4. for some even $i < m$, u_i is not of the form c' for a configuration c,
5. u_{m-1} is not an accepting confiuration,
6. there exists $1 \leq i < m-1$ with $|u_i| \neq |u_{i+1}|$,
7. $|u_m| \neq 2|u_{m-1}|$,
8. there exists $1 \leq i < m-1$ odd such that $u_i \vdash_M \operatorname{rev}(u)$ does not hold for $u' = u_{i+1}$,
9. there exists $1 \leq i < m-1$ even such that $\operatorname{rev}(u) \vdash_M u_{i+1}$ does not hold for $u' = u_i$,
10. u_m is not a palindrome over the alphabet $\{0,1\}^*$.

The conditions in points 1–5 are regular. Points 6–10 can be checked with a single counter.

The upper bound in the theorem has to be shown for the variable-size model. Since $F_K(n) = F_L(n)$ and $V_K(n) = V_L(n)$, it is enough to show the bounds for the language K. Let us first present a variable-size streaming algorithm with space complexity $\mathcal{O}(g(n))$. Assume that w is the active window. The algorithm stores the following data n, i, t, k, ℓ, s:

[1] Since $t(k)$ is monotonically increasing, the number k in the first case is unique.

- $n = |w|$ is the length of the active window.
- i is the smallest position x such that $w[x:]$ is a prefix of a word from K. If this prefix is empty, then $i = n + 1$.
- t is the number of blocks in $w[i:]$ minus 1 (where i is from the previous point); this tells us the number of computation steps that M has executed.
- k is the largest number y such that $w[i:]$ starts with $(q_0, a)a^{y-1}$; hence, k tells us the input length.
- In case $1 \leq t \leq t(k)$, ℓ is the length of the last block of $w[i:]$ (if $t = 0$ or $t = t(k) + 1$ we store some dummy value in ℓ).
- In case $t = t(k) + 1$, s is the maximal suffix of $w[i:]$ from $\{0,1\}^*$. If the length of this suffix exceeds k then s stores only its prefix of length k.

It is easy to see that these variables can be updated. The main observation is that in case $1 \leq t \leq t(k)$ and $\ell < k$ then the algorithm internally simulates M for t steps on input a^k. In this way, the algorithm can check whether the arriving symbol is the right one, namely the (possibly primed) $(\ell+1)$-th symbol of the configuration reached after t steps on input a^k. In this case, the algorithm sets $\ell := \ell + 1$, otherwise the algorithm sets $i := n + 1$. If t is set to $t(k) + 1$ then the algorithm starts to accumulate the window suffix $s \in \{0,1\}^*$ up to length k. If s has length k then the next k arriving symbols are compared in reversed order with s. If a match is obtained, the algorithm accepts if $i = 1$ at the same time.

Let us now compute the space complexity of the algorithm. The numbers n, i, t, k and ℓ need $\mathcal{O}(\log n)$ bits. Recall that s has maximal length k. But we only store symbols in s if $n \geq k(t(k) + 1) \geq \lfloor k/3 \rfloor (t(\lfloor k/3 \rfloor) + 3)$, since the window must already contain a complete computation on input a^k before s becomes non-empty. We get $\lfloor k/3 \rfloor = f(\lfloor k/3 \rfloor (t(\lfloor k/3 \rfloor) + 3)) \leq g(n)$, i.e., $k \leq 3g(n) + 3$. Finally, since $g(n)$ is the maximal value k such that $k(t(k) + 3) \leq n$ and $t(k) \in 2^{\mathcal{O}(k)}$, we get $g(n) \in \Omega(\log n)$. This shows that the algorithm works in space $\mathcal{O}(g(n))$.

To show that $F_K(n) \in \mathcal{O}(f(n))$ we can argue similarly. Of course, in the fixed-size model, we do not have to store the window size n. If the window size n is not of the form $k(t(k)+3)$ for some k then the algorithm always rejects and no space at all is needed. Otherwise, since $t(k)$ is monotonically increasing, there is a unique k with $n = k(t(k) + 3)$.

Finally, we show that $F_K(n) \geq f(n)$ for all n, which implies that $V_K(n) \geq g(n)$ for all n since $V_K(n) \geq F_K(n)$ and $V_K(n)$ is monotonic. It suffices to consider a window size $n = k(t(k) + 3)$ for some k, as otherwise $f(n) = 0$. Hence, $f(n) = k$. Moreover, consider an accepting computation $c_0 \vdash_M c_1 \vdash_M \cdots \vdash_M c_{t(k)}$ where $|c_0| = \cdots = |c_{t(k)}| = k$. Let us assume that k is even; the case that k is odd is analogous. Now consider the 2^k many distinct words $w(s) := 0^k\, c_0\, \mathrm{rev}(c_1)'\, c_2\, \mathrm{rev}(c_3)' \cdots c_{t(k)}\, s$ for $s \in \{0,1\}^k$. The length of these words is $n = k(t(k) + 3)$, which is the window size.

Consider now the minimal DFA $\mathcal{A}_n$ for the language K_n, and let r be the number of states of $\mathcal{A}_n$ (hence, $F_K(n) = \lfloor \log_2 r \rfloor$). We claim that $\mathcal{A}_n(w(s)) \neq \mathcal{A}_n(w(u))$ for all $s, u \in \{0,1\}^k$ with $s \neq u$. To see this, assume that $\mathcal{A}_n(w(s)) = \mathcal{A}_n(w(u))$ for $s, u \in \{0,1\}^k$ with $s \neq u$. Hence, $\mathcal{A}_n(w(s)\,\mathrm{rev}(s)) = \mathcal{A}_n(w(u)\,\mathrm{rev}(s))$. On the other hand, the above definition of $w(s)$ and $w(u)$ implies $w(s)\,\mathrm{rev}(s) \in K$ and $w(u)\,\mathrm{rev}(s) \notin K$, which yields a contradiction. We get $r \geq 2^k$, and thus $F_K(n) \geq k = f(n)$. ◀

Theorem 9 yields a quite dense spectrum of space complexity functions for context-free languages. We only prove the existence of context-free languages with sliding-window space complexity $n^{1/c}$ for $c \in \mathbb{N}$, $c \geq 1$:

▶ **Theorem 10.** *For every $c \in \mathbb{N}$, $c \geq 1$, there exists a one-counter language L_c such that $F_{L_c}(n) \in \mathcal{O}(n^{1/c}) \setminus o(n^{1/c})$ and $V_{L_c}(n) \in \Theta(n^{1/c})$.*

Proof. One can easily construct a deterministic LBA M that on input a^k terminates after exactly k^{c-1} steps. For instance, an LBA that terminates after exactly k^2 steps makes k phases, where in each phase the read-write head moves from the left input end to the right end or vice versa and thereby replaces the first a that is seen on the tape by a b-symbol. This construction can be iterated to obtain the above LBA M for an arbitrary k. The mapping $f(n)$ from Theorem 9 then satisfies $f(k(t(k)+3)) = f(k(k^{c-1}+3)) = f(k^c+3k) = k$ and $f(n) = 0$ if n is not of the form $k^c + 3k$ for some k. This implies $f(n) \in \mathcal{O}(n^{1/c}) \setminus o(n^{1/c})$ and $g(n) \in \Theta(n^{1/c})$ for the mapping $g(n)$ from Theorem 9. Hence, by Theorem 9 there is a one-counter language L_c with the properties stated in the theorem. ◀

To fully exploit Theorem 9 one would have to analyze the spectrum of time complexity functions of linear bounded automata. We are not aware of specific results in this direction.

5 Sliding windows over deterministic one-counter languages

The context-free language L_c from Theorem 10 is not deterministic context-free and it is open whether the same result can be obtained for deterministic context-free languages. In this section we exhibit a deterministic one-turn one-counter language with space complexity $\Theta((\log n)^2)$ in the variable-size (resp., fixed-size) model. A *t-turn pushdown automaton* has the property that in any accepting run there are at most t alternations between push and pop operations [6].

We start with the variable-size model. A maximal factor β in a word $w \in \{a,b\}^*$ of the form $\beta = ab^i$ is called a *block* of length $i+1$ in w (this notion is not related to the blocks used in the proof of Theorem 9). Define the language

$$L = \{ab^k av : k \geq 0,\, v \in \{a,b\}^{\leq k}\} \cup ab^*, \tag{8}$$

which is recognized by a deterministic one-turn one-counter automaton. Put differently, L contains those words $w \in \{a,b\}^*$ which begin with a block of length $\geq |w|/2$.

▶ **Lemma 11.** *We have $V_L(n) \in \mathcal{O}((\log n)^2)$.*

Proof. Any word $w \in \{a,b\}^*$ can be uniquely factorized as $w = b^s\beta_m\beta_{m-1}\cdots\beta_2\beta_1$ where $s, m \geq 0$ and each β_i is a block. A block β_i is *relevant* if it is at least as long as the remaining suffix, i.e. $|\beta_i| \geq \sum_{j=1}^{i-1} |\beta_j|$. For an active window $w \in \{a,b\}^*$ our variable-size algorithm maintains the window size and for each relevant block β_i its starting position and its length. If the first symbol in the window expires, every relevant block stays relevant (and the starting position is decremented) with the possible exception of a relevant block with starting position 1, which is removed. If an a arrives, we create a new relevant block of length 1. If a b arrives, we prolong the rightmost relevant block (which is also rightmost among all blocks) by 1. Furthermore, using this information we can determine whether $w \in L$: This is the case if and only if the leftmost relevant block starts at the first position and its length is at least $n/2$ where n is the current window size.

To show that the space complexity of the algorithm is $\mathcal{O}((\log n)^2)$, it suffices to show that each word $w \in \{a,b\}^n$ has $\mathcal{O}(\log n)$ relevant blocks. Let $\gamma_k, \gamma_{k-1}, \ldots, \gamma_2, \gamma_1$ be the sequence of relevant blocks in w. Then we know that $|\gamma_i| \geq \sum_{j=1}^{i-1} |\gamma_j|$ for all $1 \leq i \leq k$. Inductively, we show that $|\gamma_i| \geq 2^{i-2}|\gamma_1|$ for all $2 \leq i \leq k$. This is immediate for $i = 2$ and for the induction step, observe that $|\gamma_i| \geq |\gamma_1| + \sum_{j=2}^{i-1} |\gamma_j| \geq |\gamma_1| + |\gamma_1| \sum_{j=2}^{i-1} 2^{j-2} = 2^{i-2}|\gamma_1|$ for all $i \geq 3$. This proves $k = \mathcal{O}(\log n)$, which concludes the proof. ◀

▶ **Lemma 12.** *We have* $V_L(n) \in \Omega((\log n)^2)$.

Proof. For each $k \geq 0$ we define *arrangements* of length 3^k: The word a is the only arrangement of length $3^0 = 1$. An arrangement of length 3^{k+1} is any word of the form $ub^{3^k}v$ where $u, v \in \{a, b\}^{3^k}$, u begins with a and has at most one other a-symbol and v is any arrangement of length 3^k. Notice that an arrangement $ub^{3^k}v$ contains one or two blocks in the factor ub^{3^k}, one of which is relevant. If $\alpha_1 \neq \alpha_2$ are distinct arrangements of length 3^k, consider the maximal common suffix α_3 of α_1 and α_2 which is again an arrangement. Consider the suffixes of α_1, α_2 of length $3|\alpha_3|$. By the construction, these suffixes are also arrangements. Hence, their "middle parts" consist solely of b's, so they have the common suffix $b^{|\alpha_3|}\alpha_3$. Since α_1 and α_2 differ, there exists a number $\ell \geq |\alpha_3|$ such that α_1 has the suffix $ab^\ell\alpha_3$ and α_2 has the suffix $b^{\ell+1}\alpha_3$, or vice versa.

Now consider a variable-size sliding window algorithm for L. We claim that the algorithm can distinguish any two distinct arrangements $\alpha_1 \neq \alpha_2$ of length 3^k. Consider two instances of the algorithm, where the active windows are α_1 and α_2, respectively. By performing a suitable number of $\downarrow$-operations the two windows contain the words $ab^\ell\alpha_3$ and $b^{\ell+1}\alpha_3$, respectively. Since $|\alpha_3| \leq \ell$, we have $ab^\ell\alpha_3 \in L$ and $b^{\ell+1}\alpha_3 \notin L$.

It is easy to see that the number n_k of arrangements of length 3^k is exactly $\prod_{i=0}^{k-1} 3^i$: to construct an arrangement of length 3^k, note that among its first 3^{k-1} letters the first one is a and there is at most one further a. So, there are 3^{k-1} choices for the prefix of length 3^{k-1}. The next 3^{k-1} letters are fixed, and then one of n_{k-1} many arrangements of length 3^{k-1} follows. Thus $n_k = 3^{k-1}n_{k-1}$ and $n_0 = 1$, which yields the claim. Note that $\log_3(n_k) = \sum_{i=0}^{k-1} i = \Theta(k^2)$. Therefore, the algorithm needs $\Omega((\log n)^2)$ bits of space. ◀

▶ **Corollary 13.** *There exists a deterministic one-turn one-counter language* L *such that* $V_L(n) = \Theta((\log n)^2)$.

The language L from (8) is an example of a language where the space complexity in the fixed-size model is strictly below the space complexity in the variable-size model:

▶ **Lemma 14.** *We have* $F_L(n) \in \mathcal{O}(\log n)$.

Proof. Let $n \geq 0$ be the window size. For the active window we store (i) the starting position of the leftmost block of length at least $n/2$ (if such a block does not exist we set a special flag) and (ii) the length of the unique suffix from ab^* (again, a flag is set if the window content is in b^*). This information can be stored with $\mathcal{O}(\log n)$ bits and it can be updated at each step. Moreover, the active window belongs to L if the leftmost block of length at least $n/2$ starts at position 1. ◀

We now prove the variant of Corollary 13 for the fixed-size model: For the language L from (8) let $K = Lc^*$, which is a deterministic one-turn one-counter language.

▶ **Theorem 15.** *We have* $F_K(n) = \Theta((\log n)^2)$.

Proof. Let n be the window size. Consider the maximal suffix of the active window which has the form vc^i where $v \in \{a, b\}^*$. Using $\mathcal{O}(\log n)$ bits we maintain the starting position of that suffix and the length $|v|$. Furthermore, we maintain the same data structure as in the proof of Lemma 11 for the word $v \in \{a, b\}^*$. The algorithm accepts iff v begins at the first position, the leftmost relevant block also starts at the first position and has length at least $|v|/2$. In total, the space complexity is bounded by $\mathcal{O}((\log n)^2)$.

The proof for the lower bound is similar to the proof of Lemma 12. Let k be maximal such that $3^k \leq n$. Then the number of bits required to encode an arrangement of length 3^k is $\Omega((\log n)^2)$. The rest of the proof follows the proof of Lemma 12; we only have to replace every $\downarrow$-operation by the insertion of a c-symbol. ◀

For the language L from (8) let $L' = \{\#^j \mathrm{rev}(u)v\$^i : u \in L,\, i \geq 0,\, v \in \{a,b\}^i,\, j \geq 1\}$. Its reversal $\mathrm{rev}(L') = \{\$^i v \mid i \geq 0,\, v \in \{a,b\}^i\}\, L\, \#^+$ is accepted by a deterministic one-counter three-turn automaton.

▶ **Theorem 16.** *We have* $V_{L'}(n) = \mathcal{O}((\log n)^2)$ *and* $F_{L'}(n) \notin o((\log n)^2)$.

Proof. We first exhibit a variable-size sliding window algorithm for L'. Of course, we maintain the window size n. For the active window consider its longest suffix of the form $\#^j w\$^i$ where $w \in \{a,b\}^*$ and $i,j \geq 0$. Using $\mathcal{O}(\log n)$ bits we can maintain the numbers i, j, the length $|w|$, and the maximal number k such that b^k is a suffix of w.

Furthermore, if $j \geq 1$ we maintain for each relevant block in $\mathrm{rev}(w)$ its starting position and its length, which requires $\mathcal{O}((\log n)^2)$ bits (see the proof of Lemma 11). Let us argue why this information can be maintained. Let n, i, j, k and w have the meaning from the previous paragraph. If j is set from 0 to 1, then w is empty and $\mathrm{rev}(w)$ contains no blocks. If $j \geq 1$ we can prolong w as long as the active window does not end with \$-symbols ($i = 0$). In this case, every time an a-symbol arrives, a new block in $\mathrm{rev}(w)$ is formed, which has length $k+1$. If it is not relevant, then it is immediately discarded. Also notice that when w is prolonged by a or b, all relevant blocks in $\mathrm{rev}(w)$ stay relevant. A $\downarrow$-operation only affects w if $j \in \{0,1\}$ and $n = j + |w| + i$. In this case j is set to zero, and we no longer have to store the relevant blocks of w.

It remains to show the lower bound. Let the window size n be of the form $2 \cdot 3^k$. Again we show that any fixed-size sliding window algorithm for L' must distinguish any two distinct arrangements. Let $\alpha_1 \neq \alpha_2$ be two arrangements of length 3^k. As shown in the proof of Lemma 12, there exists a number $0 \leq \ell < 3^k$ and an arrangement α_3 of length at most ℓ such that α_1 and α_2 have the suffixes $ab^\ell\alpha_3 \in L$ and $b^{\ell+1}\alpha_3 \notin L$, respectively (or vice versa). Without loss of generality, $\alpha_1 = v_1 ab^\ell \alpha_3$ and $\alpha_2 = v_2 b^{\ell+1}\alpha_3$ for some $v_1, v_2 \in \{a,b\}^*$. Both words v_1 and v_2 have length $r := 3^k - (\ell + 1 + |\alpha_3|)$. We have

$$\mathrm{last}_n(\#^{3^k}\mathrm{rev}(\alpha_1)\$^r) = \#^{3^k-r}\mathrm{rev}(ab^\ell\alpha_3)\,\mathrm{rev}(v_1)\$^r \in L' \quad \text{and}$$
$$\mathrm{last}_n(\#^{3^k}\mathrm{rev}(\alpha_2)\$^r) = \#^{3^k-r}\mathrm{rev}(b^{\ell+1}\alpha_3)\,\mathrm{rev}(v_2)\$^r \notin L'.$$

This shows that the algorithm must distinguish the words $\#^{3^k}\mathrm{rev}(\alpha_1)$ and $\#^{3^k}\mathrm{rev}(\alpha_2)$. Note that adding a \$ at the right end of the word removes the right-most symbol (a or b) in the factor which has to belong to $\mathrm{rev}(L)$ in order to have a word from L'. The rest of the proof follows the arguments from the proof of Lemma 12. ◀

6 Open problems

Our results lead to several open problems; in particular for deterministic context-free languages: Are there deterministic context-free languages where the optimal space bound (for the variable-size or the fixed-size model) is in $o(n) \cap \omega((\log n)^2)$?

An interesting subclass of the deterministic context-free languages are the visibly pushdown languages [2, 9], which are also known as input-driven languages. Visibly pushdown languages have better algorithmic properties than general deterministic context-free languages [2, 9]. Our deterministic context-free languages from Section 5 are not visibly pushdown languages. This leads to the question, whether our space trichotomy result for regular languages [4] extends to visibly pushdown languages (or at least visibly one-counter languages).

References

1 Charu C. Aggarwal. *Data Streams - Models and Algorithms*. Springer, 2007.

2 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, STOC 2004*, pages 202–211. ACM Press, 2004.

3 Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science, STACS 2018*, volume 96 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

4 Moses Ganardi, Danny Hucke, and Markus Lohrey. Querying regular languages over sliding windows. In *Proceedings of the 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016*, volume 65 of *LIPIcs*, pages 18:1–18:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

5 Paweł Gawrychowski and Artur Jeż. Hyper-minimisation made efficient. In *Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science, MFCS 2009*, volume 5734 of *Lecture Notes in Computer Science*, pages 356–368. Springer, 2009.

6 Seymour Ginsburg and Edwin H Spanier. Finite-turn pushdown automata. *SIAM Journal on Control*, 4(3):429–453, 1966.

7 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison–Wesley, Reading, MA, 1979.

8 Rohit Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.

9 Burchard von Braunmühl and Rutger Verbeek. Input-driven languages are recognized in $\log n$ space. In *Proceedings of the 1983 International FCT-Conference, FCT 1983*, volume 158 of *Lecture Notes in Computer Science*, pages 40–51. Springer, 1983.

Average Case Analysis of Leaf-Centric Binary Tree Sources

Louisa Seelbach Benkner
Universität Siegen, Germany
seelbach@eti.uni-siegen.de

Markus Lohrey
Universität Siegen, Germany
lohrey@eti.uni-siegen.de

Abstract

We study the average size of the minimal directed acyclic graph (DAG) with respect to so-called leaf-centric binary tree sources as studied by Zhang, Yang, and Kieffer. A leaf-centric binary tree source induces for every $n \geq 2$ a probability distribution on all binary trees with n leaves. We generalize a result shown by Flajolet, Gourdon, Martinez and Devroye according to which the average size of the minimal DAG of a binary tree that is produced by the binary search tree model is $\Theta(n/\log n)$.

2012 ACM Subject Classification Mathematics of computing → Enumeration

Keywords and phrases Directed acylic graphs, average case analysis, tree compression

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.16

Related Version A full version of the paper is available at [2], `http://arxiv.org/abs/1804.10396`.

1 Introduction

One of the most important and widely used compression methods for trees is to represent a tree by its minimal *directed acyclic graph*, shortly referred to as minimal DAG. The minimal DAG of a tree t is obtained by keeping for each subtree s of t only one isomorphic copy of s to which all edges leading to roots of s-copies are redirected. DAGs found applications in numerous areas of computer science; let us mention compiler construction [1, Chapter 6.1 and 8.5], unification [14], XML compression and querying [5, 9], and symbolic model-checking (binary decision diagrams) [4]. Recently, in information theory the average size of the minimal DAG with respect to a probability distribution turned out to be the key in order to obtain tree compressors whose average redundancy converges to zero [10, 16].

In this paper, we consider the problem of deriving asymptotic estimates for the average size of the minimal DAG of a randomly chosen binary tree of size n. So far, this problem has been analyzed mainly for two particular distributions: In [8], Flajolet, Sipala and Steyaert proved that the average size of the minimal DAG with respect to the uniform distribution on all binary trees of size n is asymptotically equal to $c \cdot n/\sqrt{\ln n}$, where c is the constant $2\sqrt{\ln(4/\pi)}$. This result was extended to unranked and node-labelled trees in [3] (with a different constant c). An alternative proof to the result of Flajolet et al. was presented in [15] by Ralaivaosaona and Wagner. For the so-called binary search tree model, Flajolet, Gourdon and Martinez [7] and Devroye [6] proved that the average size of the minimal DAG becomes $\Theta(n/\log n)$. In the binary search tree model, a binary search tree of size n is built by inserting the keys $1, \ldots, n$ according to a uniformly chosen random permutation on $1, \ldots, n$.

A general concept to produce probability distributions on the set of binary trees of size n was introduced by Zhang, Yang, and Kieffer in [16] (see also [11]), where the authors extend the classical notion of an information source on finite sequences to so-called *structured binary tree sources*, or binary tree sources for short. This yields a general framework for studying the average size of a minimal DAG. Let $\mathcal{T}$ denote the set of all binary trees[1] and let $\mathcal{T}_n$ denote the set of binary trees with n leaves. A binary tree source is a tuple $(\mathcal{T}, (\mathcal{T}_n)_{n\in\mathbb{N}}, P)$, in which P is a mapping from the set of binary trees to the unit intervall $[0,1]$, such that $\sum_{t\in\mathcal{T}_n} P(t) = 1$ for every $n \geq 1$. This is a very general definition that was further restricted by Zhang et al. in order to yield interesting results. More precisely, they considered so-called *leaf-centric binary tree sources*, which are induced by a mapping $\sigma : (\mathbb{N}\setminus\{0\}) \times (\mathbb{N}\setminus\{0\}) \to [0,1]$ that satisfies $\sum_{i=1}^{n-1} \sigma(i, n-i) = 1$ for every $n \geq 2$. In other words, σ restricted to $S_n := \{(i, n-i) \colon 1 \leq i \leq n-1\}$ is a probability mass function for every $n \geq 2$. To randomly produce a tree with n leaves, one starts with a single root node labelled with n and randomly chooses a pair $(i, n-i)$ according to the distribution σ on S_n. Then, a left (resp., right) child labelled with i (resp.-, $n-i$) is attached to the root, and the process is repeated with these two nodes. The process stops at nodes with label 1. This yields a function P_σ that restricts to a probability mass function on every set $\mathcal{T}_n$ for $n \geq 2$.

The binary search tree model is the leaf-centric binary tree source where σ corresponds to the uniform distribution on S_n for every $n \geq 2$. Moreover, also the uniform distribution on all trees with n leaves can be obtained from a leaf-centric binary tree source by choosing σ suitably, see Section 4. Another well-known leaf-centric binary tree source is the *digitial search tree model* [13], where the distribution on S_n is a binomial distribution.

Let $\mathcal{D}_t$ denote the minimal DAG of a binary tree t and let $|\mathcal{D}_t|$ denote the number of nodes of $\mathcal{D}_t$. The average size of the minimal DAG with respect to a leaf-centric binary tree source $(\mathcal{T}, (\mathcal{T}_n)_{n\in\mathbb{N}}, P_\sigma)$ is the mapping

$$\mathcal{D}_\sigma(n) := \sum_{t\in\mathcal{T}_n} P_\sigma(t)|\mathcal{D}_t|. \tag{1}$$

In this work, we generalize the results of [6, 7] on the average size of the minimal DAG with respect to the binary search tree model in several ways. For this, we consider three classes of leaf-centric binary tree sources, which are defined by the following three properties of the corresponding σ-mappings:

- **(i)** There exists an integer $N \geq 2$ and a monotonically decreasing function $\psi : \mathbb{R} \to (0,1]$ such that $\psi(n) \geq \frac{2}{n-1}$ and $\sigma^*(i, n-i) \leq \psi(n)$ for every $n \geq N$ and $1 \leq i \leq n-1$. Here, σ^* is defined by $\sigma^*(i,i) = \sigma(i,i)$ and $\sigma^*(i,j) = \sigma(i,j) + \sigma(j,i)$ for $i \neq j$.
- **(ii)** There exists an integer $N \geq 2$ and a constant $0 < \rho < 1$, such that $\sigma(i, n-i) \leq \rho$ for every $n \geq N$ and $1 \leq i \leq n-1$.
- **(iii)** There is a monotonically decreasing function $\phi : \mathbb{N} \to (0,1]$ and a constant $c \geq 3$ such that for every $n \geq 2$,

$$\sum_{\frac{n}{c} \leq i \leq n-\frac{n}{c}} \sigma(i, n-i) \geq \phi(n).$$

Property (iii) generalizes the concept of *balanced* binary tree sources from [10, 11]: When randomly constructing a binary tree with respect to a leaf-centric source of type (iii), the probability that the current weight is roughly equally splitted among the two children is

[1] We consider binary trees, where every non-leaf node has a left and a right child, but the whole framework can be easily extended to binary trees, where a node may have only a left or right child.

lower bounded by a function. Therefore, for slowly decreasing functions ϕ, balanced trees are preferred by this model. The binary search tree model satisfies all three conditions (i), (ii) and (iii). As our main results, we obtain for each of these three types of leaf-centric binary tree sources asymptotic bounds for the average size of the minimal DAG:

(a) For leaf-centric sources of type (i), the average size of the minimal DAG is upper bounded by $\mathcal{O}\left(\psi\left(\frac{1}{2}\log_4(n)\right)n\right)$, which is in $o(n)$ if $\psi(x) \in o(1)$.

(b) Using a simple entropy argument based on a result from [11], we show that for every leaf-centric binary tree source of type (ii), the average size of the minimal DAG is lower bounded by $\Omega(n/\log n)$.

(c) For leaf-centric binary tree sources of type (iii), the average size of the minimal DAG is upper bounded by $\mathcal{O}\left(\frac{n}{\phi(n)\log n}\right)$, which is in $o(n)$ if $\phi(n) \in \omega(1/\log n)$.

Both (a) and (c) imply the upper bound $\mathcal{O}(n/\log n)$ for the binary search tree model [7], whereas (b) yields an information-theoretic proof of the lower bound $\Omega(n/\log n)$ from [6].

The upper bounds (a) and (c) can be applied to the problem of universal tree compression [10, 16]. It is shown in [16] that a suitable binary encoding of the DAG yields a tree encoding whose average-case redundancy converges to zero assuming the trees are produced by a leaf-centric tree source for which the average DAG size is $o(n)$. See [16] for precise definitions.

2 Preliminaries

We use the classical Landau notations $\mathcal{O}$, o, Ω and ω. Quite often, we write sums of the form $\sum_{q_0 \le k \le q_1} a_k$ for rational numbers q_0, q_1. With this, we mean the sum $\sum_{k=\lceil q_0 \rceil}^{\lfloor q_1 \rfloor} a_k$. In the following, $\log x$ will always denote the binary logarithm $\log_2 x$ of a positive real number x. With $[0,1]$ we denote the unit interval of reals, and $(0,1] = [0,1] \setminus \{0\}$.

2.1 Trees and DAGs

We define binary trees as terms over the two symbols a (for leaves) and f (for binary nodes). The set $\mathcal{T}$ of *binary trees* is the smallest set of terms in f and a such that (i) $a \in \mathcal{T}$, and (ii) if $t_1, t_2 \in \mathcal{T}$, then $f(t_1, t_2) \in \mathcal{T}$. Thus, if we consider elements in $\mathcal{T}$ as graphs in the usual way, a binary tree is an ordered, rooted tree such that each node has either exactly two or no children. With $\mathcal{T}_n$ we denote the set of binary trees which have exactly n leaves. The *size* of a binary tree t is the number of leaves of t and denoted with $|t|$. A *fringe subtree* of a binary tree t is a subtree which consists of a node of t and all its descendants. For a node v of a binary tree $t \in \mathcal{T}$, let $t[v]$ denote the fringe subtree of t which is rooted at v. The *leaf-size* of a node v of t is the size of the subtree $t[v]$. For a binary tree $t \in \mathcal{T}$ and an integer $k \ge 1$, let $N(t,k)$ denote the number of nodes of t of leaf-size greater than k.

For a binary tree $t \in \mathcal{T}$, let $\mathcal{D}_t$ denote its minimal *directed acyclic graph*, often shortly referred to as its minimal DAG. It is obtained by merging nodes u and v if $t[u]$ and $t[v]$ are isomorphic. The size $|\mathcal{D}_t|$ of $\mathcal{D}_t$ is the number of different pairwise non-isomorphic fringe subtrees of t. An example of a binary tree and its minimal DAG can be found in Figure 1.

2.2 Leaf-centric binary tree sources

In this paper we are interested in the average size of minimal DAGs. For this, we need for every $n \ge 1$ a probability distribution on $\mathcal{T}_n$. We restrict here to so-called leaf-centric binary tree sources that were studied in [11, 16]. Let Σ denote the set of all functions

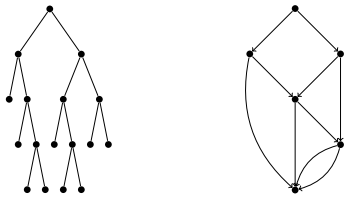


Figure 1 A binary tree (left) and its minimal DAG (right).

$\sigma : (\mathbb{N} \setminus \{0\}) \times (\mathbb{N} \setminus \{0\}) \to [0,1]$ which satisfy

$$\sum_{i=1}^{n-1} \sigma(i, n-i) = 1$$

for every integer $n \geq 2$. We define $P_\sigma : \mathcal{T} \to [0,1]$ inductively by $P_\sigma(a) = 1$ and $P_\sigma(f(u,v)) = \sigma(|u|,|v|) \cdot P_\sigma(u) \cdot P_\sigma(v)$. For every $n \geq 1$, P_σ restricts to a probability mass function on $\mathcal{T}_n$. The tuple $(\mathcal{T}, (\mathcal{T}_n)_{n\in\mathbb{N}}, P_\sigma)$ is called a *leaf-centric binary tree source*.

For an element $\sigma \in \Sigma$ define the mapping $\sigma^* : (\mathbb{N} \setminus \{0\}) \times (\mathbb{N} \setminus \{0\}) \to [0,1]$ by

$$\sigma^*(i,j) = \begin{cases} \sigma(i,j) + \sigma(j,i) & \text{if } i \neq j \\ \sigma(i,j) & \text{if } i = j. \end{cases}$$

Note that $\sigma^*(i,j) \leq 1$ for all i,j and that $\sum_{k=1}^{\lfloor n/2 \rfloor} \sigma^*(k, n-k) = 1$.

3 Average size of the minimal DAG

Consider $\sigma \in \Sigma$. The *average size of the minimal DAG* with respect to the leaf-centric binary tree source $(\mathcal{T}, (\mathcal{T}_n)_{n\in\mathbb{N}}, P_\sigma)$ is the function $\mathcal{D}_\sigma : \mathbb{N} \to \mathbb{R}$ defined by equation (1). In the following, we present three natural classes of leaf-centric binary tree sources and investigate the average size of the minimal DAG with respect to these leaf-centric binary tree sources. In particular, we present conditions on $\sigma \in \Sigma$ that imply $\mathcal{D}_\sigma(n) \in o(n)$. In order to estimate $\mathcal{D}_\sigma$, we use the so-called cut-point argument that was applied in several papers [6, 15].

For a mapping $\sigma \in \Sigma$ and integers $b \geq 1$ and $n \geq 1$, let $E_{\sigma,b}(n)$ denote the expected value of $N(t,b)$ with respect to the probability mass function P_σ on the set of binary trees $\mathcal{T}_n$:

$$E_{\sigma,b}(n) = \sum_{t \in \mathcal{T}_n} P_\sigma(t) \cdot N(t,b).$$

Clearly, $E_{\sigma,b}(n) = 0$ if $n \leq b$. The following lemma constitutes the crucial argument we need in order to estimate the average size of a minimal DAG.

▶ **Lemma 1.** *Let $\sigma \in \Sigma$ and let $n \geq b \geq 1$. Then $\mathcal{D}_\sigma(n) \leq E_{\sigma,b}(n) + 4^b/3$.*

Proof. Let $t \in \mathcal{T}_n$. The size of the minimal DAG $\mathcal{D}_t$ of t is upper bounded by

(i) the number $N(t,b)$ of nodes of t of leaf-size greater than b plus

(ii) the number of binary trees with at most b leaves.

Recall that the number of binary trees with k leaves is the $(k-1)^{\text{th}}$ Catalan number C_{k-1}, which is bounded by 4^{k-1}. Hence, the number in (ii) is upper bounded by $\sum_{k=1}^{b} 4^{k-1} \leq 4^b/3$. This proves the lemma. ◀

The integer $b \geq 1$ from Lemma 1 is called the cutpoint. In order to apply Lemma 1 to estimate $\mathcal{D}_\sigma$, we first have to obtain estimates for $E_{\sigma,b}(n)$. This will be done inductively: Let $t = f(u,v) \in \mathcal{T}_n$ and let $b < n$. The number of nodes of t of leaf-size greater than b is composed of the number of nodes of the left subtree u of leaf-size greater than b plus the number of nodes of the right subtree v of leaf-size greater than b plus one (for the root), i.e., $N(t,b) = N(u,b) + N(v,b) + 1$. This observation easily yields the following recurrence relation for the expected value $E_{\sigma,b}(n)$:

$$E_{\sigma,b}(n) \;=\; 1 + \sum_{k=b+1}^{n-1} (\sigma(k, n-k) + \sigma(n-k, k)) \cdot E_{\sigma,b}(k).$$

With our definition of σ^*, this is equivalent to

$$E_{\sigma,b}(n) = \; 1 + \sum_{k=b+1}^{n-1} \sigma^*(k, n-k) \cdot E_{\sigma,b}(k) \tag{2}$$

if $b+1 > \frac{n}{2}$ and

$$E_{\sigma,b}(n) = 1 + \sum_{k=b+1}^{n-b-1} \sigma(k, n-k)(E_{\sigma,b}(k) + E_{\sigma,b}(n-k)) \; + \sum_{k=n-b}^{n-1} \sigma^*(k, n-k) E_{\sigma,b}(k) \tag{3}$$

if $b+1 \leq \frac{n}{2}$.

3.1 Average size of the minimal DAG for bounded σ-functions

First, we consider leaf-centric binary tree sources $(\mathcal{T}, (\mathcal{T}_n)_{n\in\mathbb{N}}, P_\sigma)$, where the function values of σ (or σ^*) are upper bounded by a function. We will prove an upper as well as a lower bound on the average DAG size.

3.1.1 Upper bound on the average DAG size

▶ **Definition 2** (the class Σ_*^ψ)**.** For a monotonically decreasing function $\psi : \mathbb{R} \to (0,1]$ such that $\psi(x) \geq 2/(x-1)$ for all large enough $x > 1$, let $\Sigma_*^\psi \subseteq \Sigma$ denote the set of mappings $\sigma \in \Sigma$ such that $\sigma^*(k, n-k) \leq \psi(n)$ for all large enough $n \geq 2$ and all $1 \leq k \leq n-1$.

The restriction $\psi(x) \geq 2/(x-1)$ is quite natural, at least for odd $x \in \mathbb{N}$, because $\sum_{k=1}^{n-1} \sigma^*(k, n-k) = 2$ if n is odd.

As our first main theorem, we prove an upper bound on $\mathcal{D}_\sigma(n)$ for every $\sigma \in \Sigma_*^\psi$:

▶ **Theorem 3.** *For every $\sigma \in \Sigma_*^\psi$, we have $\mathcal{D}_\sigma(n) \in \mathcal{O}\left(\psi\left(\frac{1}{2}\log_4(n)\right) \cdot n\right)$.*

Note that Theorem 3 only makes a nontrivial statement if ψ converges to zero: if ψ is lower bounded by a nonzero constant then we only obtain the trivial bound $\mathcal{D}_\sigma(n) \in \mathcal{O}(n)$. Moreover, the bound $\mathcal{D}_\sigma(n) \in \mathcal{O}\left(\psi\left(\frac{1}{2}\log_4(n)\right) \cdot n\right)$ also holds if we require that $\sigma(k, n-k) \leq \psi(n)$ for all large enough n and $1 \leq k \leq n-1$, since the latter implies that $\sigma^*(k, n-k) \leq 2\psi(n)$.

Let us fix a monotonically decreasing function $\psi : \mathbb{R} \to (0,1]$ such that $\psi(n) \geq 2/(n-1)$ for all large enough n. Moreover, let $\sigma \in \Sigma_*^\psi$. We can choose a constant N_σ such that $\psi(n) \geq 2/(n-1)$ and $\sigma^*(k, n-k) \leq \psi(n)$ for all $n \geq N_\sigma$ and all $1 \leq k \leq n-1$. In order to prove Theorem 3, we use the cut-point argument from Lemma 1. Thus, we start with an upper bound for $E_{\sigma,b}(n)$. A similar statement for the special case of the binary search tree model was shown by Knuth [12, p. 121].

▶ **Lemma 4.** *For all n, b with $n \geq b+1 > N_\sigma$ we have $E_{\sigma,b}(n) \leq 4n\psi(b) - 2$.*

In the proof of Lemma 4, we make use of the following lemma from linear optimization:

▶ **Lemma 5.** *Let $a_0 \leq a_1 \leq \cdots \leq a_{n-1}$ be a finite sequence of monotonically increasing positive real numbers and let $0 \leq c, \omega \leq 1$ and $l := \lfloor \omega/c \rfloor$. Moreover, let $x_0, \ldots, x_{n-1}$ denote real numbers satisfying $0 \leq x_i \leq c$ for every $0 \leq i \leq n-1$ and $\sum_{k=0}^{n-1} x_k = \omega$. Then*

$$\sum_{i=0}^{n-1} a_i x_i \leq c \sum_{i=n-l}^{n-1} a_i + (\omega - lc) a_{n-l-1}. \tag{4}$$

Proof. Since $0 \leq a_0 \leq a_1 \leq \cdots \leq a_{n-1}$ and $0 \leq x_i \leq c$, the sum $\sum_{i=0}^{n-1} a_i x_i$ is maximized if we choose the maximal weight c for the l largest values $a_{n-l} \leq \cdots \leq a_{n-1}$ (i.e., $x_{n-l} = \cdots = x_{n-1} = c$), and put the remaining weight $\omega - lc$ (note that $\omega/c - 1 \leq l \leq \omega/c$, which implies $0 \leq \omega - lc \leq c$) on the $(l-1)$-th largest value a_{n-l-1} (i.e., $x_{n-l-1} = \omega - l \cdot c$). The remaining $x_1, \ldots, x_{n-l-2}$ are set to zero. Then $\sum_{i=0}^{n-1} a_i x_i$ becomes the right-hand side of (4). ◀

Proof of Lemma 4. We prove the statement inductively in $n \geq b+1 > N_\sigma$. For the base case, let $n = b+1$. We have $E_{\sigma,b}(b+1) = 1 \leq 4(b+1)\psi(b) - 2$, as $\psi(b) \geq \frac{2}{b-1}$ by assumption.

For the induction step take an $n > b+1 > N_\sigma$ such that $E_{\sigma,b}(k) \leq 4k\psi(b) - 2$ for every $b < k \leq n-1$. By assumption, we have $n - 1 \geq n - \frac{1}{\psi(n)} > \frac{n}{2}$, as $n > N_\sigma$. We distinguish three subcases:

Case 1: $\frac{n}{2} < b+1 < n - \frac{1}{\psi(n)}$. By equation (2) and the induction hypothesis, we have

$$E_{\sigma,b}(n) \leq 1 + \sum_{k=b+1}^{n-1} \sigma^*(k, n-k) \left(4k\psi(b) - 2\right). \tag{5}$$

Note that $\frac{n}{2} < b+1$ implies that $\sum_{k=b+1}^{n-1} \sigma^*(k, n-k) \leq 1$. Without loss of generality, we can assume that $\sum_{k=b+1}^{n-1} \sigma^*(k, n-k) = 1$: since $4k\psi(b) - 2 > 0$ for every k with $b+1 \leq k \leq n-1$, this makes the right-hand side in (5) only larger. Let $l := \left\lfloor \frac{1}{\psi(n)} \right\rfloor$ and $\delta := \frac{1}{\psi(n)} - l$. Applying Lemma 5 (with $a_k = 4k\psi(b) - 2$, $x_k = \sigma^*(k, n-k)$, $c = \psi(n)$ and $\omega = 1$), we get

$$E_{\sigma,b}(n) \leq 1 + \psi(n) \sum_{k=n-l}^{n-1} (4k\psi(b) - 2) + (1 - l\psi(n)) \left(4(n-l-1)\psi(b) - 2\right). \tag{6}$$

By simplifying the right hand side and using $0 \leq \delta < 1$ and $\psi(n) \leq \psi(b)$, we get

$$\begin{aligned} E_{\sigma,b}(n) &\leq 4n\psi(b) - 1 - 4l\psi(b) - 4\psi(b) + 2l^2\psi(n)\psi(b) + 2l\psi(n)\psi(b) \\ &= 4n\psi(b) - 1 - \frac{2\psi(b)}{\psi(n)} - 2\psi(b) - 2\delta\psi(n)\psi(b) + 2\delta^2\psi(n)\psi(b) \\ &\leq 4n\psi(b) - 1 - \frac{2\psi(b)}{\psi(n)} - 2\psi(b) \leq 4n\psi(b) - 2. \end{aligned}$$

Case 2: $b+1 \geq n - \frac{1}{\psi(n)}$. By equation (2) and by the induction hypothesis, we get

$$E_{\sigma,b}(n) \leq 1 + \sum_{k=b+1}^{n-1} \sigma^*(k, n-k) \left(4k\psi(b) - 2\right).$$

Again, let $l := \left\lfloor \frac{1}{\psi(n)} \right\rfloor$ and $\delta := \frac{1}{\psi(n)} - l$. Since $b+1 \geq n - \frac{1}{\psi(n)}$ by assumption and $b+1 \in \mathbb{N}$ we have $b+1 \geq n-l$. Moreover, $n - \frac{1}{\psi(n)} > \frac{n}{2}$ implies $n - l > \frac{n}{2}$. Since $n-l$ is an integer, we get $n-l-1 \geq \frac{n-1}{2}$. This implies

$$4(n-l-1)\psi(b) - 2 \geq 2(n-1)\psi(b) - 2 \geq 2(n-1)\psi(n) - 2 \geq 2(n-1)\frac{2}{n-1} - 2 > 0$$

and hence also $4k\psi(b) - 2 > 0$ for all $n-l-1 \leq k \leq n-1$. As $\sigma \in \Sigma_*^\psi$, we have $\sigma^*(k, n-k) \leq \psi(n)$ for every $1 \leq k \leq n-1$. We get

$$E_{\sigma,b}(n) \leq 1 + \psi(n) \sum_{k=n-l}^{n-1} (4k\psi(b) - 2)\,.$$

Moreover, we have $1 - \psi(n)l \geq 0$ and $4(n-l-1)\psi(b) - 2 \geq 0$ and thus

$$E_{\sigma,b}(n) \leq 1 + \psi(n) \sum_{k=n-l}^{n-1} (4k\psi(b) - 2) + (1 - \psi(n)l)\,(4(n-l-1)\psi(b) - 2)\,.$$

This is equation (6) from Case 1. The statement follows now as in Case 1.

Case 3: $b+1 \leq \frac{n}{2}$. By equation (3) and the induction hypothesis, we have

$$\begin{aligned} E_{\sigma,b}(n) &= 1 + \sum_{k=b+1}^{n-b-1} \sigma(k, n-k)\,(E_{\sigma,b}(k) + E_{\sigma,b}(n-k)) + \sum_{k=n-b}^{n-1} \sigma^*(k, n-k) E_{\sigma,b}(k) \\ &\leq 1 + (4n\psi(b) - 4) \sum_{k=b+1}^{n-b-1} \sigma(k, n-k) + \sum_{k=n-b}^{n-1} \sigma^*(k, n-k)\,(4k\psi(b) - 2)\,. \end{aligned}$$

We set $\alpha := \sum_{k=b+1}^{n-b-1} \sigma(k, n-k)$. Hence, we have $\sum_{k=n-b}^{n-1} \sigma^*(k, n-k) = 1 - \alpha$. Set $l := \lfloor \frac{1-\alpha}{\psi(n)} \rfloor$. Note that $l \leq \frac{1}{\psi(n)} \leq \frac{n-1}{2}$ and that $4k\psi(b) - 2 \geq 0$ for all $n - b \leq k \leq n-1$ since $n - b > \frac{n}{2}$ and $\psi(b) \geq \psi(n) > \frac{2}{n}$. We distinguish two subcases:

Case 3.1: $b > l$ and thus, $n - b < n - l$. Applying Lemma 5 (with $a_k = 4k\psi(b) - 2$ and $x_k = \sigma^*(k, n-k)$ for $n - b \leq k \leq n-1$ and $c = \psi(n)$, $\omega = 1 - \alpha$) yields

$$\begin{aligned} E_{\sigma,b}(n) \leq\ & 1 + (4n\psi(b) - 4)\,\alpha + \psi(n) \sum_{k=n-l}^{n-1} (4k\psi(b) - 2) \\ & + (1 - \alpha - l\psi(n))\,(4(n-l-1)\psi(b) - 2)\,. \end{aligned} \tag{7}$$

Simplifying the right-hand side yields

$$E_{\sigma,b}(n) \leq 4n\psi(b) - 2\alpha - 1 + 2l\psi(n)\psi(b) + 2l^2\psi(n)\psi(b) - 4(1-\alpha)\psi(b) - 4(1-\alpha)l\psi(b).$$

Setting $\delta := \frac{(1-\alpha)}{\psi(n)} - l$, we get

$$E_{\sigma,b}(n) \leq 4n\psi(b) - 2\alpha - 1 - 2(1-\alpha)\psi(b) - \frac{2\psi(b)(1-\alpha)^2}{\psi(n)} - 2\delta\psi(n)\psi(b) + 2\delta^2\psi(n)\psi(b).$$

As $0 \leq \delta < 1$ and $\psi(n) \leq \psi(b)$, we have

$$\begin{aligned} E_{\sigma,b}(n) &\leq 4n\psi(b) - 2\alpha - 1 - 2(1-\alpha)\psi(b) - \frac{2\psi(b)(1-\alpha)^2}{\psi(n)} \\ &\leq 4n\psi(b) - 2\alpha - 1 - 2(1-\alpha)\psi(b) - 2(1-\alpha)^2. \end{aligned}$$

With $-2\alpha - 2(1-\alpha)^2 \leq -1$ for every value $0 \leq \alpha \leq 1$, the statement follows.

Case 3.2: $b \leq l$ and thus $n - b \geq n - l$. Since $n - l - 1 \geq n - \frac{n-1}{2} - 1 = \frac{n-1}{2}$ and $\psi(b) \geq \psi(n) \geq \frac{2}{n-1}$ we have $4(n-l-1)\psi(b) - 2 \geq 0$. Thus, we also have $4k\psi(b) - 2 \geq 0$ for every $n - l \leq k \leq n - 1$. Moreover, as $\sigma^*(k, n-k) \leq \psi(n)$, we get

$$E_{\sigma,b}(n) \leq 1 + (4n\psi(b) - 4)\,\alpha + \psi(n) \sum_{k=n-l}^{n-1} (4k\psi(b) - 2)\,.$$

Furthermore, as $1 - \alpha - l\psi(n) \geq 0$, we obtain

$$\begin{aligned} E_{\sigma,b}(n) \;\leq\; & 1 + (4n\psi(b) - 4)\,\alpha + \psi(n) \sum_{k=n-l}^{n-1} (4k\psi(b) - 2) \\ & + (1 - \alpha - l\psi(n))\,(4(n-l-1)\psi(b) - 2)\,. \end{aligned}$$

This is equation (7) from Case 3.1, and we can conclude as in Case 3.1. This finishes the proof of Lemma 4. ◀

With Lemma 4, we are able to prove Theorem 3 using the cut-point argument from Lemma 1:

Proof of Theorem 3. Let $\sigma \in \Sigma_*^\psi$, $n > 4^{2N_\sigma}$ and $N_\sigma \leq b < n$. By Lemma 1 and 4 we have $\mathcal{D}_\sigma(n) \leq E_{\sigma,b}(n) + 4^b/3 \leq 4n \cdot \psi(b) + 4^b/3$. Choose $b := \lceil \log_4(n)/2 \rceil$. As $n > 4^{2N_\sigma}$, this accords with $b \geq N_\sigma$. We obtain $\mathcal{D}_\sigma(n) \leq 4n \cdot \psi\left(\log_4(n)/2\right) + \Theta(\sqrt{n})$. Since $n \cdot \psi\left(\log_4(n)/2\right) \geq \frac{2n}{\log_4(n)/2-1}$ grows faster than $\Theta(\sqrt{n})$, this finishes the proof. ◀

In the following examples, we consider the results of Theorem 3 with respect to some concrete functions ψ:

▶ **Example 6.** Let $\sigma_{\mathrm{bst}}(k, n-k) = \frac{1}{n-1}$ for every integer $1 \leq k \leq n-1$ and $n \geq 2$. The leaf-centric binary tree source $(\mathcal{T}, (\mathcal{T}_n)_{n\geq 1}, P_{\sigma_{\mathrm{bst}}})$ corresponds to the well-known binary search tree model. Let $\psi(x) = \frac{2}{x-1}$ for every $x > 1$. We find $\sigma_{\mathrm{bst}} \in \Sigma_*^\psi$. With Theorem 3, we have $\mathcal{D}_{\sigma_{\mathrm{bst}}} \in \mathcal{O}(n/\log n)$, which accords with the results of [6]. ◀

▶ **Example 7.** There are plenty of other ways to choose ψ in Theorem 3. For example $\psi(x) \in \Theta(1/x^\alpha)$ with $0 \leq \alpha \leq 1$ yields $\mathcal{D}_\sigma(n) \in \mathcal{O}(n/\log(n)^\alpha)$ for every $\sigma \in \Sigma_*^\psi$. For $\psi(x) \in \Theta(1/\log x)$ we get $\mathcal{D}_\sigma(n) \in \mathcal{O}(n/\log\log n)$ for every $\sigma \in \Sigma_*^\psi$. ◀

3.1.2 Lower bound on the average DAG size

In this section we prove a lower bound for $\mathcal{D}_\sigma(n)$.

▶ **Definition 8** (the class Σ^ρ)**.** For a constant ρ with $0 < \rho < 1$ let Σ^ρ denote the set of mappings $\sigma \in \Sigma$ such that $\sigma(k, n-k) \leq \rho$ for all large enough n and all $1 \leq k \leq n-1$.

By Theorem 3, we only know $\mathcal{D}_\sigma(n) \in \mathcal{O}(n)$ for $\sigma \in \Sigma^\rho$. In the following theorem, we present a lower bound for $\mathcal{D}_\sigma(n)$ with respect to a mapping $\sigma \in \Sigma^\rho$:

▶ **Theorem 9.** *If $\sigma \in \Sigma^\rho$, then $\mathcal{D}_\sigma(n) \in \Omega(n/\log n)$.*

Let us fix a mapping $\sigma \in \Sigma^\rho$, where $0 < \rho < 1$, and let $N_\sigma \geq 2$ such that $\sigma(k, n-k) \leq \rho$ for all $n \geq N_\sigma$ and all $1 \leq k \leq n-1$. In order to prove Theorem 9, we make use of an information-theoretic argument. We need the following notations: For a mapping $\sigma \in \Sigma$, let X_σ^n denote the random variable taking values in $\mathcal{T}_n$ according to the probability mass function P_σ on $\mathcal{T}_n$. Moreover, let $H(X_\sigma^n)$ denote the Shannon entropy of X_σ^n, i.e.,

$$H(X_\sigma^n) = \sum_{t \in \mathcal{T}_n} P_\sigma(t) \cdot \log(1/P_\sigma(t)).$$

▶ **Lemma 10.** *If $\sigma \in \Sigma^\rho$, then $H(X_\sigma^n) \geq \log\left(\frac{1}{\rho}\right)\left(\frac{n}{4N_\sigma - 4}\right)$ for every $n \geq N_\sigma$.*

In order to prove Lemma 10, we need a lower bound for $E_{\sigma,b}(n)$:

▶ **Lemma 11.** *For a mapping $\sigma \in \Sigma$ and integers $n > b \geq 1$, we have $E_{\sigma,b}(n) \geq \frac{n}{4b}$.*

Proof. We prove the statement inductively in $n \geq b+1$: For the base case, let $n = b+1$. A binary tree $t \in \mathcal{T}_{b+1}$ has exactly one node of leaf-size greater than b, which is the root of t. Thus, $E_{\sigma,b}(b+1) = 1 \geq \frac{b+1}{4b}$ for every integer $b \geq 1$. For the induction hypothesis, take an integer $n > b+1$ such that $E_{\sigma,b}(k) \geq \frac{k}{4b}$ for every integer $b+1 \leq k \leq n-1$.

In the induction step, we distinguish two cases:

Case 1: $\frac{n}{2} < b+1 \leq n-1$. We thus have $\frac{n}{4b} \leq 1$. By equation (2), we have

$$E_{\sigma,b}(n) = 1 + \sum_{k=b+1}^{n-1} \sigma^*(k, n-k) E_{\sigma,b}(k) \geq 1 \geq \frac{n}{4b}.$$

Case 2: $b+1 \leq \frac{n}{2}$. Let $\alpha := \sum_{k=b+1}^{n-b-1} \sigma(k, n-k)$. From equation (3) and the induction hypothesis we get

$$\begin{aligned}
E_{\sigma,b}(n) &= 1 + \sum_{k=b+1}^{n-b-1} \sigma(k,n-k)(E_{\sigma,b}(k) + E_{\sigma,b}(n-k)) + \sum_{k=n-b}^{n-1} \sigma^*(k,n-k) E_{\sigma,b}(k) \\
&\geq 1 + \sum_{k=b+1}^{n-b-1} \sigma(k,n-k)\frac{n}{4b} + \sum_{k=n-b}^{n-1} \sigma^*(k,n-k)\frac{k}{4b} \\
&\geq 1 + \frac{n}{4b}\sum_{k=b+1}^{n-b-1} \sigma(k,n-k) + \frac{n-b}{4b}\sum_{k=n-b}^{n-1} \sigma^*(k,n-k) \\
&= 1 + \alpha\frac{n}{4b} + (1-\alpha)\left(\frac{n-b}{4b}\right) = \frac{n}{4b} + \frac{3}{4} + \frac{\alpha}{4}.
\end{aligned}$$

As $0 \leq \alpha \leq 1$, the statement follows. ◀

Proof of Lemma 10. Lemma 10 follows from identity (4) in [11]: Define

$$h_k(\sigma) := \sum_{\substack{i,j \geq 1 \\ i+j=k}} \sigma(i,j) \log\left(\frac{1}{\sigma(i,j)}\right),$$

that is, $h_k(\sigma)$ is the Shannon entropy of the random variable taking values in $\{(i, k-i) \colon 1 \leq i \leq k-1\}$ according to the probility mass function σ. As $\sigma(i,j) \leq \rho$ for $i+j \geq N_\sigma$, we find

$$h_k(\sigma) \geq \log\left(\frac{1}{\rho}\right) \sum_{\substack{i,j \geq 1 \\ i+j=k}} \sigma(i,j) = \log\left(\frac{1}{\rho}\right)$$

for every $k \geq N_\sigma$. Identity (4) in [11] states that $H(X_\sigma^n) = \sum_{j=2}^n \left(E_{\sigma,j-1}(n) - E_{\sigma,j}(n)\right) h_j(\sigma)$. With $n \geq N_\sigma$, we obtain

$$\begin{aligned}
H(X_\sigma^n) &\geq \sum_{j=N_\sigma}^{n} \left(E_{\sigma,j-1}(n) - E_{\sigma,j}(n)\right) h_j(\sigma) \geq \log\left(\frac{1}{\rho}\right) \sum_{j=N_\sigma}^{n} \left(E_{\sigma,j-1}(n) - E_{\sigma,j}(n)\right) \\
&= \log\left(\frac{1}{\rho}\right)\left(E_{\sigma,N_\sigma-1}(n) - E_{\sigma,n}(n)\right) = \log\left(\frac{1}{\rho}\right) E_{\sigma,N_\sigma-1}(n).
\end{aligned}$$

By Lemma 11, we have $H(X_\sigma^n) \geq \log\left(\frac{1}{\rho}\right)\left(\frac{n}{4N_\sigma - 4}\right)$. This proves the statement. ◀

With Lemma 10, we are able to prove Theorem 9:

Proof of Theorem 9. We first show that a binary tree $t \in \mathcal{T}_n$ can be encoded with at most $2m\lceil\log(2n-1)\rceil$ bits, where $m = |\mathcal{D}_t| \leq 2n-1$ (note that t has exactly $2n-1$ nodes). It suffices to encode $\mathcal{D}_t$. W.l.o.g. assume that the nodes of $\mathcal{D}_t$ are the numbers $1,\dots,m$, where m is the unique leaf node of $\mathcal{D}_t$. For $1 \leq k \leq m-1$ let l_k (resp., r_k) be the left (resp., right) child of node k. We encode each number $1,\dots,m$ by a bit string of length exactly $\lceil\log(2n-1)\rceil$. The DAG $\mathcal{D}_t$ can be uniquely encoded by the bit string $l_1r_1l_2r_2\cdots l_{m-1}r_{m-1}$, which has length $2(m-1)\lceil\log(2n-1)\rceil$.

Let $\sigma \in \Sigma^\rho$. By Lemma 10, we know that $H(X_\sigma^n) \geq \log\left(\frac{1}{\rho}\right)\left(\frac{n}{4N_\sigma-4}\right)$ for every $n \geq N_\sigma$. Shannon's coding theorem implies

$$H(X_\sigma^n) \leq 2\lceil\log(2n-1)\rceil \sum_{t\in\mathcal{T}_n} P_\sigma(t)|\mathcal{D}_t| = 2\lceil\log(2n-1)\rceil\mathcal{D}_\sigma(n).$$

We get $\log(1/\rho)\left(\frac{n}{4N_\sigma-4}\right) \leq 2\lceil\log(2n-1)\rceil\mathcal{D}_\sigma(n)$ for all $n \geq 2$, which concludes the proof. ◀

3.2 Average size of the minimal DAG for weakly balanced tree sources

In this subsection, we present so-called *weakly balanced* binary tree sources, which represent a generalization of balanced binary tree sources introduced in [11] and further analysed in [10]. Let us fix a constant $c \geq 3$ for the rest of this subsection.

▶ **Definition 12** (the class Σ_ϕ)**.** For a monotonically decreasing function $\phi : \mathbb{N} \to (0,1]$ let $\Sigma_\phi \subseteq \Sigma$ denote the set of mappings σ such that for every $n \geq 2$,

$$\sum_{\frac{n}{c}\leq k\leq n-\frac{n}{c}} \sigma(k,n-k) \geq \phi(n).$$

We call a binary tree source $(\mathcal{T},(\mathcal{T}_n)_{n\geq 1},P_\sigma)$ with $\sigma \in \Sigma_\phi$ *weakly balanced*. We obtain the following upper bound for $\mathcal{D}_\sigma$ with respect to a weakly balanced tree source:

▶ **Theorem 13.** *For every $\sigma \in \Sigma_\phi$, we have $\mathcal{D}_\sigma(n) \in \mathcal{O}\left(\frac{n}{\phi(n)\log n}\right)$.*

Theorem 13 can be used to reprove the upper bound $\mathcal{D}_{\sigma_{\text{bst}}}(n) \in \mathcal{O}(n/\log n)$ for the binary search tree model from Example 6 (note that $\sum_{n/4\leq k\leq 3n/4}\frac{1}{n-1} > \frac{1}{2}$). More generally, if $\phi(n) \in \omega(1/\log n)$, then Theorem 13 yields $\mathcal{D}_\sigma(n) \in o(n)$ for every $\sigma \in \Sigma_\phi$.

Analogously to Theorem 3, we show Theorem 13 using the cut-point argument from Lemma 1. The strategy in the proof of the following lemma is similar to Lemma 4.

▶ **Lemma 14.** *For every $\sigma \in \Sigma_\phi$ and all $b \geq 1$, $n \geq b+1$ we have $E_{\sigma,b}(n) \leq \frac{cn}{\phi(n)b} - \frac{1}{\phi(n)}$.*

Proof. We prove the statement inductively in $n \geq b+1$. For the base case, note that a binary tree $t \in \mathcal{T}_{b+1}$ has exactly one node of leaf-size $> b$, which is the root of t. Thus,

$$E_{\sigma,b}(b+1) = 1 \leq \frac{c(b+1)}{\phi(b+1)b} - \frac{1}{\phi(b+1)}.$$

Let us now deal with the induction step. Take an integer $n > b+1$ such that $E_{\sigma,b}(k) \leq \frac{ck}{\phi(k)b} - \frac{1}{\phi(k)}$ for every integer $b+1 \leq k \leq n-1$. We distinguish six cases:

Case 1: $c \geq n$ and thus $c > b$. We have $\frac{n}{c} \leq 1$ and $n-1 \leq n-\frac{n}{c}$. Case 1 splits into two subcases:

Case 1.1: $\frac{n}{2} < b+1 \leq n-1$. By equation (2), the induction hypothesis, and the fact that ϕ is monotonically decreasing, we get

$$\begin{aligned} E_{\sigma,b}(n) &= 1 + \sum_{k=b+1}^{n-1} \sigma^*(k,n-k)\left(\frac{ck}{\phi(k)b} - \frac{1}{\phi(k)}\right) \\ &\leq 1 + \left(\frac{c(n-1)}{\phi(n)b} - \frac{1}{\phi(n)}\right) \sum_{k=b+1}^{n-1} \sigma^*(k,n-k). \end{aligned}$$

As $b+1 > \frac{n}{2}$ and $\sigma \in \Sigma$, we have $\sum_{k=b+1}^{n-1} \sigma^*(k,n-k) \leq 1$ and thus

$$E_{\sigma,b}(n) \leq \frac{cn}{\phi(n)b} - \frac{c}{\phi(n)b} - \frac{1}{\phi(n)} + 1 \leq \frac{cn}{\phi(n)b} - \frac{1}{\phi(n)}.$$

Case 1.2: $b+1 \leq \frac{n}{2}$. Equation (3), the induction hypothesis, and the fact that ϕ is monotonically decreasing yield

$$\begin{aligned} E_{\sigma,b}(n) &= 1 + \sum_{k=b+1}^{n-b-1} \sigma(k,n-k)\left(E_{\sigma,b}(k) + E_{\sigma,b}(n-k)\right) + \sum_{k=n-b}^{n-1} \sigma^*(k,n-k)E_{\sigma,b}(k) \\ &\leq 1 + \sum_{k=b+1}^{n-b-1} \sigma(k,n-k)\left(\frac{cn}{\phi(n)b} - \frac{2}{\phi(n)}\right) \\ &\quad + \sum_{k=n-b}^{n-1} \sigma^*(k,n-k)\left(\frac{ck}{\phi(k)b} - \frac{1}{\phi(k)}\right) \\ &\leq 1 + \left(\frac{cn}{\phi(n)b} - \frac{2}{\phi(n)}\right) \sum_{k=b+1}^{n-b-1} \sigma(k,n-k) \\ &\quad + \left(\frac{c(n-1)}{\phi(n)b} - \frac{1}{\phi(n)}\right) \sum_{k=n-b}^{n-1} \sigma^*(k,n-k). \end{aligned}$$

We set $\alpha := \sum_{k=b+1}^{n-b-1} \sigma(k,n-k)$ and get

$$\begin{aligned} E_{\sigma,b}(n) &\leq 1 + \left(\frac{cn}{\phi(n)b} - \frac{2}{\phi(n)}\right)\alpha + \left(\frac{c(n-1)}{\phi(n)b} - \frac{1}{\phi(n)}\right)(1-\alpha) \\ &= \frac{cn}{\phi(n)b} - \frac{c}{\phi(n)b} + 1 - \frac{1}{\phi(n)} + \alpha\left(\frac{c}{\phi(n)b} - \frac{1}{\phi(n)}\right). \end{aligned}$$

As $c > b$ by assumption, the last term is monotonically increasing in α. With $\alpha \leq 1$, we have

$$E_{\sigma,b}(n) \leq \frac{cn}{\phi(n)b} - \frac{2}{\phi(n)} + 1 \leq \frac{cn}{\phi(n)b} - \frac{1}{\phi(n)}.$$

Case 2: $n > c$. We have $\frac{n}{c} > 1$ and $n - \frac{n}{c} < n-1$. Case 2 splits into four subcases:

Case 2.1: $n - \frac{n}{c} < b+1 \leq n-1$. This case is very similar to Case 1.1 and left to the reader; see also the long version [2].

Case 2.2: $\frac{n}{2} < b+1 \leq n - \frac{n}{c}$. Equation (2), the induction hypothesis and the monotonicity of ϕ yield

$$\begin{aligned} E_{\sigma,b}(n) &= 1 + \sum_{b+1 \leq k \leq n-\frac{n}{c}} \sigma^*(k,n-k)E_{\sigma,b}(k) + \sum_{n-\frac{n}{c} < k \leq n-1} \sigma^*(k,n-k)E_{\sigma,b}(k) \\ &\leq 1 + \left(\frac{(c-1)n}{\phi(n)b} - \frac{1}{\phi(n)}\right) \sum_{b+1 \leq k \leq n-\frac{n}{c}} \sigma^*(k,n-k) \\ &\quad + \left(\frac{c(n-1)}{\phi(n)b} - \frac{1}{\phi(n)}\right) \sum_{n-\frac{n}{c} < k \leq n-1} \sigma^*(k,n-k). \end{aligned}$$

We set $\alpha := \sum_{n-\frac{n}{c}<k\leq n-1} \sigma^*(k, n-k)$. Since $b+1 > \frac{n}{2}$ we have $\sum_{b+1\leq k\leq n-\frac{n}{c}} \sigma^*(k, n-k) \leq 1-\alpha$ and get

$$\begin{aligned} E_{\sigma,b}(n) &\leq 1 + (1-\alpha)\left(\frac{(c-1)n}{\phi(n)b} - \frac{1}{\phi(n)}\right) + \alpha\left(\frac{c(n-1)}{\phi(n)b} - \frac{1}{\phi(n)}\right) \\ &= \frac{cn}{\phi(n)b} - \frac{n}{\phi(n)b} - \frac{1}{\phi(n)} + 1 + \alpha\frac{(n-c)}{\phi(n)b}. \end{aligned}$$

As $n > c$ by assumption, the last term is monotonically increasing in α. With $\alpha \leq 1-\phi(n)$ as $\sigma \in \Sigma_\phi$, we find

$$E_{\sigma,b}(n) \leq \frac{cn}{\phi(n)b} - \frac{1}{\phi(n)} + 1 - \frac{c}{\phi(n)b} + \frac{c}{b} - \frac{n}{b} \leq \frac{cn}{\phi(n)b} - \frac{1}{\phi(n)}.$$

Case 2.3: $\frac{n}{c} \leq b+1 \leq \frac{n}{2}$. Equation (3), the induction hypothesis, and the monotonicity of ϕ yield

$$\begin{aligned} E_{\sigma,b}(n) &= 1 + \sum_{k=b+1}^{n-b-1} \sigma(k, n-k)(E_{\sigma,b}(k) + E_{\sigma,b}(n-k)) \\ &\quad + \sum_{n-b\leq k\leq n-\frac{n}{c}} \sigma^*(k, n-k)E_{\sigma,b}(k) + \sum_{n-\frac{n}{c}<k\leq n-1} \sigma^*(k, n-k)E_{\sigma,b}(k) \\ &\leq 1 + \left(\frac{cn}{\phi(n)b} - \frac{2}{\phi(n)}\right)\sum_{k=b+1}^{n-b-1} \sigma(k, n-k) \\ &\quad + \left(\frac{(c-1)n}{\phi(n)b} - \frac{1}{\phi(n)}\right)\sum_{n-b\leq k\leq n-\frac{n}{c}} \sigma^*(k, n-k) \\ &\quad + \left(\frac{c(n-1)}{\phi(n)b} - \frac{1}{\phi(n)}\right)\sum_{n-\frac{n}{c}<k\leq n-1} \sigma^*(k, n-k). \end{aligned}$$

With $\alpha := \sum_{k=b+1}^{n-b-1} \sigma(k, n-k)$ and $\beta := \sum_{n-\frac{n}{c}<k\leq n-1} \sigma^*(k, n-k)$ one can simplify this to

$$E_{\sigma,b}(n) = \frac{cn}{\phi(n)b} - \frac{n}{\phi(n)b} - \frac{1}{\phi(n)} + 1 + \alpha\left(\frac{n}{\phi(n)b} - \frac{1}{\phi(n)}\right) + \beta\left(\frac{n-c}{\phi(n)b}\right). \tag{8}$$

As $b < n$ and $c < n$ by assumption, the term in the last line is monotonically increasing in α and β. Using this fact, as well as $0 \leq \beta \leq 1-\phi(n)$ (as $\sigma \in \Sigma_\phi$), $0 \leq \alpha \leq 1$ and $\alpha+\beta \leq 1$, one can show that

$$\alpha\left(\frac{n}{\phi(n)b} - \frac{1}{\phi(n)}\right) + \beta\left(\frac{n-c}{\phi(n)b}\right) \leq \frac{n}{\phi(n)b} - 1$$

(see the long version [2] for details). Plugging this into (8) yields

$$E_{\sigma,b}(n) \leq \frac{cn}{\phi(n)b} - \frac{n}{\phi(n)b} - \frac{1}{\phi(n)} + 1 + \frac{n}{\phi(n)b} - 1 = \frac{cn}{\phi(n)b} - \frac{1}{\phi(n)}.$$

Case 2.4: $b+1 < \frac{n}{c}$. This case is very similar to Case 1.2 and left to the reader; see also the long version [2].

◀

Proof of Theorem 13. Let $n \geq 2$ and let $1 \leq b < n$. By Lemma 1 and 14, we have

$$\mathcal{D}_\sigma(n) \leq E_{\sigma,b}(n) + \frac{4^b}{3} \leq \frac{cn}{\phi(n)b} + \frac{4^b}{3}.$$

Choosing $b := \lceil \frac{1}{2}\log_4(n) \rceil$, the statement follows. ◀

In the following corollary we identify a constant $\nu \in (0,1]$ with the function mapping every $n \in \mathbb{N}$ to ν. The corollary follows immediately from Theorem 13 and 9.

▶ **Corollary 15.** *For all* $0 < \nu, \rho < 1$ *and all* $\sigma \in \Sigma_\nu \cap \Sigma^\rho$ *we have* $\mathcal{D}_\sigma(n) \in \Theta(n/\log n)$.

▶ **Example 16.** In this example, we investigate the binomial random tree model, which was studied in [11] for the case $p = 1/2$, and which is a slight variant of the digital search tree model, see [13]. Let $0 < p < 1$ and define $\sigma_p \in \Sigma$ by

$$\sigma_p(k, n-k) = p^{k-1}(1-p)^{n-k-1}\binom{n-2}{k-1}$$

for every integer $n \geq 2$ and $1 \leq k \leq n-1$. We use the abbreviation $\pi(i) = \sigma_p(i, n-i)$ in the following. By the binomial theorem, we have $\sum_{k=1}^{n-1} \pi(k) = 1$. In the following, we will prove that $\mathcal{D}_{\sigma_p}(n) \in \mathcal{O}(n/\log n)$. We distinguish two cases.

Case 1: $0 < p \leq \frac{1}{2}$. Let $\nu := 1 - \frac{4-4p}{4+p}$. We find $\nu > 0$ for $0 < p \leq \frac{1}{2}$. We claim that with $c := \frac{6}{p}$, we have $\sigma_p \in \Sigma_\nu$. Then Theorem 13 yields $\mathcal{D}_{\sigma_p}(n) \in \mathcal{O}(n/\log n)$.
In order to prove $\sigma_p \in \Sigma_\nu$, we show

$$\sum_{\frac{np}{6} \leq i \leq n - \frac{np}{6}} \sigma_p(i, n-i) = \sum_{\frac{np}{6} \leq i \leq n - \frac{np}{6}} \pi(i) \geq 1 - \frac{4-4p}{4+p}.$$

Without loss of generality, let $n \geq 3$. Let X_p^n denote the random variable taking values in the set $\{1, \ldots, n-1\}$ according to the probability mass function π. Thus, $X_p^n = 1 + Y_p^n$, where Y_p^n is binomially distributed with parameters $n-2$ and p. For the expected value and variance of X_p^n we obtain $\mathsf{E}[X_p^n] = p(n-2)+1$ and $\mathsf{Var}[X_p^n] = p(1-p)(n-2)$. Let $\kappa := p(n-2)/2$ so that $\mathsf{E}[X_p^n] = 2\kappa + 1$ and $\mathsf{Var}[X_p^n] = 2\kappa(1-p)$. By Chebyshev's inequality, we have ($\mathsf{Prob}(A)$ denotes the probability of the event A)

$$\begin{aligned}\mathsf{Prob}\left(|X_p^n - \mathsf{E}[X_p^n]| < \kappa + 1\right) &\geq 1 - \frac{\mathsf{Var}[X_p^n]}{(\kappa+1)^2} = 1 - \frac{2\kappa(1-p)}{\kappa^2 + 2\kappa + 1} \geq 1 - \frac{2(1-p)}{\kappa+2} \\ &= 1 - \frac{4(1-p)}{p(n-2)+4} \geq 1 - \frac{4(1-p)}{p+4}\end{aligned}$$

where the last inequality holds due to $n \geq 3$. Moreover, with $\mathsf{E}[X_p^n] = 2\kappa + 1$, we have

$$\mathsf{Prob}\left(|X_p^n - \mathsf{E}[X_p^n]| < \kappa + 1\right) = \sum_{\kappa < i < 3\kappa + 2} \pi(i).$$

As $n \geq 3$ and $0 < p \leq \frac{1}{2}$, we have $\kappa \geq \frac{pn}{6}$ and $3\kappa + 2 \leq n - \frac{pn}{6}$. Thus, we have

$$\sum_{\frac{pn}{6} \leq i \leq n - \frac{pn}{6}} \pi(i) \geq \sum_{\kappa < i < 3\kappa+2} \pi(i) = \mathsf{Prob}\left(|X_p^n - \mathsf{E}[X_p^n]| < \kappa + 1\right) \geq 1 - \frac{4(1-p)}{p+4}.$$

This finishes the proof of Case 1.

Case 2: $\frac{1}{2} < p < 1$. Define a mapping $\vartheta : \mathcal{T} \to \mathcal{T}$ inductively by $\vartheta(a) = a$ and $\vartheta(f(u,v)) = f(\vartheta(v), \vartheta(u))$. Intuitively, ϑ exchanges the right child node and the left child node of every node of a binary tree t. It is easy to see that $\vartheta : \mathcal{T}_n \to \mathcal{T}_n$ is a bijection for every $n \geq 1$ and that ϑ^2 is the identity mapping. Moreover, t and $\vartheta(t)$ have the same number of different pairwise non-isomorphic subtrees and thus, $|\mathcal{D}_t| = |\mathcal{D}_{\vartheta(t)}|$. We show inductively in $n \geq 1$, that $P_{\sigma_p}(\vartheta(t)) = P_{\sigma_{1-p}}(t)$ for a binary tree $t \in \mathcal{T}_n$: For the base case, let $t = a$. We find $P_{\sigma_p}(\vartheta(a)) = 1 = P_{\sigma_{1-p}}(a)$.

For the induction step, let $t = f(u, v) \in \mathcal{T}_n$. We have

$$\begin{aligned} P_{\sigma_p}(\vartheta(t)) &= P_{\sigma_p}(f(\vartheta(v), \vartheta(u))) &&= \sigma_p(|\vartheta(v)|, |\vartheta(u)|) P_{\sigma_p}(\vartheta(v)) P_{\sigma_p}(\vartheta(u)) \\ &&&= \sigma_p(|v|, |u|) P_{\sigma_{1-p}}(u) P_{\sigma_{1-p}}(v), \end{aligned}$$

where the last equality holds by the induction hypothesis. Moreover, with $|u| = n - |v|$ and by definition of σ_p, we find that $\sigma_p(|v|, |u|) = \sigma_{1-p}(|u|, |v|)$. Thus, we have

$$\sigma_p(|v|, |u|) P_{\sigma_{1-p}}(u) P_{\sigma_{1-p}}(v) = \sigma_{1-p}(|u|, |v|) P_{\sigma_{1-p}}(u) P_{\sigma_{1-p}}(v) = P_{\sigma_{1-p}}(t).$$

This finishes the induction. Altogether, and as $\vartheta : \mathcal{T}_n \to \mathcal{T}_n$ is a bijection, we get

$$\mathcal{D}_{\sigma_p}(n) = \sum_{t \in \mathcal{T}_n} P_{\sigma_p}(t) |\mathcal{D}_t| = \sum_{t \in \mathcal{T}_n} P_{\sigma_p}(\vartheta(t)) |\mathcal{D}_{\vartheta(t)}| = \sum_{t \in \mathcal{T}_n} P_{\sigma_{1-p}}(t) |\mathcal{D}_t| = \mathcal{D}_{\sigma_{1-p}}(n).$$

Since $\frac{1}{2} < p < 1$, we have $0 < 1 - p < \frac{1}{2}$. Thus, the result for Case 2 follows from Case 1.

4 Open Problems

Perhaps the most natural probability distribution on the set of binary trees with n leaves is the uniform distribution with $P_\sigma(t) = 1/C_{n-1}$ for every $t \in \mathcal{T}_n$, where C_n denotes the n^{th} Catalan number. The corresponding leaf-centric binary tree source is induced by the mapping $\sigma_{\text{eq}} \in \Sigma$ with $\sigma_{\text{eq}}(k, n-k) = C_{k-1} C_{n-k-1} / C_{n-1}$. In [8], it was shown that $\mathcal{D}_{\sigma_{\text{eq}}}(n) \in \Theta(n/\sqrt{\log n})$. Unfortunately, our main results Theorem 3 and Theorem 13 only yield the trivial bound $\mathcal{D}_{\sigma_{\text{eq}}} \in \mathcal{O}(n)$: An easy computation shows that $\sigma_{\text{eq}} \in \Sigma^\rho$ with $\rho = 1/4$ and $\sigma_{\text{eq}} \in \Sigma_\phi$ with $\phi(n) \in \Theta(1/\sqrt{n})$. An interesting open problem would be to find a nontrivial subset $\Sigma' \subseteq \Sigma$ that contains σ_{eq} and such that $\mathcal{D}_\sigma(n) \in \mathcal{O}(n/\sqrt{\log n})$ for all $\sigma \in \Sigma'$.

Another type of binary tree sources are so-called *depth-centric binary tree sources*, which yield probability distributions on the set of binary trees of a fixed depth; see for example [10, 16]. Depth-centric binary tree sources resemble leaf-centric binary tree sources in many ways. An interesting problem would be to estimate the average size of the minimal DAG with respect to certain classes of depth-centric binary tree sources.

References

1 Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley series in computer science / World student series edition. Addison-Wesley, 1986.

2 Louisa Seelbach Benkner and Markus Lohrey. Average case analysis of leaf-centric binary tree sources. *CoRR*, abs/1804.10396, 2018. `arXiv:1804.10396`.

3 Mireille Bousquet-Mélou, Markus Lohrey, Sebastian Maneth, and Eric Noeth. XML compression via DAGs. *Theory of Computing Systems*, 57(4):1322–1371, 2015.

4 Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

5 Peter Buneman, Martin Grohe, and Christoph Koch. Path queries on compressed XML. In Johann Christoph Freytag et al., editors, *Proceedings of the 29th Conference on Very Large Data Bases, VLDB 2003*, pages 141–152. Morgan Kaufmann, 2003.

6 Luc Devroye. On the richness of the collection of subtrees in random binary search trees. *Information Processing Letters*, 65(4):195–199, 1998.

7 Philippe Flajolet, Xavier Gourdon, and Conrado Martínez. Patterns in random binary search trees. *Random Structures & Algorithms*, 11(3):223–244, 1997.

8 Philippe Flajolet, Paolo Sipala, and Jean-Marc Steyaert. Analytic variations on the common subexpression problem. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming, ICALP 1990*, volume 443 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 1990.

9 Markus Frick, Martin Grohe, and Christoph Koch. Query evaluation on compressed trees (extended abstract). In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science, LICS 2003*, pages 188–197. IEEE Computer Society Press, 2003.

10 Danny Hucke and Markus Lohrey. Universal tree source coding using grammar-based compression. In *Proceedings of the 2017 IEEE International Symposium on Information Theory, ISIT 2017*, pages 1753–1757. IEEE, 2017.

11 John C. Kieffer, En-Hui Yang, and Wojciech Szpankowski. Structural complexity of random binary trees. In *Proceedings of the 2009 IEEE International Symposium on Information Theory, ISIT 2009*, pages 635–639. IEEE, 2009.

12 Donald E. Knuth. *The Art of Computer Programming: Volume 3 – Sorting and Searching*. Addison-Wesley, 1998.

13 Conrado Martínez. *Statistics under the BST model*. Dissertation, Universidad Politécnica de Cataluna, 1991.

14 Mike Paterson and Mark N. Wegman. Linear unification. *Journal of Computer and System Sciences*, 16(2):158–167, 1978.

15 Dimbinaina Ralaivaosaona and Stephan G. Wagner. Repeated fringe subtrees in random rooted trees. In *Proceedings of the Twelfth Workshop on Analytic Algorithmics and Combinatorics, ANALCO 2015*, pages 78–88. SIAM, 2015.

16 Jie Zhang, En-Hui Yang, and John C. Kieffer. A universal grammar-based code for lossless compression of binary trees. *IEEE Transactions on Information Theory*, 60(3):1373–1386, 2014.

Expressive Power, Satisfiability and Equivalence of Circuits over Nilpotent Algebras

Paweł M. Idziak

Jagiellonian University, Faculty of Mathematics and Computer Science,
Department of Theoretical Computer Science
Krakow, Poland
idziak@tcs.uj.edu.pl

Piotr Kawałek

Jagiellonian University, Faculty of Mathematics and Computer Science,
Department of Theoretical Computer Science
Krakow, Poland
piotr.kawalek@student.uj.edu.pl

Jacek Krzaczkowski

Jagiellonian University, Faculty of Mathematics and Computer Science,
Department of Theoretical Computer Science
Krakow, Poland
jacek.krzaczkowski@uj.edu.pl

Abstract

Satisfiability of Boolean circuits is NP-complete in general but becomes polynomial time when restricted for example either to monotone gates or linear gates. We go outside Boolean realm and consider circuits built of any fixed set of gates on an arbitrary large finite domain. From the complexity point of view this is connected with solving equations over finite algebras. This in turn is one of the oldest and well-known mathematical problems which for centuries was the driving force of research in algebra. Let us only mention Galois theory, Gaussian elimination or Diophantine Equations. The last problem has been shown to be undecidable, however in finite realms such problems are obviously decidable in nondeterministic polynomial time.

A project of characterizing finite algebras $\mathbf{A}$ with polynomial time algorithms deciding satisfiability of circuits over $\mathbf{A}$ has been undertaken in [12]. Unfortunately that paper leaves a gap for nilpotent but not supernilpotent algebras. In this paper we discuss possible attacks on filling this gap.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic, Theory of computation → Problems, reductions and completeness, Theory of computation → Circuit complexity, Theory of computation → Constraint and logic programming, Mathematics of computing → Combinatorial algorithms

Keywords and phrases circuit satisfiability, solving equations, Constraint Satisfaction Problem, structure theory

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.17

Funding The project is partially supported by Polish NCN Grant # 2014/14/A/ST6/00138.

1 Introduction

Solving equations (or systems of equations) over an algebra $\mathbf{A}$ is one of the oldest and well-known mathematical problems which for centuries was the driving force of research in algebra. Let us only mention Galois theory, Gaussian elimination or Diophantine Equations. In fact, for $\mathbf{A}$ being the ring of integers this is the famous 10th Hilbert Problem on Diophantine Equations, which has been shown to be undecidable [17]. In finite realms such problems are obviously decidable in nondeterministic polynomial time.

The decision version of solving systems of equations is strictly connected with Constraint Satisfaction Problem for relational structures. A bisimulation between these two problems has been presented in [15] and [5]. Due to this bisimulation and the recent dichotomy results for CSP [2, 20] one can translate the beautiful splitting conditions into the language of satisfiability of systems of equations. Unfortunately such a bisimulation between satisfiability of a single equation (denoted by PolSat) and CSP is not known. Therefore the search for a characterization of finite algebras with tractable PolSat seems to be challenging. There were isolated results for particular algebraic structures like groups [7, 10, 9], rings [8] or lattices [19]. However, already a study of PolSat for groups has shown [11] that for the alternating group $\mathbf{A}_4$ the problem has a polynomial time solution in the pure group language, while it is NP-complete after endowing $\mathbf{A}_4$ with binary commutator operation (which is obviously definable by group multiplication). Here the polynomial time complexity is a result of artificial inflation of the size of input – indeed the terms used in proving NP-completeness are exponentially longer in pure group language than in the language allowing group commutator. This unwanted phenomena should be eliminated when one wants to get a characterization of abstract algebras with polynomial time procedures for satisfiability of single equations. This project has been taken over in [12] where a measure of the size of input is based on circuits representing terms/polynomials. This eliminates the mentioned exponential inflation of the size of terms by replacing all terms by circuits that computes them. For a detailed discussion of this we refer to [12]. Here we only recall definitions (in this new setting) of two problems based on PolSat and its dual.

- Csat($\mathbf{A}$)
  given a circuit over the algebra $\mathbf{A}$ with two output gates $\mathbf{g}_1, \mathbf{g}_2$ is there a valuation of input gates $\overline{x} = (x_1, \ldots, x_n)$ that gives the same output on both $\mathbf{g}_1$ and $\mathbf{g}_2$, i.e. $\mathbf{g}_1(\overline{x}) = \mathbf{g}_2(\overline{x})$ for some $\overline{x} \in A^n$.
- Ceqv($\mathbf{A}$)
  given a circuit over the algebra $\mathbf{A}$ is it true that for all inputs $\overline{x}$ we have the same values on given two output gates $\mathbf{g}_1, \mathbf{g}_2$, i.e. $\mathbf{g}_1(\overline{x}) = \mathbf{g}_2(\overline{x})$.

This new approach proved itself to be very useful as the mentioned results for groups, rings and lattices have been extended in [12] to much more general setting. Actually the main result of [12] shows that both nilpotent groups and distributive lattices (isolated in [7] and [19] as those with Ptime algorithm for PolSat) form a paradigm for algebras with Csat solvable in polynomial time. Unfortunately nilpotent groups hide an extremely interesting phenomena, namely every finite nilpotent group is a direct product of groups of prime power order. In universal algebraic terms this means that nilpotent groups are supernilpotent. In fact, one of the the results contained in [12] (independently also shown in [14]) generalizes polynomial time algorithms from nilpotent groups to supernilpotent algebras. On the other hand, the results of [12] leave a gap: solvable algebras considered in [12] that are not nilpotent have NP-complete Csat and co-NP-complete Ceqv, while the complexity of Csat and Ceqv for nilpotent but not supernilpotent algebras is not known. Unfortunately to fill this gap the

PTIME algorithms presented in [1, 12, 14] are useless – they do not work in general nilpotent setting. They rely on the fact that if an equation has a solution then it must have one among relatively small set S of tuples (although there may exist some other solutions outside the set S). More precisely: if $\mathbf{A}$ is a supernilpotent algebra (or a distributive lattice) then there is a constant d so that for each natural number n there is $S_n \subseteq A^n$ such that

- $|S_n|$ is $O(n^d)$,
- for two n-ary polynomials $\mathbf{s}$ and $\mathbf{t}$ the equation $\mathbf{s}(\overline{x}) = \mathbf{t}(\overline{x})$ has a solution $\overline{x} \in A^n$ iff it has a solution in S_n.

Also showing that two n-ary polynomials over an algebra determine the same n-ary function is reduced to checking this on a relatively small set S_n.

In this paper we show two things:

1. For nilpotent but not supernilpotent finite algebras one cannot expect polynomial time algorithms for CSAT or CEQV based on small search spaces S_n described above, (unless $\mathsf{P} = \mathsf{NP}$).
2. On the other hand there are finite nilpotent but not supernilpotent algebras with tractable CSAT and CEQV problems.

The examples mentioned in the second item are expanded abelian groups $(\mathbb{Z}_{pq}; +, f)$ where p, q are different primes, $+$ is the addition modulo pq and $f(x)$ is a unary function that returns x modulo p. Algorithms solving CEQV and CSAT for those algebras in polynomial time are described in sections 4 and 5, respectively. Those algorithms are based on a precise analysis of some kind of normal form of polynomials. The existence of such nice normal form is shown in section 3.

The evidence for the first item is provided by algebras with the very same clone of all polynomials as $(\mathbb{Z}_{pq}; +, f)$ but given by a different and infinite set of precisely chosen basic operations. In fact we show in section 6 that such algebras have co-NP-complete CEQV problem and NP-complete CSAT.

The reader should be warned here that leaving the safe realm, in which only finitely many basic operations are allowed, results in several fundamental problems. To start with, note that presenting an equation $\mathbf{s}(\overline{x}) = \mathbf{t}(\overline{x})$ we need to identify the basic operations that occur in $\mathbf{s}$ or $\mathbf{t}$. In fact, even the decidability of such redefined CSAT is not clear. One way to overcome this is to have an algebra $\mathbf{A} = (A; \mathbf{f}_0, \mathbf{f}_1, \ldots)$ encoded by a Turing machine $TM_{\mathbf{A}}$ which given the (k-ary) operation $\mathbf{f}_m$ and $a_1, \ldots, a_k \in A$ returns $TM_{\mathbf{A}}(m, a_1, \ldots, a_k) = \mathbf{f}_m(a_1, \ldots, a_k)$. Such approach puts extended CSAT($\mathbf{A}$) into NP whenever $TM_{\mathbf{A}}$ works in polynomial time. But it can be applied only to algebras with recursively enumerable set of fundamental operations.

The other way is to present an instance $\mathbf{s}(\overline{x}) = \mathbf{t}(\overline{x})$ of the problem together with the descriptions of all fundamental operations that occur in $\mathbf{s}$ or $\mathbf{t}$. Again such description may be done twofold:

- (CSAT$_{\mathrm{T}}$) by presenting the tables of the occurring basic operations, or
- (CSAT$_{\mathrm{TM}}$) by (polynomial time) algorithms $TM_{\mathbf{f}}$ computing the values $\mathbf{f}(a_1, \ldots, a_k)$.

It may seem that presenting operations by tables is more natural, as in many cases the complexity of such extended CSAT$_{\mathrm{T}}$ coincides with the complexity of its original (finite) version CSAT whenever the clone of an algebra is finitely generated (cf. Theorems 4.2, 5.2 and 6.2). On the other hand presenting the tables can again be treated as an artificial inflation of the input size. Indeed, Theorems 6.1 and 6.2 provide examples of nilpotent algebras with polynomial time CSAT$_{\mathrm{T}}$ and NP-complete CSAT$_{\mathrm{TM}}$. In fact item (1) above relies on such examples.

2 Background material

We use standard notation of universal algebra and computational complexity theory which can be found for example in [4], [18]. In particular, by an *algebra* we mean a pair $\mathbf{A} = (A; F)$, where A is a nonempty set and F is a family of finitary operations on A. Together with basic operations of $\mathbf{A}$, i.e. the operations from the set F, we often consider the derived operations – terms and polynomials of $\mathbf{A}$. By a polynomial of $\mathbf{A}$ we mean its term with some variables substituted by constants from A. We often refer to the syntactical side of the set F of operations as the type of $\mathbf{A}$. If $F = \{f_1, \ldots, f_s\}$ we simply write $(A; f_1, \ldots, f_s)$ rather than $(A; F)$ and say that $\mathbf{A}$ is of finite type. We restrict ourselves to finite algebras, i.e. algebras with finite universe A but we do allow infinite sets F of basic operations. The set of all terms (or polynomials) of $\mathbf{A}$ is to be denoted by $\mathrm{Clo}\,\mathbf{A}$ (or $\mathrm{Pol}\,\mathbf{A}$). Two algebras $\mathbf{A}$ and $\mathbf{B}$ are said to be *polynomially equivalent* if they have the same universes and $\mathrm{Pol}\,\mathbf{A} = \mathrm{Pol}\,\mathbf{B}$.

This paper is also restricted to algebras that belong to congruence modular varieties, i.e. to algebras $\mathbf{A}$ that together with all subalgebras $\mathbf{D}$ of the powers $\mathbf{A}^n$ having modular lattices $\mathsf{Con}\,\mathbf{D}$ of their congruences. Congruence modular varieties include most known and well-studied algebras such as groups, rings, modules (and their generalizations like quasigroups, loops, near-rings, nonassociative rings, Lie algebras), lattices (and their extensions like Boolean algebras, Heyting algebras or other algebras connected with multi-valued logics including MV-algebras).

One reason of our restriction to algebras from congruence modular varieties is that the paper [12] deals with such algebras. The other one is that in congruence modular setting there is a pretty well working notion of commutator of congruences that nicely generalizes commutator of normal subgroups (in group theory) and multiplication of ideals (in ring theory). A deep study of this commutator is described in [6]. Here we only recall a couple of definitions needed later. For congruences $\alpha, \beta, \gamma \in \mathsf{Con}\,\mathbf{A}$ we say that α *centralizes* β *modulo* γ (and denote this by $C(\alpha, \beta; \gamma)$) if for every $n \geqslant 1$, every $(n+1)$-ary term $\mathbf{t}$ of $\mathbf{A}$, every $(a, b) \in \alpha$, and every $(c_1, d_1), \ldots, (c_n, d_n) \in \beta$ we have

$$\mathbf{t}(a, \overline{c}) \overset{\gamma}{\equiv} \mathbf{t}(a, \overline{d}) \quad \text{iff} \quad \mathbf{t}(b, \overline{c}) \overset{\gamma}{\equiv} \mathbf{t}(b, \overline{d}).$$

Now the *commutator* $[\alpha, \beta]$ of the congruences $\alpha, \beta \in \mathsf{Con}\,\mathbf{A}$ is the smallest congruence $\gamma \in \mathsf{Con}\,\mathbf{A}$ for which $C(\alpha, \beta; \gamma)$. With the help of the commutator one can define solvable, nilpotent and Abelian congruences. In this paper we are interested only in the two last concepts. To define nilpotency we first iterate the commutator by putting $\theta^{(0)} = \theta$ and $\theta^{(i+1)} = [\theta, \theta^{(i)}]$, whenever $\theta \in \mathsf{Con}\,\mathbf{A}$. Now we say that θ is *k-nilpotent* if θ^k is the identity relation 0_A on $\mathbf{A}$ and the algebra $\mathbf{A}$ is *nilpotent* if the largest congruence 1_A of $\mathbf{A}$ is k-nilpotent for some positive integer k. As in group theory, $\mathbf{A}$ is called *Abelian* if 1_A is 1-nilpotent. Abelian algebras from congruence modular varieties have been shown in [6] to have particularly nice structure. In fact they are *affine*, i.e. polynomially equivalent to unitary modules (over a ring with unit).

We will also need the following strengthening of the nilpotency. First, for a bunch of congruences $\alpha_1, \ldots, \alpha_k, \beta, \gamma \in \mathsf{Con}\,\mathbf{A}$ we say that $\alpha_1, \ldots, \alpha_k$ centralize β modulo γ, and write $C(\alpha_1, \ldots, \alpha_k, \beta; \gamma)$, if for all polynomials $\mathbf{p} \in \mathrm{Pol}\,\mathbf{A}$ and all tuples $\overline{a}_1 \overset{\alpha_1}{\equiv} \overline{b}_1, \ldots, \overline{a}_k \overset{\alpha_k}{\equiv} \overline{b}_k$ and $\overline{u} \overset{\beta}{\equiv} \overline{v}$ such that

$$\mathbf{p}(\overline{x}_1, \ldots, \overline{x}_k, \overline{u}) \overset{\gamma}{\equiv} \mathbf{p}(\overline{x}_1, \ldots, \overline{x}_k, \overline{v})$$

for all possible choices of $(\overline{x}_1, \ldots, \overline{x}_k)$ in $\{\overline{a}_1, \overline{b}_1\} \times \ldots \times \{\overline{a}_k, \overline{b}_k\}$ but $(\overline{b}_1, \ldots . \overline{b}_k)$, we also have

$$\mathbf{p}(\overline{b}_1, \ldots, \overline{b}_k, \overline{u}) \overset{\gamma}{\equiv} \mathbf{p}(\overline{b}_1, \ldots, \overline{b}_k, \overline{v}).$$

This notion was introduced by A. Bulatov [3] and further developed by E. Aichinger and N. Mudrinski [1]. In particular they have shown that for all $\alpha_1, \dots, \alpha_k \in \mathsf{Con}\,\mathbf{A}$ there is the smallest congruence γ with $C(\alpha_1, \dots, \alpha_k; \gamma)$ called the k-ary commutator and denoted by $[\alpha_1, \dots, \alpha_k]$. Such generalized commutator behaves especially well in algebras from congruence modular varieties. In particular this commutator is monotone, join-distributive and we have $[\alpha_1, [\alpha_2, \dots, \alpha_k]] \leqslant [\alpha_1, \dots, \alpha_k]$. Thus every k-supernilpotent algebra, i.e. algebra satisfying $[\underbrace{1, \dots, 1}_{k+1 \text{ times}}] = 0$, is k-nilpotent. The following properties, that can be easily inferred from the deep work of R. Freese and R. McKenzie [6] and K. Kearnes [13], have been summarized in [1].

▶ **Theorem 2.1.** *For a finite algebra* $\mathbf{A}$ *from a congruence modular variety the following conditions are equivalent:*

1. $\mathbf{A}$ *is* k*-supernilpotent,*
2. $\mathbf{A}$ *is* k*-nilpotent, decomposes into a direct product of algebras of prime power order and the clone* $\operatorname{Clo}\mathbf{A}$ *is generated by finitely many operations,*
3. $\mathbf{A}$ *is* k*-nilpotent and all commutator polynomials have rank at most* k*.*

Commutator polynomials mentioned in condition (3) of Theorem 2.1 are the paradigms for the failure of supernilpotency and are easily seen to be useful in coding a k-ary conjunction. We say that $\mathbf{t}(x_1, \dots, x_{k-1}, z) \in \operatorname{Pol}_k\mathbf{A}$ is a commutator polynomial of rank k if

- $\mathbf{t}(a_1, \dots, a_{k-1}, b) = b$ whenever $b \in \{a_1, \dots, a_{k-1}\} \subseteq A$,
- $\mathbf{t}(a_1, \dots, a_{k-1}, b) \neq b$ for some $a_1, \dots, a_{k-1}, b \in A$.

As we have mentioned in the Introduction this paper is intended to give a better understanding of the problems Csat and Ceqv for nilpotent but not supernilpotent algebras. Like in the groups, in nilpotent setting the inputs $\mathbf{g}_1(\overline{x}) = \mathbf{g}_2(\overline{x})$ of Csat or Ceqv can be restricted to the ones in which one of the polynomials is constant, i.e. of the form $\mathbf{g}(\overline{x}) = c$. Indeed, with the help of Corollary 7.4 in [6] it suffices to choose any $c \in A$ and put $\mathbf{g}(\overline{x}) = \mathbf{m}(\mathbf{g}_1(\overline{x}), \mathbf{g}_2(\overline{x}), c)$, where $\mathbf{m}(x, y, z)$ is a Mal'cev term for $\mathbf{A}$, i.e. a term satisfying $\mathbf{m}(x, x, y) = y = \mathbf{m}(y, x, x)$ for all $x, y \in A$.

3 The structure of 2-nilpotent algebras

To fill the nilpotent versus supernilpotent gap mentioned in the Introduction we need to understand the structure of nilpotent algebras. Since all algebras considered in this paper are 2-nilpotent we will use the description of their structure presented in Chapter VII of [6]. It reduces to an action of one abelian algebra over the other abelian one. Thus, after fixing the set F of operations we need two affine algebras:

- an upper one, say $\mathbf{U}$, which is polynomially equivalent to a module $(U; \oplus)$ over a ring $\mathbf{R}_U$, and
- a lower one, say $\mathbf{L}$, which is polynomially equivalent to a module $(L; +)$ over a ring $\mathbf{R}_L$,

and for each basic operation $f \in F$, say k-ary, we need a function $\widehat{f} : U^k \longrightarrow L$. This allows us to construct an algebra $\mathbf{L} \otimes^F \mathbf{U}$ of type F on the set $L \times U$ by putting

$$f^{\mathbf{L}\otimes^F\mathbf{U}}((l_1, u_1), \dots, (l_k, u_k)) = (f^{\mathbf{L}}(l_1, \dots, l_k) + \widehat{f}(u_1, \dots, u_k), f^{\mathbf{U}}(u_1, \dots, u_k)). \tag{1}$$

The usefulness of this construction is described in Corollary 7.2 in [6], where every 2-nilpotent algebra of type F is shown to be of the form $\mathbf{L} \otimes^F \mathbf{U}$ for some triple $\mathbf{U}, \mathbf{L}, \left\{\widehat{f} : f \in F\right\}$. The rest of this section is devoted to presenting a nice normal form of arbitrary polynomial $\mathbf{p}$ of $\mathbf{A}$ that will be useful to construct polynomial time algorithms for Ceqv($\mathbf{A}$) and Csat($\mathbf{A}$).

To start with, note that the equation (1) remains valid for f being not just a basic operation but an arbitrary polynomial of $\mathbf{A}$, where $\widehat{f}$ is appropriately chosen. Moreover having in mind that both $\mathbf{L}$ and $\mathbf{U}$ are affine we know that $f^{\mathbf{L}}(l_1,\dots,l_k)$ and $f^{\mathbf{U}}(u_1,\dots,u_k)$ are affine combinations $\sum_{i=1}^{k}\lambda_i l_i + l_0$ and $\bigoplus_{i=1}^{k}\alpha_i u_i \oplus u_0$. Thus for a polynomial $\mathbf{p}$ we have

$$\mathbf{p}^{\mathbf{L}\otimes^F\mathbf{U}}((l_1,u_1),\dots,(l_k,u_k)) = \left(\sum_{i=1}^{k}\lambda_i l_i + \widehat{\mathbf{p}}(u_1,\dots,u_k), \bigoplus_{i=1}^{k}\alpha_i u_i + u_0\right),$$

where l_0 is absorbed by $\widehat{\mathbf{p}}$. Moreover, $\widehat{\mathbf{p}}(u_1,\dots,u_k)$ can be presented as a sum of elements of the form

$$\mu\cdot\widehat{g}\left(\bigoplus_{i=1}^{k}\beta_i^{(1)}u_i\oplus u_0^{(1)},\dots,\bigoplus_{i=1}^{k}\beta_i^{(s)}u_i\oplus u_0^{(s)}\right),$$

with $\widehat{g}$ ranging over basic operations (and its occurrences) used to build $\mathbf{p}$.

The normal form, we have just started to build, has particularly nice shape in the cases where the $\widehat{g}$'s can be presented as affine combinations of unary $\widehat{f}$'s. One of such case is presented in the following Lemma.

▶ **Lemma 3.1.** *Let* $\mathbf{U}$ *and* $\mathbf{L}$ *be algebras polynomially equivalent to 1-dimentional vector spaces over prime fields of different characteristics. Moreover let* $f: U\longrightarrow L$ *be such that* $f(0_{\mathbf{U}})=0_{\mathbf{L}}$ *and* $\sum_{u\in U}f(u)\neq 0_{\mathbf{L}}$*. Then, every function* $g\colon U^k\longrightarrow L$ *can be expressed by*

$$g(x_1,\dots,x_k) = \sum_{(\overline{\beta},u)\in F_U^k\times U}\mu_{\overline{\beta},u}\cdot f\left(\bigoplus_{i=1}^{k}\beta_i x_i\oplus u\right). \tag{2}$$

Proof. Let $h^k_{a_1,\dots,a_k,a}\colon U^k\longrightarrow L$ be constantly $0_{\mathbf{L}}$ except $h^k_{a_1,\dots,a_k,a}(a_1,\dots,a_k)=a$. Observe that for every function $g\colon U^k\longrightarrow L$ we have

$$g(x_1,\dots,x_k) = \sum_{(a_1,\dots,a_k)\in U^k} h^k_{a_1,\dots,a_k,g(a_1,\dots,a_k)}(x_1,\dots,x_k).$$

Moreover we can express one spike function by any other one by putting

$$h^k_{a_1,\dots,a_k,a}(x_1,\dots,x_k) = a\cdot b^{-1}\cdot h^k_{b_1,\dots,b_k,b}(x_1-a_1+b_1,\dots,x_k-a_k+b_k).$$

In the above we use 1-dimensionality so that the universes of $\mathbf{L}$ and $\mathbf{F}_L$ coincide, so that the vectors $a,b\in L$ can be also treated as scalars from $\mathbf{F}_L$.

The last two displays yield that to finish the proof of the Lemma it suffices to represent, for each k, one spike function in the form described in (2). The rest of the proof shows how this can be done in the presence of a unary function f described in the Lemma, which is to be called $(\mathbf{U},\mathbf{L})$-normal in the rest of the paper. Our construction of such spike functions is based on counting partitions (into special sums) of elements in vector spaces over finite fields.

Being left with representing one spike function of arbitrary large arity by an expression of the form (2) we start with letting p and q to be characteristics of the fields $\mathbf{F}_U$ and $\mathbf{F}_L$, respectively. Moreover let $f\colon U\longrightarrow L$ be $(\mathbf{U},\mathbf{L})$-normal . Put

$$t_s(x_1,\dots,x_s) = \sum_{\emptyset\neq I\subseteq\{1,\dots,s\}}(-1)^{|I|}f\left(\bigoplus_{i\in I}x_i\right)$$

and note that $t_s(x_1,\dots,x_s)=0_{\mathbf{L}}$ whenever $0_{\mathbf{U}}\in\{x_1,\dots,x_s\}$. We fix an enumeration of the set $U=\{0_{\mathbf{U}},u_1,\dots,u_{p-1}\}$ and define

$$w_k(x_1,\dots,x_k)=t_{k(p-1)}(x_1-u_1,\dots,x_k-u_1,x_1-u_2,\dots,x_k-u_2,\dots,x_1-u_{p-1},\dots,x_k-u_{p-1}).$$

Observe that $w_k(\overline{x})=0_{\mathbf{L}}$ whenever at least one of the x_i's is non-zero. To prove that w_k is a spike function it remains to show $w_k(0_{\mathbf{U}},\dots,0_{\mathbf{U}})\neq 0_{\mathbf{L}}$. Our first claim is

$$w_k(0_{\mathbf{U}},\dots,0_{\mathbf{U}}) \quad = \quad t_{p-1}(u_1,\dots,u_{p-1}). \tag{3}$$

Since t_s is fully symetric, we have

$$w_k(0_{\mathbf{U}},\dots,0_{\mathbf{U}})=t_{k(p-1)}(\overbrace{u_1,\dots,u_1}^{k\text{ times}},\overbrace{u_2,\dots,u_2}^{k\text{ times}},\dots,\overbrace{u_{p-1},\dots,u_{p-1}}^{k\text{ times}}).$$

Without loss of generality we may assume that $k=q^a$, as for any other k' to get a k'-ary spike we choose the smallest power $q^a\geqslant k'$ and replace q^a-k' arguments with $0_{\mathbf{U}}$. From the definition of $t_{k(p-1)}$ we get

$$\begin{aligned}&t_{k(p-1)}(u_1,\dots,u_1,u_2,\dots,u_2,\dots,u_{p-1},\dots,u_{p-1})=\\&=\sum_{k=1}^{(p-1)\cdot q^a}(-1)^k\sum_{\substack{k_1+\dots+k_{p-1}=k\\k_i\leqslant q^a}}\binom{q^a}{k_1}\cdot\ldots\cdot\binom{q^a}{k_{p-1}}f\left(\bigoplus_{i=1}^{k_1}u_1\oplus\ldots\oplus\bigoplus_{i=1}^{k_{p-1}}u_{p-1}\right).\end{aligned}$$

Observe that $\binom{q^a}{k_i}$ is divisible by q whenever $k_i\notin\{0,q^a\}$. Thus the only summands that do not vanish are those with $k_i\in\{0,q^a\}$ for all i so that $\binom{q^a}{k_1}\cdot\ldots\cdot\binom{q^a}{k_{p-1}}=1$. Thus, by changing notation, we have

$$\begin{aligned}&t_{k(p-1)}(u_1,\dots,u_1,u_2,\dots,u_2,\dots,u_{p-1},\dots,u_{p-1})=\\&=\sum_{k'=1}^{p-1}(-1)^{k'\cdot q^a}\sum_{\substack{k'_1+\dots+k'_{p-1}=k'\\k'_i\in\{0,1\}}}f\left(\bigoplus_{i=1}^{k'_1\cdot q^a}u_1\oplus\ldots\oplus\bigoplus_{i=1}^{k'_{p-1}\cdot q^a}u_{p-1}\right)=\\&=\sum_{k'=1}^{p-1}(-1)^{k'\cdot q^a}\sum_{\substack{S\subseteq\{1,\dots,p-1\}\\|S|=k'}}f\left(\bigoplus_{i\in S}u_i\right),\end{aligned}$$

where the last equality follows from the fact that the multisets

$$\left\{\left\{\bigoplus_{i=1}^{k'_1\cdot q^a}u_1\oplus\ldots\oplus\bigoplus_{i=1}^{k'_{p-1}\cdot q^a}u_{p-1}\colon k'_1+\ldots+k'_{p-1}=k',k'_i\in\{0,1\}\right\}\right\}$$

and

$$\left\{\left\{\bigoplus_{i\in S}u_i\colon S\subseteq\{1,\dots,p-1\},|S|=k'\right\}\right\}.$$

are equal. To complete the proof of (3) observe that $(-1)^{k'\cdot q^a}=(-1)^{k'}$. This is obvious for q being odd, while otherwise it follows from the fact that $x=-x$.

We will conclude our proof by showing that

$$w_k(0_{\mathbf{U}},\dots,0_{\mathbf{U}})\neq 0_{\mathbf{L}}. \tag{4}$$

From the definition of t_s we know that

$$t_{p-1}(u_1,\dots,u_{p-1}) = \sum_{k=1}^{p-1}(-1)^k \sum_{\substack{S\subseteq\{1,\dots,p-1\}\\ |S|=k}} f\left(\bigoplus_{i\in S} u_i\right).$$

Now, if $\ell^k_{u_i}$ denotes a number of partition of the element u_i into a sum of k non-zero pairwise different elements from U then by appropriately grouping the $f\left(\bigoplus_{i\in S} u_i\right)$'s and noting that $\mathbf{f}(0_\mathbf{U}) = 0_\mathbf{L}$ we can replace the last sum by

$$\sum_{k=1}^{p-1}(-1)^k \sum_{i=1}^{p-1} \ell^k_{u_i} f(u_i)$$

The numbers $\ell^k_{u_i}$ were calculated in [16] to be $\ell^k = \ell^k_{u_i} = \frac{1}{p}\left(\binom{p-1}{k} + (-1)^{k+1}\right)$ independently of $u_i \neq 0_\mathbf{U}$. This gives that

$$t_{p-1}(u_1,\dots,u_{p-1}) = \sum_{k=1}^{p-1}(-1)^k \ell^k \sum_{i=1}^{p-1} f(u_i) = \left(\sum_{k=1}^{p-1}(-1)^k \ell^k\right)\left(\sum_{i=1}^{p-1} f(u_i)\right).$$

Using the explicit formulas for the ℓ^k's we get

$$\begin{aligned}
\sum_{k=1}^{p-1}(-1)^k \ell^k &= \frac{1}{p}\sum_{k=1}^{p-1}(-1)^k\left[\binom{p-1}{k} + (-1)^{k+1}\right]\\
&= \frac{1}{p}\sum_{k=1}^{p-1}\left[(-1)^k\binom{p-1}{k} + (-1)^{2k+1}\right]\\
&= \frac{1}{p}\left[\sum_{k=1}^{p-1}(-1)^k\binom{p-1}{k} + \sum_{k=1}^{p-1}(-1)\right]\\
&= \frac{1}{p}\left[-1-(p-1)\right] = -1
\end{aligned}$$

This gives us that

$$w_k(0_\mathbf{U},\dots,0_\mathbf{U}) = t_{p-1}(u_1,\dots,u_{p-1}) = -\sum_{i=1}^{p-1} f(u_i)$$

is not equal to $0_\mathbf{L}$ as f is $(\mathbf{U},\mathbf{L})$-normal . Thus the claim (4) holds, proving that w_k is a spike function, as required. ◀

Lemma 3.1 yields that, in its setting, every polynomial $\mathbf{p}$ of $\mathbf{A} = \mathbf{L} \otimes^F \mathbf{U}$ can be represented as

$$\mathbf{p}((l_1,u_1),\dots,(l_k,u_k)) = \left(\sum_{i=1}^{k}\lambda_i l_i + \sum_{\substack{\overline{\beta}\in R_U^k\\ u\in U}} \mu_{\overline{\beta},u}\cdot f\left(\bigoplus_{i=1}^{k}\beta_i u_i \oplus u\right), \bigoplus_{i=1}^{k}\alpha_i u_i \oplus u_0\right) \quad (5)$$

where f is a $(\mathbf{U},\mathbf{L})$-normal function. Representations of the above form are to be called f-normal. The size of such normal form is essentially the number of non-zero coefficients among the λ_i's, $\mu_{\overline{\beta},u}$'s and α_i's.

Most of our arguments in this paper refer to f-normal forms where the modules hidden in $\mathbf{U}$ and $\mathbf{L}$ are actually 1-dimensional vector spaces over prime fields of different characteristics. However f-normal forms could be useful in more general settings as shown in the following Lemma as well as in Lemma 4.1.

▶ **Lemma 3.2.** *Let* $\mathbf{A} = \mathbf{L} \otimes^F \mathbf{U}$ *be a finite 2-nilpotent algebra. Then for each* $f\colon U \longrightarrow L$ *there exists a polynomial time procedure that returns* f*-normal form for a polynomial* $\mathbf{g}$ *of* $\mathbf{A}$ *presented together with* f*-normal forms of all the basic operations occurring in* $\mathbf{g}$*.*

Proof. To prove the lemma we start with two polynomials $\mathbf{h}_1, \mathbf{h}_2$ of $\mathbf{A}$ presented in their f-normal forms to carefully compute f-normal form of the superposition

$$\mathbf{g}(x_1, \dots, x_{k_1+k_2-1}) = \mathbf{h}_2(\mathbf{h}_1(x_1, \dots, x_{k_1}), x_{k_1+1}, \dots, x_{k_1+k_2-1})$$

in a polynomial time. Let

$$\mathbf{h}_j((l_1, u_1), \dots, (l_{k_j}, u_{k_j})) = \left(\sum_{i=1}^{k_j} \lambda_i^{(j)} l_i + \sum_{(\overline{\beta}, u) \in \Gamma_j} \mu^{(j)}_{\overline{\beta}, c} \cdot f\left(\bigoplus_{i=1}^{k_j} \beta_i u_i \oplus u \right), \bigoplus_{i=1}^{k_j} \alpha_i^{(j)} u_i \oplus u_0^{(j)} \right)$$

for appropriate $\Gamma_j \subseteq \mathbf{R}_U^{k_j} \times U$.
Now the second and the first coordinates of $\mathbf{g}((l_1, u_1), \dots, (l_{k_1+k_2-1}, u_{k_1+k_2-1}))$ are easily seen to be

$$\bigoplus_{i=1}^{k_1} \alpha_1^{(2)} \alpha_i^{(1)} u_i \oplus \bigoplus_{i=2}^{k_2} \alpha_i^{(2)} u_{i+k_1-1} \oplus \alpha_1^{(2)} u_0^{(1)} \oplus u_0^{(2)},$$

and

$$\sum_{i=1}^{k_1} \lambda_1^{(2)} \lambda_i^{(1)} l_i + \sum_{i=2}^{k_2} \lambda_i^{(2)} l_{i+k_1-1} + \lambda_1^{(2)} \cdot \sum_{(\overline{\beta}, u) \in \Gamma_1} \mu^{(1)}_{\overline{\beta}, u} \cdot f\left(\bigoplus_{i=1}^{k_1} \beta_i u_i \oplus u \right) + g'(\overline{u}),$$

where $g'(u_1, \dots, u_{k_1+k_2-1})$ is obtained from $\sum_{(\overline{\beta}, u) \in \Gamma_2} \mu^{(2)}_{\overline{\beta}, u} \cdot f\left(\beta_1 u_1 \oplus \bigoplus_{i=2}^{k_2} \beta_i u_{k_1+i-1} \oplus u \right)$ by substituting every occurrence of u_1 by $\bigoplus_{i=1}^{k_1} \alpha_i^{(1)} u_i \oplus u_0^{(1)}$.

It should be obvious that both these coordinates give f-normal form of $\mathbf{g}$ and that they can be computed in a polynomial time in size of f-normal forms of $\mathbf{h}_1$ and $\mathbf{h}_2$. Actually a careful inspection of our argument allows us to bound the number of non-zero summands so that the f-normal forms of more complicated superpositions of polynomials of $\mathbf{A}$ can be also computed efficiently. ◀

Combining Lemmas 3.1 and 3.2 we get

▶ **Corollary 3.3.** *Let* $\mathbf{L}$ *and* $\mathbf{U}$ *be algebras polynomially equivalent to one dimensional vectors spaces over prime fields of different characteristics and* $f\colon U \longrightarrow L$ *be* $(\mathbf{U}, \mathbf{L})$*-normal . Then every polynomial operation of* $\mathbf{A} = \mathbf{L} \otimes^F \mathbf{U}$ *has* f*-normal forms and, if the type* F *of* $\mathbf{A}$ *is finite then one of these* f*-normal forms can be computed in polynomial time.*

4 Equivalence

The last paragraph of Section 2 shows that in the nilpotent setting in CEQV it suffices to consider equivalence of two polynomials one of which is constant.

▶ **Lemma 4.1.** *Let* $\mathbf{A} = \mathbf{L} \otimes^F \mathbf{U}$ *with* $\mathbf{L}$ *and* $\mathbf{U}$ *being polynomially equivalent to one-dimensional vector spaces over finite fields of different characteristics. Then there exists a polynomial time algorithm which for polynomials* $\mathbf{p}$ *of* $\mathbf{A}$ *given in some* f*-normal form decides if* $\mathbf{p}$ *is a constant function.*

Proof. Let

$$\mathbf{p}((l_1,u_1),\dots,(l_k,u_k)) = \left(\mathbf{p}^{\mathbf{L}}(l_1,\dots,l_k) + \sum_{\substack{\overline{\beta}\in\mathbf{F}_U^k\\ u\in U}} \mu_{\overline{\beta},u}\cdot f\left(\bigoplus_{i=1}^k \beta_i u_i \oplus u\right), \mathbf{p}^{\mathbf{U}}(u_1,\dots,u_k)\right)$$

be an f-normal form of $\mathbf{p}$. Obviously such polynomial $\mathbf{p}$ is constant iff both coordinates of the right-hand side above are constant. This can be efficiently checked for $\mathbf{p}^{\mathbf{U}}(u_1,\dots,u_k)$ as it is simply a polynomial of a vector space. Since both summands of the first coordinate on the right-hand side depend on disjoint sets of variables (the l_i's and the u_i's) to keep their sum constant we need to keep both summands constant. Again, checking that for $\mathbf{p}^{\mathbf{L}}$ (in a vector space) is fast so that we are left with the expression of the form

$$\widehat{\mathbf{p}}(u_1,\dots,u_k) = \sum_{\substack{(\overline{\beta},u)\in\mathbf{F}_U^k\times U\\ \overline{\beta}\neq\overline{0}}} \mu_{\overline{\beta},u}\cdot f\left(\bigoplus_{i=1}^k \beta_i u_i \oplus u\right). \tag{6}$$

This in turn can be done with the help of the following claim.

($\star$) $\widehat{\mathbf{p}}$ is constant iff for each $\overline{\beta}\in\mathbf{F}_U^k\setminus\{\overline{0}\}$ the function $S_{\overline{\beta}}(x) = \sum_{(\kappa,u)\in F_U^*\times U}\mu_{\kappa\cdot\overline{\beta},u}\cdot f(\kappa x\oplus u)$ is constant on U.

Indeed, having ($\star$) we argue as follows. If for a particular $\overline{\beta}$ all the $\mu_{\kappa\cdot\overline{\beta},u}$'s are zero then $S_{\overline{\beta}}(x)=0$ for all $x\in U$. For any other $\overline{\beta}$ we simply check if $S_{\overline{\beta}}$ is constant by computing all of the $|U|$ values for the x's. This is fast as we have at most $|U|^2$ summands. Finally, the number of the $\overline{\beta}$'s of the second kind is linear in the number of non-zero coefficients $\mu_{\overline{\beta},u}$. This in turn is bounded by the length of the expression (6) which is the part of the input.

Thus we are left with the proof of ($\star$). To see the 'only if' direction for $\overline{\beta}\in\mathbf{F}_{\mathbf{U}}^k\setminus\{\overline{0}\}$ and $a\in U$ define $\mathbf{O}_{a,\overline{\beta}} = \{\overline{u}\in U^k : \bigoplus_{i=1}^k\beta_i u_i = a\}$. Observe that the size of the solution set $\mathbf{O}_{a,\overline{\beta}}$ of a nontrivial linear equation is always $|U|^{k-1}$ independently of the choice of $(a,\overline{\beta})$. Now, since $\widehat{\mathbf{p}}$ is constant, for each $\overline{\beta}\in F_U^k\setminus\{0\}^k$ and $a,b\in L$ we have

$$\begin{aligned}
0 &= \sum_{\overline{u}\in\mathbf{O}_{a,\overline{\beta}}}\widehat{\mathbf{p}}(\overline{u}) - \sum_{\overline{u}\in\mathbf{O}_{b,\overline{\beta}}}\widehat{\mathbf{p}}(\overline{u})\\
&= \sum_{\overline{u}\in\mathbf{O}_{a,\overline{\beta}}}\sum_{\substack{(\overline{\gamma},u)\in\mathbf{F}_U^k\times U\\ \overline{\gamma}\neq\overline{0}}}\mu_{\overline{\gamma},u}\cdot f\left(\bigoplus_{i=1}^k\gamma_i u_i\oplus u\right) - \sum_{\overline{u}\in\mathbf{O}_{b,\overline{\beta}}}\sum_{\substack{(\overline{\gamma},u)\in\mathbf{F}_U^k\times U\\ \overline{\gamma}\neq\overline{0}}}\mu_{\overline{\gamma},u}\cdot f\left(\bigoplus_{i=1}^k\gamma_i u_i\oplus u\right)\\
&= \sum_{\substack{(\overline{\gamma},u)\in\mathbf{F}_U^k\times U\\ \overline{\gamma}\neq\overline{0}}}\mu_{\overline{\gamma},u}\cdot\left(\sum_{\overline{u}\in\mathbf{O}_{a,\overline{\beta}}} f\left(\bigoplus_{i=1}^k\gamma_i u_i\oplus u\right) - \sum_{\overline{u}\in\mathbf{O}_{b,\overline{\beta}}} f\left(\bigoplus_{i=1}^k\gamma_i u_i\oplus u\right)\right)\\
&= \sum_{\substack{(\overline{\gamma},u)\in\mathbf{F}_U^k\times U\\ \overline{\gamma}\neq\overline{0}}}\mu_{\overline{\gamma},u}\cdot\left(t(a,\overline{\beta},\overline{\gamma},u) - t(b,\overline{\beta},\overline{\gamma},u)\right),
\end{aligned}$$

where $t(x,\overline{\beta},\overline{\gamma},u) = \sum_{\overline{u}\in\mathbf{O}_{x,\overline{\beta}}} f\left(\bigoplus_{i=1}^k\gamma_i u_i\oplus u\right)$. Now, if the vectors $\overline{\beta},\overline{\gamma}\in\mathbf{F}_U^k$ are linearly dependent, i.e. $\overline{\gamma}=\kappa\cdot\overline{\beta}$ we have $t(x,\overline{\beta},\overline{\gamma},u) = |U|^{k-1}\cdot f(\kappa\cdot x\oplus u)$. Otherwise, if $\overline{\beta}$ and $\overline{\gamma}$ are linearly independent then $t(x,\overline{\beta},\overline{\gamma},u) = |U|^{k-2}\sum_{d\in U} f(d)$, as the system of the following

two equations

$$\begin{cases} \bigoplus_{i=1}^{k} \beta_i u_i = x \\ \bigoplus_{i=1}^{k} \gamma_i u_i \oplus u = d \end{cases}$$

has exactly $|U|^{k-2}$ solutions. Summing up in the big display above the summands with $\overline{\gamma}$'s that are linearly independent with $\overline{\beta}$ diminishes so that this display reduces to

$$\begin{aligned} 0 &= \sum_{\substack{(\overline{\gamma},u)\in \mathbf{F}_U^k \times U \\ \overline{\gamma}\neq\overline{0}}} \mu_{\overline{\gamma},u} \cdot \big(t(a,\overline{\beta},\overline{\gamma},u) - t(b,\overline{\beta},\overline{\gamma},u)\big) \\ &= |U|^{k-1} \cdot \left(\sum_{(\kappa,u)\in \mathbf{F}_U^* \times U} \mu_{\kappa\cdot\overline{\beta},u} f(\kappa\cdot a \oplus u) - \sum_{(\kappa,u)\in \mathbf{F}_U^* \times U} \mu_{\kappa\cdot\overline{\beta},u} f(\kappa\cdot b \oplus u) \right). \end{aligned}$$

Since $|U|$ and $|L|$ are coprime, the difference in the parenthesis is zero which shows the 'only if' direction of $(\star)$.

To prove the 'if' direction observe that $R = \{(\overline{\beta},\kappa\overline{\beta})\colon \overline{\beta}\in F_U^k \setminus \{\overline{0}\}, \kappa\in \mathbf{F}_U^*\}$ is an equivalence relation. Let $\overline{\beta}^{(1)},\overline{\beta}^{(2)},\dots,\overline{\beta}^{(m)}$ be the transversal of R. Then for $\overline{x}\in U^k$ we have

$$\begin{aligned} \widehat{\mathbf{p}}(x_1,\dots,x_k) &= \sum_{\substack{(\overline{\gamma},u)\in F_{\mathbf{U}}^k \times U \\ \overline{\gamma}\neq\overline{0}}} \mu_{\overline{\gamma},u} \cdot f\left(\bigoplus_{i=1}^{k} \gamma_i \cdot x_i \oplus u\right) \\ &= \sum_{j\in\{1,\dots,m\}} \sum_{(\kappa,u)\in F_{\mathbf{U}}^* \times U} \mu_{\kappa\cdot\overline{\beta}^{(j)},u} \cdot f\left(\kappa\cdot\bigoplus_{i=1}^{k} \beta_i^{(j)} x_i \oplus u\right) \\ &= \sum_{j\in\{1,\dots,m\}} S_{\overline{\beta}^{(j)}}\left(\bigoplus_{i=1}^{k} \beta_i^{(j)} x_i\right). \end{aligned}$$

Now our assumption that all the $S_{\overline{\beta}^{(j)}}$ are constant shows that $\widehat{\mathbf{p}}$ is constant, as well. ◀

Corollary 3.3 together with Lemma 4.1 immediately give the following theorem.

▶ **Theorem 4.2.** *Let* $\mathbf{A} = \mathbf{L} \otimes^F \mathbf{U}$ *with* $\mathbf{L}$ *and* $\mathbf{U}$ *being polynomially equivalent to* 1*-dimensional vector spaces over prime fields of different characteristics. Then* $\textsc{Ceqv}(\mathbf{A})$ *is in P.*

5 Satisfiability

Again, in nilpotent realm the last paragraph of Section 2 allows us to fix one side of the equations considered in Csat to be a constant polynomial.

▶ **Lemma 5.1.** *Let* $\mathbf{A} = \mathbf{L} \otimes^F \mathbf{U}$ *with* $\mathbf{L}$ *and* $\mathbf{U}$ *being polynomially equivalent to one-dimensional vector spaces over finite fields of different characteristics. Then there exists a polynomial time algorithm which for a constant* $c\in A$ *and polynomials* $\mathbf{p}$ *of* $\mathbf{A}$ *given in some* f*-normal form decides if the equation* $\mathbf{p}(\overline{x}) = c$ *has a solution.*

Proof. Since our polynomial is given in f-normal form we start with the following equation

$$\left(p^{\mathbf{L}}(l_1,\dots,l_k) + \sum_{(\overline{\beta},u)\in \mathbf{F}_U^k \times U} \mu_{\overline{\beta},u} \cdot f\left(\bigoplus_{i=1}^{k} \beta_i \cdot u_i \oplus u\right), p^{\mathbf{U}}(u_1,\dots,u_k) \right) = (c_L, c_U). \quad (7)$$

We start with observing that since $\mathbf{L}$ and $\mathbf{U}$ are polynomially equivalent to one-dimensional vector spaces the range of $p^{\mathbf{L}}$ (and $p^{\mathbf{U}}$) is either one element or the entire L (or U). Now, if $p^{\mathbf{L}}$ is not constant then it suffices to check whether $p^{\mathbf{U}}(u_1,\dots,u_k) = c_U$ has a solution in U, as the solution in first coordinate always exists. Thus we assume that $p^{\mathbf{L}}$ is constant and put $d = c_L - p^{\mathbf{L}}(0_{\mathbf{L}},\dots,0_{\mathbf{L}})$. Moreover we may assume that $p^{\mathbf{U}}$ is not constant or equal to c_U, as otherwise our equation has no solution.

We want to reduce our equation in $\mathbf{A}$ to an equivalent equation of the form

$$\sum_{(\overline{\beta},u)\in\mathbf{F}_U^k\times U} \nu_{\overline{\beta},u}\cdot f\left(\bigoplus_{i=1}^{k}\beta_i\cdot u_i\oplus u\right) \;=\; d \tag{8}$$

in $\mathbf{L}$, but with the u_i's taking values in U. Now, if $p^{\mathbf{U}}$ is constant (and therefore equal to c_U) then we are done with $\nu_{\overline{\beta},u} = \mu_{\overline{\beta},u}$ for all $\overline{\beta}$'s and u's. Otherwise $p^{\mathbf{U}}(u_1,\dots,u_k) = c_U$ reduces to something of the form $u_j = \bigoplus_{i\neq j}\delta_i u_i \oplus c_U$. This allows us to replace all occurrences of the u_j by the sum $\bigoplus_{i\neq j}\delta_i u_i\oplus c_U$ and after recalculating the coefficients $\mu_{\overline{\beta},u}$ we get the desired $\nu_{\overline{\beta},u}$'s as required in (8).

Denote the left-hand side of (8) by $p'(u_1,\dots,u_k)$ and note that since the set L carries the multiplication inherited from the field $\mathbf{F}_L$ the equation $p'(u_1,\dots,u_k) = d$ has a solution iff

$$\prod_{d'\in L-\{d\}} (p'(u_1,\dots,u_k) - d') = 0_{\mathbf{L}}$$

is not an identity. Note however that the product above is not directly expressible as a polynomial of $\mathbf{A}$. Nevertheless distributing over the factors we can replace the product by the sum

$$\sum_{(\overline{\beta}^{(1)},u^{(1)})}\sum_{(\overline{\beta}^{(2)},u^{(2)})}\cdots\sum_{(\overline{\beta}^{(m)},u^{(m)})} \nu_{\mathcal{I}}\cdot f\left(\bigoplus_{i=1}^{k}\beta_i^{(1)}u_i\oplus u^{(1)}\right)\cdot\ldots\cdot f\left(\bigoplus_{i=1}^{k}\beta_i^{(m)}u_i\oplus u^{(m)}\right),$$

where $\mathcal{I} = \left(\overline{\beta}^{(1)},u^{(1)},\dots,\overline{\beta}^{(m)},u^{(m)}\right)$ and $m = |U|-1$. Lemma 3.1 supplies us with a representation of the product $f(x_1)\cdot\ldots\cdot f(x_m)$ sending U^m into L by an expression of the right-hand side of (2). This leads to the representation of the form

$$\prod_{d'\in L-\{d\}} (p'(u_1,\dots,u_k) - d') = \sum_{(\overline{\beta},u)\in\mathbf{F}_U^n\times U} \mu'_{\overline{\beta},u}\cdot f\left(\bigoplus_{i=1}^{k}\beta_i\cdot u_i\oplus u\right)$$

which, with the help of Lemma 4.1, can be efficiently checked not to be constantly $0_{\mathbf{L}}$. This in turn is equivalent for the starting equation to have a solution. ◀

Corollary 3.3 and Lemma 5.1 yield the following result.

▶ **Theorem 5.2.** *Let* $\mathbf{A} = \mathbf{L}\otimes^F\mathbf{U}$ *with* $\mathbf{L}$ *and* $\mathbf{U}$ *being polynomially equivalent to one dimensional vectors spaces over prime fields of different characteristics. Then* $\textsc{Csat}(\mathbf{A})$ *is in P.*

6 Algebras with infinitely many operations

The reasons we consider infinite languages in this paper are twofold. One motivation comes from the fact that a desire for a simple extension of our polynomial time algorithms from supernilpotent algebras to nilpotent ones is hopeless. Indeed, as it has been already mentioned

in the Introduction, those algorithms are based on a reduction to a small search space. Its size and shape is bounded as a result of the bound for the essential arity of commutator terms (they serve as multi-ary internal conjunctions replacing lack of an external one). At first glance the existence of polynomial time algorithms for nilpotent but not supernilpotent algebras is not so obvious as they do have commutator terms of arbitrary large arity so that one can try to interpret NP-complete problems like in solvable but nonnilpotent case. However all known commutator terms have exponential size with respect to the number of variables whenever they are produced from finitely many operations. In fact the circuits representing those known commutator polynomials are also of exponential size. However, allowing infinitely many operations, we do have the possibility to express arbitrary large conjunctions by short polynomials. This phenomena is presented in the next Theorem.

▶ **Theorem 6.1.** *For two different prime numbers p, q there exists a 2-nilpotent algebra $\mathbf{A} = \mathbf{L} \otimes^F \mathbf{U}$ with $\mathbf{U}$ and $\mathbf{L}$ being polynomially equivalent to one dimensional vector spaces over the fields $GF(p)$ and $GF(q)$ respectively, such that $\textsc{Ceqv}_{\textsc{tm}}(\mathbf{A})$ is co-NP-complete and $\textsc{Csat}_{\textsc{tm}}(\mathbf{A})$ is NP-complete.*

Proof. We start with choosing $a \in U - \{0\}$ and $b \in L - \{0\}$ to define the following family of functions:

$$f_k((l_1^1, u_1^1), (l_2^1, u_2^1), (l_3^1, u_3^1), \ldots, (l_1^k, u_1^k), (l_2^k, u_2^k), (l_3^k, u_3^k)) = (\widehat{f_k}(u_1^1, u_2^1, u_3^1, \ldots, u_1^k, u_2^k, u_3^k), 0),$$

where

$$\widehat{f_k}(u_1^1, u_2^1, u_3^1, \ldots, u_1^k, u_2^k, u_3^k) = \begin{cases} b, & \text{if } a \in \{u_1^i, u_2^i, u_3^i\} \text{ for each } i, \\ 0, & \text{otherwise.} \end{cases}$$

Obviously the values of the f_k's can be computed by a single Turing machine in $O(k)$ time.

Now we define $\mathbf{A}$ to be $(L \times U; +_{\mathbf{A}}, \{f_k\}_{k=1}^{\infty})$, where $(L; +)$ and $(U; \oplus)$ are the groups of order q and p, respectively, and $(l_1, u_1) +_{\mathbf{A}} (l_2, u_2) = (l_1 + l_2, u_1 \oplus u_2)$.

To see that $\textsc{Csat}_{\textsc{tm}}(\mathbf{A})$ is NP-complete observe that a 3-CNF formula

$$(\ell_1^1 \vee \ell_2^1 \vee \ell_3^1) \wedge \cdots \wedge (\ell_1^k \vee \ell_2^k \vee \ell_3^k) \tag{9}$$

is satisfiable if and only if the following equation has a solution in $\mathbf{A}$

$$f_k(z_1^1, z_2^1, z_3^1, \ldots, z_1^k, z_2^k, z_3^k) = (b, 0),$$

where $z_i^j = x_i^j$ if ℓ_i^j is a positive literal and $z_i^j = (0, a) - x_i^j$ otherwise.

Similarly a formula (9) is not satisfiable iff the following equation holds in $\mathbf{A}$.

$$f_k(z_1^1, z_2^1, z_3^1, \ldots, z_1^k, z_2^k, z_3^k) = (0, 0).$$

This shows co-NP-completeness of $\textsc{Ceqv}_{\textsc{tm}}(\mathbf{A})$. ◀

Note here that the examples with co-NP-complete $\textsc{Ceqv}_{\textsc{tm}}$ and NP-complete $\textsc{Csat}_{\textsc{tm}}$ do exist even for $p = q$. Actually they are provided (but without detailed description of the input size) in [14]. However, if $p = q$ the resulting algebras must have infinitely many basic operations (as otherwise they would be supernilpotent), while for $p \neq q$ the algebras have finitely generated clone of operations but are presented with infinitely many basic operations only to (artificially) compress the size of the input.

Note also that in fact Theorem 6.1 actually establishes much more than just hardness of $\textsc{Csat}_{\textsc{tm}}$ and $\textsc{Ceqv}_{\textsc{tm}}$. Indeed, these examples show that (unless P = NP) for nilpotent but

not supernilpotent finite algebras one cannot expect polynomial time algorithms for CSAT or CEQV based on small search spaces S_n described in the Introduction. This is because the existence of such search spaces do not depend on the finiteness of the language.

Representing functions by Turing machines gives us a way to compress the input. However one can consider this as a drawback. That is because in this approach we can no longer treat basic operations occurring in the input as parameters in the way we do it for the finite set of operations. This probably denies the intuition behind what should an algorithm parameterized by an algebra mean. That is why $\mathrm{CEQV}_{\mathrm{T}}(\mathbf{A})$ and $\mathrm{CSAT}_{\mathrm{T}}(\mathbf{A})$, as described in the Introduction, appear to be more natural candidates for transferring these problems to the realm with possibly infinitely many basic operations. In contrast to Theorem 6.1 we have the following.

▶ **Theorem 6.2.** *Let* $\mathbf{A} = \mathbf{L} \otimes^F \mathbf{U}$ *with* $\mathbf{L}$ *and* $\mathbf{U}$ *being polynomially equivalent to one dimensional vector spaces over prime fields of different characteristics. Then* $\mathrm{CEQV}_{\mathrm{T}}(\mathbf{A})$ *and* $\mathrm{CSAT}_{\mathrm{T}}(\mathbf{A})$ *are in P.*

Proof. First note that from Corollary 3.3 every polynomial over $\mathbf{A}$ can be represented in some f-normal form. In view of Lemmas 3.2, 4.1 and 5.1 it suffices to show that obtaining f-normal forms of basic operations can be done in time polynomial in size of their tables.

To represent the basic operation g in the form of the right-hand side of (5) we need to compute all the λ_i's, $\mu_{\overline{\beta},u}$'s, α_i's and u_0 from the table of g. For $x \in L \times U$ we will use $\Pi_L(x)$ and $\Pi_U(x)$ to denote the first and second coordinate of x.

Now, to compute the α_i's and u_0 it suffices to solve the following system of $k+1$ linear equations

$$\bigoplus_{i=1}^{k} \alpha_i u_i \oplus u_0 = \Pi_U(g((0,u_1),\dots,(0,u_k)))\,,$$

where $(u_1,\dots,u_k)$ ranges over the set $\{(0,\dots,0),(1,0,\dots,0),(0,1,0,\dots,0)\dots,(0,\dots,0,1)\} \subseteq U^k$ and $\Pi_U(g((0,u_1),\dots,(0,u_k)))$ can be read from the table of g. Similarly, the λ_i's are the solutions of the following system of k linear equations

$$\sum_{i=1}^{k} \lambda_i l_i + \Pi_L(g((0,0),\dots,(0,0))) = \Pi_L(g((l_1,0),\dots,(l_k,0)))\,,$$

where again $(l_1,\dots,l_k)$ ranges over the set $\{(1,0,\dots,0),(0,1,0,\dots,0)\dots,(0,\dots,0,1)\} \subseteq L^k$. Finally, the $\mu_{\overline{\beta},u}$'s can be recovered from the following system of linear equations

$$\sum_{(\overline{\beta},u)\in \mathbf{F}_U^k \times U} \mu_{\overline{\beta},u} \cdot f\left(\bigoplus_{i=1}^{k} \beta_i u_i \oplus u\right) = \Pi_L(g((0,u_1),\dots,(0,u_k)))\,.$$

This time the system consists of $|U|^k$ equations (one for each $(u_1,\dots,u_k) \in U^k$) but this number is linear in the size of the table of g, as g is k-ary. ◀

References

1 Erhard Aichinger and Nebojša Mudrinski. Some applications of higher commutators in Mal'cev algebras. *Algebra Universalis*, 63(4):367–403, 2010.

2 A. A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 319–330, Oct. 2017. doi:10.1109/FOCS.2017.37.

3 Andrei Bulatov. On the number of finite Mal'tsev algebras. *Contributions to General Algebra*, 13:41–54, 2000.

4 Stanley Burris and H. P. Sankappanavar. *A course in universal algebra*, volume 78 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1981.

5 Tomás Feder, Florent Madelaine, and Iain A. Stewart. Dichotomies for classes of homomorphism problems involving unary functions. *Theoret. Comput. Sci.*, 314(1-2):1–43, 2004.

6 Ralph Freese and Ralph McKenzie. *Commutator theory for congruence modular varieties*, volume 125 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, 1987.

7 Mikael Goldmann and Alexander Russell. The complexity of solving equations over finite groups. *Inform. and Comput.*, 178(1):253–262, 2002.

8 Gábor Horváth. The complexity of the equivalence and equation solvability problems over nilpotent rings and groups. *Algebra Universalis*, 66(4):391–403, 2011.

9 Gábor Horváth. The complexity of the equivalence and equation solvability problems over meta-Abelian groups. *Journal of Algebra*, 433:208–230, 2015.

10 Gábor Horváth and Csaba Szabó. The Complexity of Checking Identities over Finite Groups. *Internat. J. Algebra Comput.*, 16(5):931–940, 2006. doi:10.1142/S0218196706003256.

11 Gábor Horváth and Csaba Szabó. Equivalence and equation solvability problems for the alternating group A_4. *Journal of Pure and Applied Algebra*, 216(10):2170–2176, 2012.

12 Paweł M. Idziak and Jacek Krzaczkowski. Satisfiability in multi-valued circuits. In *2018 Thirty-Third Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2018.

13 K.A. Kearnes. Congruence modular varieties with small free spectra. *Algebra Universalis*, 42(3):165–181, Oct 1999. doi:10.1007/s000120050132.

14 Michael Kompatscher. The equation solvability problem over nilpotent mal'cev algebras. *arXiv*, 2017. arXiv:1710.03083.

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

16 Jiyou Li and Daqing Wan. On the subset sum problem over finite fields. *Finite Fields and Their Applications*, 14(4):911–929, 2008. doi:10.1016/j.ffa.2008.05.003.

17 Ju. V. Matijasevič. The Diophantineness of enumerable sets. *Dokl. Akad. Nauk SSSR*, 191:279–282, 1970.

18 Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley Publishing Company, Reading, MA, 1994.

19 Bernhard Schwarz. The Complexity of Satisfiability Problems over Finite Lattices. In *2004 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, page 31–43, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

20 D. Zhuk. A proof of CSP dichotomy conjecture. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 331–342, Oct. 2017. doi:10.1109/FOCS.2017.38.

Lagrange's Theorem for Binary Squares

P. Madhusudan[1]
Department of Computer Science, Thomas M. Siebel Center for Computer Science,
201 North Goodwin Avenue, Urbana, IL 61801-2302, USA
madhu@illinois.edu
https://orcid.org/0000-0002-9782-721X

Dirk Nowotka
Department of Computer Science, Kiel University, D-24098 Kiel, Germany
dn@informatik.uni-kiel.de
https://orcid.org/0000-0002-5422-2229

Aayush Rajasekaran
School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada
arajasekaran@uwaterloo.ca

Jeffrey Shallit
School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada
shallit@uwaterloo.ca
https://orcid.org/0000-0003-1197-3820

Abstract

We show how to prove theorems in additive number theory using a decision procedure based on finite automata. Among other things, we obtain the following analogue of Lagrange's theorem: every natural number > 686 is the sum of at most 4 natural numbers whose canonical base-2 representation is a *binary square*, that is, a string of the form xx for some block of bits x. Here the number 4 is optimal. While we cannot embed this theorem itself in a decidable theory, we show that *stronger* lemmas that imply the theorem can be embedded in decidable theories, and show how automated methods can be used to search for these stronger lemmas.

2012 ACM Subject Classification Theory of computation → Models of computation, Theory of computation → Constructive mathematics, Mathematics of computing → Discrete mathematics

Keywords and phrases binary square, theorem-proving, finite automaton, decision procedure, decidable theory, additive number theory

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.18

1 Introduction

Additive number theory is the study of the additive properties of integers [12]. In particular, an *additive basis of order* h is a subset $S \subseteq \mathbb{N}$ such that every natural number is the sum of h members, not necessarily distinct, of S. The principal problem of additive number theory is to determine whether a given subset S is an additive basis of order h for some h, and if so, to determine the smallest value of h. There has been much research in the area, and deep techniques, such as the Hardy-Littlewood circle method, have been developed to solve these kinds of problems [20].

One of the earliest results in additive number theory is Lagrange's famous theorem [10] that every natural number is the sum of four squares [5, 11]. In the terminology of

[1] This material is based upon work supported by the National Science Foundation under Grant No. 1527395.

the previous paragraph, this means that $S = \{0^2, 1^2, 2^2, 3^2, \ldots\}$ forms an additive basis of order 4. The celebrated problem of Waring (1770) (see, e.g., [4, 19, 21]) is to determine the corresponding least order $g(k)$ for k'th powers. Since it is easy to see that numbers of the form $4^a(8k + 7)$ cannot be expressed as the sum of three squares, it follows that $g(2) = 4$. It is known that $g(3) = 9$ and $g(4) = 19$.

In a variation on this concept we say that $S \subseteq \mathbb{N}$ is an *asymptotic additive basis of order* h if every *sufficiently large* natural number is the sum of h members, not necessarily distinct, of S. The classical function $G(k)$ is defined to be the least asymptotic basis order for k'th powers. From above we have $G(2) = 4$. It is known that $G(14) = 16$, and $4 \leq G(3) \leq 7$. Despite much work, the exact value of $G(3)$ is currently unknown.

Recently there has been interest in doing additive number theory on sets of natural numbers whose base-k representations have certain properties. For example, Banks [1] proved that every natural number is the sum of at most 49 natural numbers whose decimal representation is a palindrome. This was improved by Cilleruelo, Luca, and Baxter [2] to 3 summands for every base $b \geq 5$. The remaining cases $b = 2, 3, 4$ were recently resolved [17].

In this paper we consider a variation on Lagrange's theorem. Instead of the ordinary notion of the square of an integer, we consider "squares" in the sense of formal language theory [8]. That is, we consider x, the canonical binary (base-2) representation of an integer N, and call N a *binary square* if $N = 0$, or if $x = yy$ for some nonempty string y that starts with a 1. Thus, for example, $N = 221$ is a binary square, since 221 in base 2 is $11011101 = (1101)(1101)$. The first few binary squares are

$$0, 3, 10, 15, 36, 45, 54, 63, 136, 153, 170, 187, 204, 221, 238, 255, \ldots;$$

they form sequence A020330 in the *On-Line Encyclopedia of Integer Sequences* (OEIS) [18]. Clearly a number $N > 0$ is a binary square if and only if it is of the form $a(2^n + 1)$ for $n \geq 1$ and $2^{n-1} \leq a < 2^n$. This is a very natural sequence to study, since the binary squares have density $\Theta(N^{1/2})$ in the natural numbers, just like the ordinary squares. (There exist sets of density $\Theta(N^{1/2})$ that do not form an asymptotic basis of finite order, so density considerations alone do *not* imply our result.)

In this paper we prove the following result.

▶ **Theorem 1.** *The binary squares form an asymptotic basis of order* 4. *More precisely, every natural number* $N > 686$ *is the sum of* 4 *binary squares. There are* 56 *exceptions, given below:*

$$1, 2, 4, 5, 7, 8, 11, 14, 17, 22, 27, 29, 32, 34, 37, 41, 44, 47, 53, 62, 95, 104, 107, 113, 116, 122, 125,$$
$$131, 134, 140, 143, 148, 155, 158, 160, 167, 407, 424, 441, 458, 475, 492, 509, 526, 552, 560,$$
$$569, 587, 599, 608, 613, 620, 638, 653, 671, 686.$$ ◀

The novelty in our approach is that we obtain this theorem in additive number theory *using very little number theory at all*. Instead, we use an approach based on formal language theory, reducing the proof of the theorem to a decidable language emptiness problem. Previously we obtained similar results for palindromes [15, 16, 17].

1.1 Search for appropriate lemmas and proving the theorem

The technique we use for encoding Theorem 1 as a problem dealing with automata is to ask, for all sufficiently large integers N, whether there exist four binary squares with representation $x_i x_i$, $1 \leq i \leq 4$, such that the sum of the numbers they represent is N. Since the language of

binary squares is not regular, we use an encoding where we represent only one copy of each x_i and reuse it to represent the number. However, it turns out that we cannot represent the desired theorem directly as an emptiness/universality problem of finite automata. The reason is that when representing only one copy of the x_i, we can do "school addition" (aligning them and adding the numbers, columnwise, with a carry) only if the words x_i are roughly of the same *length*. More precisely, we require the lengths of the squares employed to either be bounded by a constant, or differ from each other and from the number N only by a bounded length.

For fixed constants k_i, $1 \leq i \leq 4$, we observe that the set of all binary representations of N for which there exist four words x_i, $1 \leq i \leq 4$, of lengths $L - k_i$, such that the binary representation of N is of length $2L$ and the sum of the numbers represented by $x_i x_i$, $1 \leq i \leq 4$ is N, is a regular language. Thus we can prove, using known decision algorithms for automata, lemmas that assert that all numbers of a particular form can be represented by a sum of four binary squares, where the binary squares are of various lengths $L - k_1$, $L - k_2$, $L - k_3$, and $L - k_4$, for a finite set of tuples $\langle k_1, k_2, k_3, k_4 \rangle$ (see Lemma 5 for such a lemma).

Proving such a lemma for a particular set of combinations of lengths implies the theorem, of course, but the lemma itself is stronger. The truth of such stronger lemmas is decidable, while we don't have a way to directly decide the theorem itself!

Thus we need a *search* for an appropriate lemma for a particular combination of length differences that is valid. Given that checking these lemmas for any set of combinations is decidable, we can do the search for these lemmas automatically. We tried various combinations and succeeded in proving one lemma, namely Lemma 5, that implies our theorem.

The above technique can be generalized to some extent – evidently, we could also consider the analogous results for other powers such as cubes, and bases $b \geq 2$, but we do not do that in this paper.

1.2 Notation

We are concerned with the binary representation of numbers, so let us introduce some notation. If N is a natural number, then by $(N)_2$ we mean the string giving the canonical base-2 representation of N, having no leading zeroes. For example, $(43)_2 = 101011$. The canonical representation of 0 is ϵ, the empty string.

If $2^{n-1} \leq N < 2^n$ for $n \geq 1$, we say that N is an *n-bit integer* in base 2. Note that the first bit of the binary representation of an n-bit integer is always nonzero. The *length* of an integer N satisfying $2^{n-1} \leq N < 2^n$ is defined to be n; alternatively, the length of N is $1 + \lfloor \log_2 N \rfloor$. For $n \geq 1$ we define $C_n = \{a \cdot (2^n + 1) \ : \ 2^{n-1} \leq a < 2^n\}$, the set of all $2n$-bit binary squares.

2 A classical approach

In this section we describe how one can apply classical number-theoretic and combinatorial tools to this problem to obtain some results weaker than Theorem 1. The idea is to show that the numbers that are the sum of two binary squares form a set of positive lower asymptotic density. (In contrast, our approach via automata, which we discuss in later sections, provides more precise results.)

For sets $S, T \subseteq \mathbb{N}$ we define the sumset $S + T = \{s + t \ : \ s \in S, \ t \in T\}$. The cardinality of a finite set S is denoted by $|S|$. Given a set $S \subseteq \mathbb{N}$, the *lower asymptotic density* of S is defined to be

$$d(S) = \liminf_{n \to \infty} \frac{|\{x \in S \ : \ 1 \leq x \leq n\}|}{n}.$$

We first prove

▶ **Lemma 2.** *For $n \geq 1$ we have $|C_n + C_{n+1}| = 2^{2n-1}$.*

Proof. Since $|C_n| = 2^{n-1}$ and $|C_{n+1}| = 2^n$, this lemma is equivalent to the claim that each member of the sumset $C_n + C_{n+1}$ has a *unique* representation as the sum of one element of C_n and one element of C_{n+1}.

We argue by contradiction. Suppose the representation is not unique, and there exist integers a, a' with $2^{n-1} \leq a, a' < 2^n - 1$ and integers b, b' with $2^n \leq b, b' < 2^{n+1}$ such that $(a, a') \neq (b, b')$ but

$$a \cdot (2^n + 1) + b \cdot (2^{n+1} + 1) = a' \cdot (2^n + 1) + b' \cdot (2^{n+1} + 1). \tag{1}$$

Computing Eq. (1) modulo 2^n+1, we see that $-b \equiv -b' \pmod{2^n+1}$. Since $2^n \leq b, b' < 2^{n+1}$ we see the congruence in fact implies that $b = b'$. But then $a = a'$, a contradiction. ◀

▶ **Theorem 3.** *The numbers that are the sum of two binary squares form a set of lower asymptotic density $\geq 1/40$.*

Proof. Let S_2 be the set of numbers that are the sum of two binary squares. Clearly $C_n + C_{n+1} \subseteq S_2$.

There are 2^{2n-1} elements in the sumset $C_n + C_{n+1}$, whose largest element is $(2^n - 1)2^n + (2^{n+1} - 1)2^{n+1} = 5 \cdot 2^{2n} - 3 \cdot 2^n$. Given an integer $m \geq 14$, choose $n \geq 1$ such that $5 \cdot 2^{2n} - 3 \cdot 2^n \leq m < 5 \cdot 2^{2n+2} - 3 \cdot 2^{n+1}$. Then

$$\frac{|\{x \in S_2 \ : \ 1 \leq x \leq m\}|}{m} \geq \frac{2^{2n-1}}{5 \cdot 2^{2n+2}} = \frac{1}{40}.$$

◀

▶ **Corollary 4.** *The binary squares form an asymptotic basis of finite order.*

Proof. This is a direct consequence of a result of Nathanson [13, Theorem 11.6, p. 366], which says that if a subset S of $\mathbb{N}$ has $0 \in S$, $\gcd(S) = 1$, and has positive lower asymptotic density, then it is an asymptotic basis of finite order. It is now easy to check that the hypotheses are fulfilled for $S = S_2$. ◀

▶ Remark. It would be interesting to determine the exact lower asymptotic density of the set S_2. Numerical computation suggests that perhaps $d(S_2) \doteq .14$.

3 The automaton approach: the main lemma

Now we turn to a completely different approach to the theorem for binary squares, as sketched in Section 1, using automata theory. This allows us to obtain the upper bound 4 for the number of binary squares, a stronger result than obtained using the classical approach.

Our main lemma is

▶ **Lemma 5.**

(a) *Every length-n integer, n odd, $n \geq 13$, is the sum of binary squares as follows: either*

- *one of length $n-1$ and one of length $n-3$, or*
- *two of length $n-1$ and one of length $n-3$, or*
- *one of length $n-1$ and two of length $n-3$, or*
- *one each of lengths $n-1$, $n-3$, and $n-5$, or*
- *two of length $n-1$ and two of length $n-3$, or*
- *two of length $n-1$, one of length $n-3$, and one of length $n-5$.*

(b) *Every length-n integer, n even, $n \geq 18$, is the sum of binary squares as follows: either*
- *two of length $n-2$ and two of length $n-4$, or*
- *three of length $n-2$ and one of length $n-4$, or*
- *one each of lengths n, $n-4$, and $n-6$, or*
- *two of length $n-2$, one of length $n-4$, and one of length $n-6$.*

Lemma 5 almost immediately proves Theorem 1:

Proof. If $N < 2^{17} = 131072$, the result can be proved by a completely straightforward computation using dynamic programming: to form the sumset $S \oplus T$, given finite sets of natural numbers S and T, we use a bit vector corresponding to the elements of S, and then take its XOR shifted by each element of T. When we do this, we find that there are
- 256 binary squares $< 2^{17}$;
- 19542 numbers $< 2^{17}$ that are the sum of two binary squares;
- 95422 numbers $< 2^{17}$ that are the sum of three binary squares;
- 131016 numbers $< 2^{17}$ that are the sum of four binary squares.

Otherwise $N \geq 2^{17}$, so $(N)_2$ is a binary string of length $n \geq 18$. If n is odd, the result follows from Lemma 5 (a). If n is even, the result follows from Lemma 5 (b). ◀

It now remains to prove Lemma 5. We do this in the next section.

4 Proof of Lemma 5

In this section we prove Lemma 5 in detail.

Proof. The basic idea is to use nondeterministic finite automata (NFAs). These are finite-state machines where each input corresponds to multiple computational paths; an input is accepted iff *some* computational path leads to a final state. We assume the reader is familiar with the basics of this theory; if not, please consult, e.g., [8]. For us, an NFA is a quintuple $(Q, \Sigma, \delta, q_0, F)$, where Q is the set of states, Σ is the input alphabet, δ is the transition function, q_0 is the initial state, and F is the set of final states.

We construct an NFA that, on input an integer N written in binary, "guesses" a representation as a sum of binary squares, and then verifies that the sum is indeed N. Everything is done using a reversed representation, with least significant digits processed first. There are some complications, however.

First, with an NFA we cannot verify that a guessed string is indeed a binary square, as the language $\{xx \ : \ x \in 1\{0,1\}^*\}$ is not a regular language. So instead we only guess the "first half" of a binary square. Now, however, we are forced to choose a slightly unusual representation for N, in order to be able to compare the sum of our guessed powers with the input N. If N were represented in its ordinary base-2 representation, this would be impossible with an NFA, since once we process the guessed "first half" and compare it to the input, we would no longer have the "second half" (identical to the first) to compare to the rest of the input.

To get around this problem, we represent integers N in a kind of "folded representation" over the input alphabet $\Sigma_2 \cup (\Sigma_2 \times \Sigma_2)$, where $\Sigma_k = \{0, 1, \ldots, k-1\}$. The idea is to present our NFA with two bits of the input string at once, so that we can add both halves of our guessed powers at the same time, verifying that we are producing N as we go. Note that we use slightly different representations for the two parts of Lemma 5. The precise representations are detailed in their respective subsections.

We can now prove Lemma 5 by phrasing it as a language inclusion problem. For each of the two parts of the lemma, we can build an NFA A that only accepts such folded strings if

they repesent numbers that are the sum of any of the combination of squares as described in the lemma. We also create an NFA, B, that accepts all valid folded representations that are sufficiently long. We then check the assertion that the language recognized by B is a subset of that recognized by A.

4.1 Odd-length inputs

Again, to flag certain positions of the input tape, we use an extended alphabet. Define

$$\Gamma = \{1_f\} \cup \bigcup_{\alpha\in\{a,b,c,d,e\}} \{[0,0]_\alpha, [0,1]_\alpha, [1,0]_\alpha, [1,1]_\alpha\}.$$

Let N be an integer, and let $n = 2i+1$ be the length of its binary representation. We write $(N)_2 = a_{2i}a_{2i-1}\cdots a_1a_0$ and fold this to produce the input string

$$[a_i, a_0]_a[a_{i+1}, a_1]_a \cdots [a_{2i-5}, a_{i-5}]_a[a_{2i-4}, a_{i-4}]_b[a_{2i-3}, a_{i-3}]_c[a_{2i-2}, a_{i-2}]_d[a_{2i-1}, a_{i-1}]_e a_{2i_f}.$$

Let A_{odd} be the NFA that recognizes those odd-length integers, represented in this folded format, that are the sum of binary squares meeting any of the 6 conditions listed in Lemma 5 (a). We construct A_{odd} as the union of several automata $A(t_{n-1}, t_{n-3}, m_a)$ and $B(t_{n-1}, t_{n-3}, t_{n-5}, m_b)$. The parameters t_p represent the number of summands of length p we are guessing. The parameters m_a and m_b are the carries that we are guessing will be produced by the first half of the summed binary squares. A-type machines try summands of lengths $n-1$ and $n-3$ only, while B-type machines include at least one $(n-5)$-length summand. We note that for the purpose of summing, guessing t binary squares is equivalent to guessing a *single* square over the larger alphabet Σ_{t+1}.

We now consider the construction of a single automaton

$$A(t_{n-1}, t_{n-3}, m) = (Q \cup \{q_{\text{acc}}, q_0, s_1\}, \Gamma, \delta, q_0, \{q_{\text{acc}}\}).$$

The elements of Q have 4 non-negative parameters and are of the form $q(x_1, x_2, c_1, c_2)$. Because the t_{n-3} summand is not aligned with the input, we use our states to "remember" our guesses. When we make a guess at the higher end of the t_{n-3} summand, it must be used as the guess for its lower end on the *next step*. We remember this guess by storing it as the x_2 parameter. The parameter $x_1 \le t_{n-3}$ is the last digit of the guessed summand of length $n-3$. We use c_1 to track the higher carry, and c_2 to track the lower carry. We must have $c_1, c_2 < t_{n-1} + t_{n-3}$.

We now discuss the transition function, δ of our NFA. In this section, we say that the sum of natural numbers, μ_1 and μ_2, "produces" an output bit of $\theta \in \Sigma_2$ with a "carry" of γ if $\mu_1 + \mu_2 \equiv \theta \pmod 2$ and $\gamma = \left\lfloor \frac{\mu_1+\mu_2}{2} \right\rfloor$.

We allow a transition from q_0 to $q(x_1, x_2, c_1, c_2)$ on the letter $[j,k]_a$ iff there exists $0 \le r \le t_{n-1}$ such that $x_2 + r + m$ produces an output of j with a carry of c_1 and $x_1 + r$ produces an output of k with a carry of c_2.

We allow a transition from $q(x_1, x_2, c_1, c_2)$ to $q(x_1', x_2', c_1', c_2')$ on the letters $[j,k]_a$ and $[j,k]_b$ iff there exists $0 \le r \le t_{n-1}$ such that $x_2' + r + c_1$ produces an output of j with a carry of c_1' and $x_2 + r + c_2$ produces an output of k with a carry of c_2'. Elements of Q have identical transitions on inputs with subscripts a and b. The reason we have the letters with subscript b is for B-machines, which guess a summand of length $n-5$.

There is only one letter of the input with the subscript c, and it corresponds to the last higher guess of the summand of length $n-3$. We allow a transition from $q(x_1, x_2, c_1, c_2)$ to $q(x_1', t_{n-3}, c_1', c_2')$ on the letter $[j,k]_c$ iff there exists $0 \le r \le t_{n-1}$ such that $t_{n-3} + r + c_1$

produces an output of j with a carry of c_1' and $x_2 + r + c_2$ produces an output of k with a carry of c_2'.

There is only one letter of the input with the subscript d, and it corresponds to the second-last lower guess of the summand of length $n-3$. We allow a transition from $q(x_1, t_{n-3}, c_1, c_2)$ to $q(x_1', 0, c_1', c_2')$ on the letter $[j,k]_d$ iff there exists $0 \leq r \leq t_{n-1}$ such that $r + c_1$ produces an output of j with a carry of c_1' and $t_{n-3} + r + c_2$ produces an output of k with a carry of c_2'.

There is only one letter of the input with the subscript e, and it corresponds to the last lower guess of the summand of length $n-3$. We allow a transition from $q(x_1, 0, c_1, c_2)$ to s_1 on the letter $[j,k]_e$ iff $t_{n-1} + c_1$ produces an output of j with a carry of 1 and $x_1 + t_{n-1} + c_2$ produces an output of k with a carry of m.

Finally, we add a transition from s_1 to q_{acc} on the letter 1_f.

We now consider the construction of a single automaton

$$B(t_{n-1}, t_{n-3}, t_{n-5}, m) = (P \cup Q \cup \{q_{\text{acc}}, q_0, s_1\}, \Gamma, \delta, q_0, \{q_{\text{acc}}\}).$$

The elements of P have 6 non-negative parameters and are of the form $q(x_1, x_2, y_1, y_3, c_1, c_2)$. The parameter $x_1 \leq t_{n-3}$ is the last digit of the guessed summand of length $n-3$ and $x_2 \leq t_{n-3}$ is the previous higher guess of the length-$n-3$ summand. The parameter $y_1 \leq t_{n-5}$ is the last digit of the guessed summand of length $n-5$ and $y_3 \leq t_{n-5}$ is the previous higher guess of the length-$n-5$ summand. We use c_1 to track the higher carry, and c_2 to track the lower carry. We must have $c_1, c_2 < t_{n-1} + t_{n-3} + t_{n-5}$. The elements of Q have 8 non-negative parameters and are of the form

$$q(x_1, x_2, y_1, y_2, y_3, y_4, c_1, c_2).$$

The parameter $x_1 \leq t_{n-3}$ is the last digit of the guessed summand of length $n-3$ and $x_2 \leq t_{n-3}$ is the previous higher guess of the length-$n-3$ summand. The parameters $y_1, y_2 \leq t_{n-5}$ are the last digit and the second-last digit of the guessed summand of length $n-5$ respectively. The parameter $y_3, y_4 \leq t_{n-5}$ are the two most recent higher guess of the length-$n-5$ summand, with y_4 being the most recent one. We use c_1 to track the higher carry, and c_2 to track the lower carry. We must have $c_1, c_2 < t_{n-1} + t_{n-3} + t_{n-5}$.

We now discuss the transition function, δ of our NFA. We allow a transition from q_0 to $p(x_1, x_2, y_1, y_3, c_1, c_2)$ on the letter $[j,k]_a$ iff there exists $0 \leq r \leq t_{n-1}$ such that $x_2 + y_3 + r + m$ produces an output of j with a carry of c_1 and $x_1 + y_1 + r$ produces an output of k with a carry of c_2.

We use a transition from $p(x_1, x_2, y_1, y_3, c_1, c_2)$ to $q(x_1, x_2', y_1, y_2', y_3, y_4', c_1', c_2')$ on the letter $[j,k]_a$ iff there exists $0 \leq r \leq t_{n-1}$ such that $x_2' + y_4' + r + c_1$ produces an output of j with a carry of c_1 and $x_2 + y_2' + r + c_2$ produces an output of k with a carry of c_2.

We use a transition from $q(x_1, x_2, y_1, y_2, y_3, y_4, c_1, c_2)$ to $q(x_1, x_2', y_1, y_2, y_4, y_4', c_1', c_2')$ on the letter $[j,k]_a$ iff there exists $0 \leq r \leq t_{n-1}$ such that $x_2' + y_4' + r + c_1$ produces an output of j with a carry of c_1 and $x_2 + y_3 + r + c_2$ produces an output of k with a carry of c_2.

We use a transition from $q(x_1, x_2, y_1, y_2, y_3, t_{n-5}, c_1, c_2)$ to $q(x_1, x_2', y_1, y_2, t_{n-5}, t_{n-5}, c_1', c_2')$ on the letter $[j,k]_b$ iff there exists $0 \leq r \leq t_{n-1}$ such that $x_2' + r + c_1$ produces an output of j with a carry of c_1 and $x_2 + y_3 + r + c_2$ produces an output of k with a carry of c_2.

We use a transition from $q(x_1, x_2, y_1, y_2, t_{n-5}, t_{n-5}, c_1, c_2)$ to $q(x_1, t_{n-3}, y_1, y_2, t_{n-5}, t_{n-5}, c_1', c_2')$ on the letter $[j,k]_c$ iff there exists $0 \leq r \leq t_{n-1}$ such that $t_{n-3} + r + c_1$ produces an output of j with a carry of c_1 and $x_2 + y_3 + r + c_2$ produces an output of k with a carry of c_2.

We use a transition from $q(x_1, t_{n-3}, y_1, y_2, t_{n-5}, t_{n-5}, c_1, c_2)$ to $q(x_1, t_{n-3}, y_1, y_2, t_{n-5}, t_{n-5}, c_1', c_2')$ on the letter $[j,k]_d$ iff there exists $0 \leq r \leq t_{n-1}$ such that $r + c_1$ produces an

output of j with a carry of c_1 and $t_{n-3} + y_1 + r + c_2$ produces an output of k with a carry of c_2.

We use a transition from $q(x_1, t_{n-3}, y_1, y_2, t_{n-5}, t_{n-5}, c_1, c_2)$ to s_1 on the letter $[j,k]_e$ iff $t_{n-1} + c_1$ produces an output of j with a carry of 1 and $x_1 + y_2 + t_{n-1} + c_2$ produces an output of k with a carry of m.

Finally, we add a transition from s_1 to q_{acc} on the letter 1_f.

We now turn to verification of the inclusion assertion. We used the Automata Library toolchain of the `ULTIMATE` program analysis framework [7, 6] to establish our results. The `ULTIMATE` code proving our result can be found in the file `OddSquareConjecture.ats` at `https://cs.uwaterloo.ca/~shallit/papers.html`. Since the constructed machines get very large, we wrote a C++ program generating these machines, which can be found in the file `OddSquares.cpp` at the same location.

The final machine, A_{odd}, has 2258 states. The syntax checker, B, has 8 states. We then asserted that the language recognized by B is a subset of that recognized by A. ULTIMATE verified this assertion in under a minute. Since this test succeeded, the proof of Lemma 5 (a) is complete.

4.2 Even-length inputs

In order to flag certain positions of the input tape, we use an extended alphabet. Define

$$\Gamma = \left(\bigcup_{\alpha \in \{a,b,c,d,e\}} \{[0,0]_\alpha, [0,1]_\alpha, [1,0]_\alpha, [1,1]_\alpha\} \right) \cup \left(\bigcup_{\beta \in \{f,g,h,i\}} \{0_\beta, 1_\beta\} \right).$$

Let N be an integer, and let $n = 2i + 4$ be the length of its binary representation. We write $(N)_2 = a_{2i+3}a_{2i+2} \cdots a_1 a_0$ and fold this to produce the input string

$$[a_i, a_0]_a[a_{i+1}, a_1]_b[a_{i+2}, a_2]_c[a_{i+3}, a_3]_c \cdots \\ \cdots [a_{2i-3}, a_{i-3}]_c[a_{2i-2}, a_{i-2}]_d[a_{2i-1}, a_{i-1}]_e a_{2i_f} a_{2i+1_g} a_{2i+2_h} a_{2i+3_i}.$$

Let A_{even} be the NFA that recognizes the even-length integers, represented in this folded format, iff the integer is the sum of binary squares meeting any of the 4 conditions listed in Lemma 5 (b). We construct A_{even} as the union of several automata $A(t_n, t_{n-2}, t_{n-4}, t_{n-6}, m)$. The parameters t_p represent the number of summands of length p we are guessing. The parameter m is the carry that we are guessing will be produced by the first half of the summed binary squares. Again, guessing t binary squares is equivalent to guessing a *single* square over the larger alphabet Σ_{t+1}.

We now consider the construction of a single automaton

$$A(t_n, t_{n-2}, t_{n-4}, t_{n-6}, m) = (Q \cup \{q_{\text{acc}}\}, \Gamma, \delta, q_0, \{q_{\text{acc}}\}).$$

The elements of Q have 8 non-negative parameters and are of the form $q(x_1, x_2, x_3, y_1, z_1, z_2, c_1, c_2)$. The parameter x_1 is the second digit of the guessed summand of length n. The parameters x_2 and x_3 represent the previous 2 lower guesses of the length-n summand; these must be the next 2 higher guesses of this summand. The parameter y_1 represents the previous lower guess of the length-$(n-2)$ summand. We set z_1 as the last digit of the guessed summand of length $n-6$, while z_2 is the previous higher guess of this summand. Finally, c_1 tracks the lower carry, while c_2 tracks the higher carry. For any p, we must have $x_p \le t_n$, $y_p \le t_{n-2}$, $z_p \le t_{n-6}$, and $c_p < t_n + t_{n-2} + t_{n-4} + t_{n-6}$. The initial state, q_0, is $q(0,0,0,0,0,0,0,0)$.

We now discuss the transition function, δ of our NFA. Note that in our representation of even-length integers, the first letter of the input must have the subscript a, and it is the only letter to do so. We only allow the initial state to have outgoing transitions on such letters.

We allow a transition from q_0 to $q(x_1, 0, x_3, y_1, z_1, z_2, c_1, c_2)$ on the letter $[j, k]_a$ iff there exists $0 \leq r \leq t_{n-4}$ such that $x_1 + t_{n-2} + r + z_2 + m$ produces an output of j with a carry of c_2 and $x_3 + y_1 + r + z_1$ produces an output of k with a carry of c_1.

The second letter of the input must have the subscript b, and it is the only letter to do so. We allow a transition from $q(x_1, 0, x_3, y_1, z_1, z_2, c_1, c_2)$ to $q(x_1, x_3, x'_3, y'_1, z_1, z'_2, c'_1, c'_2)$ on the letter $[j, k]_b$ iff there exists $0 \leq r \leq t_{n-4}$ such that $t_n + y_1 + r + z'_2 + c_2$ produces an output of j with a carry of c'_2 and $x'_3 + y'_1 + r + z_2 + c_1$ produces an output of k with a carry of c'_1.

We allow a transition from $q(x_1, x_2, x_3, y_1, z_1, z_2, c_1, c_2)$ to $q(x_1, x_3, x'_3, y'_1, z_1, z'_2, c'_1, c'_2)$ on the letter $[j, k]_c$ iff there exists $0 \leq r \leq t_{n-4}$ such that $x_2 + y_1 + r + z'_2 + c_2$ produces an output of j with a carry of c'_2 and $x'_3 + y'_1 + r + z_2 + c_1$ produces an output of k with a carry of c'_1.

The letter of the input with the subscript d corresponds to the last guess of the lower half of the summand of length $n - 6$, and it is the only letter to do so. We allow a transition from $q(x_1, x_2, x_3, y_1, z_1, t_{n-6}, c_1, c_2)$ to $q(x_1, x_3, x'_3, y'_1, z_1, 0, c'_1, c'_2)$ on the letter $[j, k]_d$ iff there exists $0 \leq r \leq t_{n-4}$ such that $x_2 + y_1 + r + c_2$ produces an output of j with a carry of c'_2 and $x'_3 + y'_1 + r + t_{n-6} + c_1$ produces an output of k with a carry of c'_1.

The letter of the input with the subscript e corresponds to the last guess of both halves of the summand of length $n - 4$, and it is the only letter to do so. We allow a transition from $q(x_1, x_2, x_3, y_1, z_1, 0, c_1, c_2)$ to $q(x_1, x_3, x'_3, y'_1, 0, 0, 0, c'_2)$ on the letter $[j, k]_e$ iff $x_2 + y_1 + t_{n-4} + c_2$ produces an output of j with a carry of c'_2 and $x'_3 + y'_1 + t_{n-4} + z_1 + c_1$ produces an output of k with a carry of m.

We allow a transition from $q(x_1, x_2, x_3, y_1, 0, 0, 0, c_2)$ to $q(x_1, x_3, 0, 0, 0, 0, 0, c'_2)$ on the letter j_f iff $x_2 + y_1 + c_2$ produces an output of j with a carry of c'_2.

We allow a transition from $q(x_1, x_2, 0, 0, 0, 0, c_2)$ to $q(x_1, 0, 0, 0, 0, 0, 0, c'_2)$ on the letter j_g iff $x_2 + t_{n-2} + c_2$ produces an output of j with a carry of c'_2.

We allow a transition from $q(x_1, 0, 0, 0, 0, 0, c_2)$ to $q(0, 0, 0, 0, 0, 0, 0, c'_2)$ on the letter j_h iff $x_1 + c_2$ produces an output of j with a carry of c'_2.

We allow a transition from $q(0, 0, 0, 0, 0, 0, 0, c_2)$ to q_{acc} on the letter 1_i iff $t_n + c_2$ produces an output of 1 with a carry of 0.

The final machine, A_{even} is constructed as the union of 15 automata:

- $A(0, 2, 2, 0, m)$, varying m from 0 to 3
- $A(0, 3, 1, 0, m)$, varying m from 0 to 3
- $A(1, 0, 1, 1, m)$, varying m from 0 to 2
- $A(0, 2, 1, 1, m)$, varying m from 0 to 3

We now turn to verification of the inclusion assertion. The `ULTIMATE` code proving our result can be found in the file `EvenSquareConjecture.ats` at `https://cs.uwaterloo.ca/~shallit/papers.html`. Since the constructed machines get very large, we wrote a C++ program generating these machines, which can be found in the file `EvenSquares.cpp` at the same location.

The final machine, A_{even}, has 1343 states. The syntax checker, B, has 12 states. We then asserted that the language recognized by B is a subset of that recognized by A. ULTIMATE verified this assertion in under a minute. Since this test succeeded, the proof of Lemma 5 (b) is complete. ◀

▶ **Corollary 6.** *Given an integer $N > 686$, we can find an expression for N as the sum of four binary squares in time linear in* $\log N$.

Proof. For $N < 131072$, we do this with a simple brute-force search via dynamic programming, as explained previously. Otherwise we construct the appropriate automaton A (depending on whether the binary representation of N has either even or odd length). Now carry out the usual direct product construction for intersection of languages on A and B, where B is the automaton accepting the folded binary representation of N. The resulting automaton has at most $c \log N$ states and transitions. Now use the usual depth-first search of the transition graph to find a path from the initial state to a final state. The labels of this path gives the desired representation. ◀

4.3 Ensuring correctness

As in every machine-based proof, we want some assurance that our calculations were correct.

We tested our machine by calculating those integers of length 8 that can be expressed as the sum of up to 3 binary squares of length 4, and up to 4 binary squares of length 6. We then used the `ULTIMATE` framework to test that those length-8 integers are accepted by our machine, but all others are rejected. The code running this test can be found as `Minus2Minus4SquareConjecture - Test 1` at `https://cs.uwaterloo.ca/ shallit/papers.html`.

We also tested the machine by calculating those integers of length 10 that can be expressed as the sum of up to to 2 binary squares of length 6, and up to 4 binary squares of length 8. We then built the analogous machine and confirmed that these length-10 integers are accepted, but all others are rejected. We then repeated this test for those integers of length 10 that can be expressed as the sum of up to to 3 binary squares of length 6, and up to 3 binary squares of length 8. The code running these tests can be found as `Minus2Minus4SquareConjecture - Test 2` and `Minus2Minus4SquareConjecture - Test 3` at `https://cs.uwaterloo.ca/~shallit/papers.html`.

5 Optimality

In this section we show that the "4" in Theorem 1 is optimal.

▶ **Theorem 7.** *For $n \geq 1$, n odd, $n \neq 9$, the number 2^n is not the sum of three or fewer (positive) binary squares.*

Proof. Let $m \geq 0$ and $n = 2m + 1$ be odd. The cases $m = 0, 1, 2, 3$ are easy to verify by hand, so assume $m \geq 4$.

In what follows we distinguish between "mod" used in the ordinary notion of congruence (where $x \equiv a \pmod b$ means that b divides $x - a$), and the use of "mod" as a function, where $x = a \bmod b$ means both that $x \equiv a \pmod b$ and that $0 \leq a < b$.

Clearly $N := 2^n$ is not a binary square.

Suppose N is the sum of two positive binary squares. The largest binary square $< N$ is clearly $2^{2m} - 1$. Hence the sum of two binary squares is either larger than N, or no larger than $2(2^{2m} - 1) = 2^{2m+1} - 2 < N$, a contradiction.

The remaining case is that 2^{2m+1} is the sum of three binary squares, say $N = A + B + C$ with

$$A = a(2^e + 1) \geq B = b(2^f + 1) \geq C = c(2^g + 1)$$

with $e \geq f \geq g$ and $2^{e-1} \leq a < 2^e$, $2^{f-1} \leq b < 2^f$, and $2^{g-1} \leq c < 2^g$. Clearly $1 \leq e, f, g \leq m$.

We first observe that $e = m$. For otherwise, $e \leq m-1$ and the inequality $e \geq f \geq g$ implies

$$N = A + B + C \leq 3(2^{m-1} - 1)2^{m-1} < 3 \cdot 2^{2m-2} < N,$$

a contradiction.

Similarly, we observe that $f = m$. For otherwise

$$N = A + B + C \leq (2^m - 1)2^m + 2(2^{m-1} - 1)2^{m-1} < 3 \cdot 2^{2m-1} < N,$$

a contradiction.

Thus, setting $d = a + b$, we see that $N = d(2^m + 1) + c(2^g + 1)$ where $2^m \leq d \leq 2^{m+1} - 2$. Suppose $d = 2^{m+1} - 2$. Then $N = d(2^m + 1) + c(2^g + 1)$ implies that $C = c(2^g + 1) = 2$. But $C = 2$ is not a binary square. So in fact $2^m \leq d \leq 2^{m+1} - 3$.

Next we argue that $g > m/2$. For otherwise $g \leq m/2$ and we have

$$N = d(2^m+1)+c(2^g+1) \leq (2^{m+1}-3)(2^m+1)+(2^{m/2}-1)(2^{m/2}+1) = 2^{2m+1}-4 = N-4,$$

a contradiction.

Next we argue that $g < m$. For otherwise $g = m$ and then $N = 2^{2m+1} = A + B + C = (a + b + c)(2^m + 1)$. But then 2^{2m+1} is divisible by the odd number $2^m + 1$, a contradiction.

Now consider the equation $N = d(2^m + 1) + c(2^g + 1)$ and take it modulo $2^m + 1$. We have $2^{2m+1} - 2 = 2(2^m - 1)(2^m + 1) \equiv 0 \pmod{2^m + 1}$, and so $N = 2^{2m+1} \equiv 2 \pmod{2^m + 1}$.

Thus we get

$$c(2^g + 1) \equiv 2 \pmod{2^m + 1}. \tag{2}$$

It suffices to show that the congruence (2) has no solutions in the possible range for c, except when $m = 4$ and $g = 3$. In order to see this, we need a technical lemma.

▶ **Lemma 8.** *Suppose $m, g \geq 1$ are integers with $m/2 < g < m$. Suppose c is an integer with $2^{g-1} \leq c < 2^g$. Using Euclidean division, find the unique expression of c as $t \cdot 2^{m-g} + u$ for $0 \leq u < 2^{m-g}$. Then*

$$c(2^g + 1) \bmod (2^m + 1) = t(2^{m-g} - 1) + u(2^g + 1).$$

Proof. We have

$$\begin{aligned} c(2^g + 1) &= (t \cdot 2^{m-g} + u)(2^g + 1) \\ &= t \cdot 2^m + t \cdot 2^{m-g} + u(2^g + 1) \\ &= t(2^m + 1) + t(2^{m-g} - 1) + u(2^g + 1) \\ &\equiv t(2^{m-g} - 1) + u(2^g + 1) \pmod{2^m + 1}. \end{aligned}$$

This last congruence alone does not prove what we want; we also have to show that

$$0 \leq t(2^{m-g} - 1) + u(2^g + 1) < 2^m + 1$$

so that the residues don't "wrap around" when computed modulo $2^m + 1$. However, $t = \lfloor c/2^{m-g} \rfloor = 2^{2g-m} - 1$, and so

$$\begin{aligned} t(2^{m-g} - 1) + u(2^g + 1) &\leq (2^{2g-m} - 1)(2^{m-g} - 1) + (2^{m-g} - 1)(2^g + 1) \\ &= 2^m - 2^{2g-m} < 2^m + 1, \end{aligned}$$

as desired. ◀

Now from the Lemma we see that the expression $c(2^g + 1) \bmod (2^m + 1)$ achieves its smallest value when $c = 2^{g-1}$ (for then $t = 2^{2g-m-1}$ and $u = 0$), and this smallest value is $2^{2g-m-1}(2^{m-g} - 1) > 2$, except when $m = 4$, $g = 3$. ◀

▶ Remark. When $m = 4$ and $g = 3$, letting $c = 28$ and $d = 4$ we get the solution $512 = 2^9 = 28 \cdot (2^4 + 1) + 4 \cdot (2^3 + 1)$. This corresponds to two distinct expressions of 2^9 as the sum of three binary squares: $512 = 255 + 221 + 36$ and $512 = 238 + 238 + 36$.

6 Other results

Our technique can be used to obtain other results in additive number theory. For example, recently Crocker [3] and Platt & Trudgian [14] studied the integers representable as the sum of two ordinary squares and two powers of 2. The analogue of this theorem is the following:

▶ **Lemma 9.**

(a) *Every length-n integer, n odd, $n \geq 7$, is the sum of at most two powers of 2 and either:*
- *at most two binary squares of length $n - 1$, or*
- *at most one binary square of length $n - 1$ and one of length $n - 3$.*

(b) *Every length-n integer, n even, $n \geq 10$, is the sum of at most two powers of 2 and either:*
- *at most one binary square of length n and one of length $n - 4$, or*
- *at most one binary square of length $n - 2$ and one of length $n - 4$.*

Proof. We use a similar proof strategy as before. The `ULTIMATE` code proving our result can be found in the files `OddSquarePowerConjecture.ats` and `EvenSquarePowerConjecture.ats` at `https://cs.uwaterloo.ca/~shallit/papers.html`; there one can also find the generators can be found as `OddSquarePower.cpp` and `EvenSquarePower.cpp`.

The final machines for the odd-length and even-length cases have 806 and 2175 states respectively. The language inclusion assertions all hold. This concludes the proof. ◀

We thus have the following theorem:

▶ **Theorem 10.** *Every natural number N is the sum of at most two binary squares and at most two powers of 2.*

Proof. For $N < 512$, the result can be easily verified. Otherwise, we use Lemma 9 (a) if N is an odd-length binary number and Lemma 9 (b) if it is even. ◀

We also consider the notion of *generalized binary squares*. A number N is called a generalized binary square if one can concatenate 0 or more leading zeroes to its binary representation to produce a binary square. As an example, 9 is a generalized binary square, since 9 in base 2 is 1001, which can be written as $001001 = (001)(001)$. The first few generalized binary squares are

$$0, 3, 5, 9, 10, 15, 17, 18, 27, 33, 34, 36, 45, 51, 54, 63, \ldots;$$

they form sequence A175468 in the OEIS [18].

In what follows, when we refer to the length of a generalized binary square, we mean the length including the leading zeroes. Thus, 9 is a generalized binary square of length 6 (and not 4).

▶ **Lemma 11.**

(a) *Every length-n integer, $n \geq 7$, n odd, is the sum of 3 generalized binary squares, of lengths $n + 1$, $n - 1$, and $n - 3$.*

(b) *Every length-n integer, $n \geq 8$, n even, is the sum of 3 generalized binary squares, of lengths n, $n-2$, and $n-4$.*

Proof. We use a very similar proof strategy as in the proof of Lemma 5. We drop the requirement that the most significant digit of our guessed squares be 1, thus allowing for generalized binary squares. Note that the square of length $n+1$ in part (a) must start with a 0.

The `ULTIMATE` code proving our result can be found in the files `OddGenSquareConjecture.ats` and `EvenGenSquareConjecture.ats` at `https://cs.uwaterloo.ca/~shallit/papers.html`; there one can also find the generators `OddGeneralizedSquares.cpp` and `EvenGeneralizedSquares.cpp`. The final machines for the odd-length and even-length cases have 132 and 263 states respectively. ◀

We thus have the following theorem:

▶ **Theorem 12.** *Every natural number $N > 7$ is the sum of 3 generalized binary squares.*

Proof. For $7 < N < 64$ the result can be easily verified. Otherwise, we use Lemma b (a) is an odd-length binary number and Lemma b (b) if it is even. ◀

7 Further work

Numerical evidence suggests the following two conjectures:

▶ **Conjecture 13.** Let α_3 denote the lower asymptotic density of the set S_3 of natural numbers that are the sum of three binary squares. Then $\alpha_3 < 0.9$.

We could also focus on sums of *positive* binary squares. (For the analogous problem dealing with ordinary squares, see, e.g., [5, Chapter 6].) It seems likely that our method could be used to prove the following result.

▶ **Conjecture 14.** Every natural number > 1772 is the sum of exactly four positive binary squares. There are 112 exceptions, given below:

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 25, 27, 28, 29, 30, 32, 34, 35,$$
$$37, 39, 41, 42, 44, 46, 47, 49, 51, 53, 56, 58, 62, 65, 67, 74, 83, 88, 95, 100, 104, 107, 109, 113, 116,$$
$$122, 125, 131, 134, 140, 143, 148, 149, 155, 158, 160, 161, 167, 170, 173, 175, 182, 184, 368, 385,$$
$$402, 407, 419, 424, 436, 441, 458, 475, 492, 509, 526, 543, 552, 560, 569, 587, 599, 608, 613,$$
$$620, 625, 638, 647, 653, 671, 686, 698, 713, 1508, 1541, 1574, 1607, 1640, 1673, 1706, 1739, 1772.$$

Other interesting things to investigate include estimating the number of distinct representations of N as a sum of four binary squares, both in the case where order matters, and where order does not matter. These are sequences A290335 and A298731 in the OEIS, respectively.

In recent work [9] it was proved, using a combinatorial and number-theoretic approach, that the binary k'th powers form an asymptotic basis of finite order for the multiples of $\gcd(k, 2^k - 1)$. However, the constant obtained thereby is rather large.

References

1 W. D. Banks. Every natural number is the sum of forty-nine palindromes. *INTEGERS — Electronic J. Combinat. Number Theory*, 16, 2016. #A3.

2 J. Cilleruelo, F. Luca, and L. Baxter. Every positive integer is a sum of three palindromes. *Math. Comp.*, 2017. Published electronically at `http://dx.doi.org/10.1090/mcom/3221`.

3 R. C. Crocker. On the sum of two squares and two powers of k. *Colloq. Math.*, 112:235–267, 2008.

4 W. J. Ellison. Waring's problem. *Amer. Math. Monthly*, 78:10–36, 1971.

5 E. Grosswald. *Representations of Integers as Sums of Squares*. Springer-Verlag, 1985.

6 M. Heizmann, D. Dietsch, M. Greitschus, J. Leike, B. Musa, C. Schätzle, and A. Podelski. Ultimate automizer with two-track proofs. In M. Chechik and J.-F. Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems — 22nd International Conference, TACAS 2016*, volume 9636 of *Lecture Notes in Computer Science*, pages 950–953. Springer-Verlag, 2016.

7 M. Heizmann, J. Hoenicke, and A. Podelski. Software model checking for people who love automata. In N. Sharygina and H. Veith, editors, *Computer Aided Verification — 25th International Conference, CAV 2013*, volume 8044 of *Lecture Notes in Computer Science*, pages 36–52. Springer-Verlag, 2013.

8 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

9 D. M. Kane, C. Sanna, and J. Shallit. Waring's theorem for binary powers. Preprint, available at `https://arxiv.org/abs/1801.04483`, 2018.

10 J.-L. Lagrange. Démonstration d'un théoréme d'arithmétique. *Nouv. Mém. Acad. Roy. Sc. de Berlin*, pages 123–133, 1770. Also in *Oeuvres de Lagrange*, **3** (1869), pp. 189–201.

11 C. J. Moreno and S. S. Wagstaff, Jr. *Sums of Squares of Integers*. Chapman and Hall/CRC, 2005.

12 M. B. Nathanson. *Additive Number Theory: The Classical Bases*. Springer-Verlag, 1996.

13 M. B. Nathanson. *Elementary Methods in Number Theory*. Springer-Verlag, 2000.

14 D. Platt and T. Trudgian. On the sum of two squares and at most two powers of 2. Preprint, available at `https://arxiv.org/abs/1610.01672`, 2016.

15 A. Rajasekaran. Using automata theory to solve problems in additive number theory. Master's thesis, School of Computer Science, University of Waterloo, 2018.

16 A. Rajasekaran, J. Shallit, and T. Smith. Sums of palindromes: an approach via nested-word automata. Preprint, available at `https://arxiv.org/abs/1706.10206`, 2017.

17 A. Rajasekaran, J. Shallit, and T. Smith. Sums of palindromes: an approach via automata. In R. Niedermeier and B. Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*, Leibniz International Proceedings in Informatics, pages 54:1–54:12. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2018.

18 N. J. A. Sloane. The on-line encyclopedia of integer sequences. Available at `https://oeis.org`, 2016.

19 C. Small. Waring's problem. *Math. Mag.*, 50:12–16, 1977.

20 R. C. Vaughan. *The Hardy–Littlewood Method*, volume 125 of *Cambridge Tracts in Mathematics*. Cambridge University Press, 2nd edition, 1997.

21 R. C. Vaughan and T. Wooley. Waring's problem: a survey. In M. A. Bennett, B. C. Berndt, N. Boston, H. G. Diamond, A. J. Hildebrand, and W. Philipp, editors, *Number Theory for the Millennium. III*, pages 301–340. A. K. Peters, 2002.

A Two-Sided Error Distributed Property Tester For Conductance

Hendrik Fichtenberger
TU Dortmund, Dortmund, Germany
hendrik.fichtenberger@tu-dortmund.de
https://orcid.org/0000-0003-3246-5323

Yadu Vasudev
Indian Institute of Technology Madras, Chennai, India
yadu@cse.iitm.ac.in
https://orcid.org/0000-0001-7918-7194

Abstract

We study property testing in the distributed model and extend its setting from testing with one-sided error to testing with two-sided error. In particular, we develop a two-sided error property tester for general graphs with round complexity $\mathcal{O}(\log(n)/(\epsilon\Phi^2))$ in the CONGEST model, which accepts graphs with conductance Φ and rejects graphs that are ϵ-far from having conductance at least $\Phi^2/1000$ with constant probability. Our main insight is that one can start $\text{poly}(n)$ random walks from a few random vertices without violating the congestion and unite the results to obtain a consistent answer from all vertices. For connected graphs, this is even possible when the number of vertices is unknown. We also obtain a matching $\Omega(\log n)$ lower bound for the LOCAL and CONGEST models by an indistinguishability argument. Although the power of vertex labels that arises from two-sided error might seem to be much stronger than in the sequential query model, we can show that this is not the case.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases property testing, distributed algorithms, conductance

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.19

Related Version `https://arxiv.org/abs/1707.06126`

Funding 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 n° 307696.

Acknowledgements We would like to thank Gopal Pandurangan for pointing out related work [7, 21], and we would like to thank Pan Peng for inspiring discussions on spectral graph theory. We are grateful for the helpful comments of anonymous reviewers.

1 Introduction

Property testing algorithms derive approximate decisions by probing a sublinear part of the input only. A tester for a graph property $\mathcal{P}$ is a randomized algorithm that, with high constant probability, accepts graphs that have the property $\mathcal{P}$ and rejects graphs that are ϵ-far from having the property $\mathcal{P}$, that is, at least an ϵ-fraction of the edges have to be modified to make the graph have the property $\mathcal{P}$. Two-sided error testers may err on all graphs, while one-sided error testers have to present a witness when rejecting a graph. See [12, 13, 14] for introductions and surveys.

Testing graph properties in the classic, sequential computing model has been studied quite extensively. Property testing in the distributed CONGEST model was first studied by Brakerski and Patt-Shamir [1] and later more thoroughly by Censor-Hillel et al. [2]. In this model, each vertex of the graph is equipped with a processor that has a unique identifier of size $\mathcal{O}(\log n)$ (alternatively, one may define the model such that vertices pick their identifier randomly) and it knows only its neighboring vertices. The vertices of the graph communicate with each other in synchronized rounds such that in each round only communication of length $\mathcal{O}(\log n)$ is allowed on every edge. Finally, every vertex casts a vote and a decision rule is applied on all votes to derive the answer of the tester. The complexity measure is the amount of rounds required to test the property. Edge congestion and round complexity strictly limit the amount of information on the whole graph that a single vertex can gather. In [2], it is shown that many one-sided error testers for dense graphs carry over from the sequential to the distributed setting. Furthermore, tight logarithmic bounds for testing bipartiteness and cycle-freeness in bounded degree graphs are proved. In [8, 11], subgraph-freeness is studied for subgraphs on at most five vertices, trees and cliques.

1.1 Our Results

We extend the study of distributed property testing to testers with two-sided error. In particular, we present a two-sided error distributed testing algorithm in the CONGEST model for conductance.

▶ **Theorem 1.** *Testing whether a graph $G = (V, E)$ has conductance at least Φ or is ϵ-far from having conductance at least $\Phi^2/1000$ with two-sided error has complexity $\mathcal{O}(\log(|V| + |E|)/(\epsilon\Phi^2))$ in the CONGEST model.*

In contrast to previous one-sided error testers, our tester can be implemented such that all vertices accept or all vertices reject (see also the discussion of decision rules in Section 5). Furthermore, we prove that the size of the input graph is not required to be known a priori to perform the test if it is connected. On the other hand, there exists no tester for disconnected graphs if no prior knowledge of the graph is assumed at all. Since communication between two connected components is not allowed, we cannot distinguish a graph G with high conductance from a graph G' that is composed of two isolated copies of G.

In the setting of property testing, we aim for an efficient method to check whether the input has the desired property or is at least close to it. For example, we might run the tester for conductance once in a while on a peer-to-peer network to check whether the topology has changed significantly such that efficient communication is no longer possible. Therefore, we think of Φ as a parameter with no or only weak dependence on n.

We complement this result by showing that any distributed tester with this gap requires $\Omega(\log(n + m))$ rounds of communication regardless of the final decision rule.

▶ **Theorem 2.** *Let ϵ, $d > 0$ be constants, and let Φ, c be constants that depend on d. Testing whether a d-regular graph $G = (V, E)$ has conductance at least Φ or is ϵ-far from having conductance at least $c\Phi^2$ requires $\Omega(\log(|V| + |E|))$ rounds of communication in the LOCAL and CONGEST models.*

1.2 Related Work

In the classic, sequential setting of property testing, the problem of testing conductance in bounded degree graphs was first studied in [6, 15]. Kale and Seshadhri [16] and Nachmias and Shapira [22] give $\tilde{\mathcal{O}}(\Phi^{-2}\sqrt{n})$-query two-sided error testers that accept graphs that have

conductance at least Φ and reject graphs that are ϵ-far from having conductance at least $\Omega(\Phi^2)$. An algorithm for testing the cluster structure of graphs has been given in [5]. Testing conductance in unbounded degree graphs in the stronger rotation map model with query complexity roughly $\tilde{\mathcal{O}}(\Phi^{-2}\sqrt{m})$ was studied in [17], and testing conductance properties restricted to small sets has been studied in [18]. However, the optimal query complexity for testing conductance in general graphs of unbounded degree is still open.

In the CONGEST model, random walks have been analyzed by Censor-Hillel et al. [2] to design a tester for bipartiteness. The idea there is to perform a constant number of random walks from every vertex and to test if two such walks intersect in a cycle of odd length. It is crucial to bound the number of random walks that traverse an edge in one step. In contrast, we can perform polynomially many random walks from a constant number of vertices in the graph and we can afford that all walks traverse the same edge simultaneously.

Distributed random walks have also been studied in [7] and [21]. In particular, [21] show that one can approximate the mixing time τ_v of a vertex v in $\mathcal{O}(\tau_v \log n)$ rounds by running $\text{poly}(n)$ random walks v and comparing their endpoint distribution to the stationary distribution. The graph's mixing time $\tau = \max_v \tau_v$ relates to the conductance by $c_1\Phi^2/\log n \leq 1/\tau \leq c_2\Phi$. A straightforward approach based on [21] leads to an $\mathcal{O}(n \log^2(n)/\Phi^2)$ round algorithm for *approximating* Φ with a multiplicative gap of $\Theta(\Phi/\log n)$. In comparison, our *tester's* gap does not depend on n and its complexity is only logarithmic in n. One reason is that if the graph is *far from* having conductance $\Omega(\Phi^2)$, there exist many vertices with large mixing times compared to the case that the graph has conductance Φ (see the proof of Theorem 1 for details). This is not necessarily the case if the graph is not ϵ-far from having conductance $\Omega(\Phi^2)$.

1.3 Overview

The classic tester [16] for bounded degree graphs exploits that random walks converge rapidly to the uniform distribution in graphs with high conductance and they mix slowly for at least a small fraction of start vertices in graphs that are ϵ-far from having high conductance. This boils down to approximating the collision probability of random walks for a few start vertices. However, in general graphs the stationary distribution is not uniform, and collisions at vertices with high degree are more important than at vertices with low degree.

In the distributed model, one has to take care of edge congestion, too. Simulating $\omega(1)$ random walks while keeping them distinguishable is very costly. A key observation is that for approximating the discrepancy, it is sufficient to maintain only some statistics of the random walks, which reduces the congestion significantly: one has to transfer only the number of random walks that pass through an edge for each of a constant number of start vertices.

For the lower bound, we construct two distributions on graphs of high and low conductance respectively such that the vertices' neighborhoods of radius $\Omega(\log n)$ are isomorphic. The idea is that within only $\mathcal{O}(\log n)$ rounds, all vertices receive the same information up to isomorphism and therefore cannot distinguish between the two distributions. However, since the tester is allowed to have two-sided error and it only needs to distinguish graphs that have the property from graphs that are far, it is plausible that it can glean information about the two distributions from the vertex labels of the subgraphs it has seen. For example, certain sets of labeled subgraphs might be present in (many of) the graphs with high conductance that are absent in (many of) the graphs with low conductance. Lower bounds for sequential testers usually argue that with high probability, one can assume that all probes are independent. Since the local views of the vertices in the distributed model always have a large overlap, we base our argument on a reduction and a random labeling argument instead.

2 Preliminaries

Let $G = (V, E)$ be a graph and let $S, T \subseteq V, S \cap T = \emptyset$ be sets of vertices. We denote $|V|$ and $|E|$ by n and m respectively for the graph G at hand. Let $d(v)$ be the degree of vertex $v \in V$. We write $\bar{S}$ for the set $V \backslash S$. The set of vertices in $\bar{S}$ that are adjacent to some $u \in S$ is denoted by $\Gamma(S)$. The *volume* of S is the sum of degrees of vertices in S, that is, $\mathrm{vol}(S) := \sum_{v \in S} d(v)$. The cut between S and T is denoted by $E(S, T) = E \cap (S \times T)$. For a set $S \subseteq V$ such that $\mathrm{vol}(S) \leq \mathrm{vol}(\bar{S})$, the conductance of S is $\mathrm{cond}(S) = |E(S, \bar{S})|/\mathrm{vol}(S)$. The conductance of G is defined as $\Phi(G) = \min_{S \subset V, \mathrm{vol}(S) \leq \mathrm{vol}(\bar{S})} \mathrm{cond}(S)$.

2.1 Distributed Computing

In the distributed computational model, a computation network $G = (V, E)$ with a processor associated to each vertex $v \in V$ is given. Each processor v has access to numbered communication channels to its neighbors in G. Additionally, it may have some specific input $I(v)$. The computation operates in synchronized rounds that are divided into three phases. In each round, each processor may do some local computation first, then it may send a message to each of its neighbors, and finally it receives the messages sent from its neighbors.

▶ **Definition 3** (Distributed Computational Model, DCM). *Let $G = (V, E)$ be a graph and $p_G = (p_v)_{v \in V}$ with $p_v : [d(v)] \to \Gamma(v)$ be a bijective function, that is, an adjacency list representation of G. Let $I : V \to \{0, 1\}^*$ be a mapping from the set of vertices to bit strings. An instance of the distributed computational model on G, p_G and I, $\mathrm{DCM}(G, p_G, I)$, is defined as follows. Each vertex $v \in V$ is a processor that has communication access to its neighbors $p_v(1), \ldots, p_v(d(v))$ by ports numbered $1, \ldots, d(v)$. The model operates in synchronized rounds, where each round r consists of three phases: (i) Each vertex performs local computation, (ii) each vertex v sends a message to its neighbor $p_v(i)$, denoted $s_r(v, i)$, for all $i \in d(v)$, (iii) each vertex u receives a message from its neighbor $p_u(j)$, for all $j \in d(u)$. The distributed computational model DCM is the set of all instances $\mathrm{DCM}(G, p_G, I)$.*

The LOCAL model is the subset of the DCM such that for each vertex $v \in V$, the input $I(v)$ is only n and a numerical vertex identifier from $[n^c]$ for some universal constant c. The CONGEST model is the subset of the LOCAL model such that the size of each message $s_r(v, i)$ is restricted to $c \log n$ bits.

A distributed network decision algorithm $\mathrm{DNDA}(\mathcal{A}, O)$ is an algorithm $\mathcal{A}$ that is deployed to the vertices of a DCM to decide a property of an instance of the model. In particular, the output of $\mathcal{A}$ is a single bit, and the final decision is obtained by applying a function $O(\cdot)$ to the union of all vertices' answers.

▶ **Definition 4** (Distributed Network Decision Algorithm). *Let $\mathcal{A}$ be an algorithm that takes a bit string as input and outputs a single bit, and let $O : \{0, 1\}^* \to \{0, 1\}$ be a function. When the distributed network decision algorithm $\mathrm{DNDA}(\mathcal{A}, O)$ is run on an instance $\mathrm{DCM}(G, p_G, I)$, a copy of $\mathcal{A}$ is deployed to every vertex v with input $I(v)$ and run in parallel as described in Definition 3. We refer to the copy of $\mathcal{A}$ deployed to v by $\mathcal{A}_v$. When every vertex v_i has terminated its computation with output bit b_{v_i}, the decision of $\mathrm{DNDA}(\mathcal{A}, O)$ is $O(b_{v_1} b_{v_2} \cdots b_{v_n})$.*

2.2 Distributed property testing

A distributed property testing algorithm is a distributed algorithm as defined in Definition 4 that accepts graphs that have a property, and rejects graphs that are ϵ-far from the property. We say that a graph G with n vertices and m edges is ϵ-far from a property $\mathcal{P}$ if at least ϵm

edges of G have to be modified to make the new graph have the property $\mathcal{P}$. A one-sided error distributed ϵ-tester accepts all graphs with property $\mathcal{P}$, whereas it rejects, with probability at least $2/3$, all graphs that are ϵ-far from the property. In this paper, we give a two-sided (error) property tester that is also allowed to err, with probability at most $1/3$, when the graph has the property.

▶ **Definition 5** (Two-sided tester). *A two-sided (error) distributed ϵ-tester for a property $\mathcal{P}$ is a* DNDA$(\mathcal{A}, O)$*, where $O(b_{v_1} b_{v_2} \cdots b_{v_n}) = 1$ iff $b_{v_i} = 1$ for all $v_i \in V$ such that the following conditions hold: (i) if G has the property $\mathcal{P}$, then, with probability at least $2/3$, $b_{v_i} = 1$ for all $v_i \in V$, (ii) if G is ϵ-far from $\mathcal{P}$, then, with probability at least $2/3$, there exists a $v_i \in V$ such that $b_{v_i} = 0$.*

The guarantees given by our tester are actually a bit stronger in the sense that the tester can be modified such that either $b_v = 0$ or $b_v = 1$ for all $v \in V$ simultaneously.

3 Testing Using Random Walks

In this section we will present the distributed algorithm for testing whether a graph has conductance at least Φ or is ϵ-far from having conductance at least $\Phi^2/1000$. The core idea of the algorithm is to perform random walks from a small set of vertices and test whether these walks converge to the stationary distribution rapidly, which is the case for graphs with high conductance. It is based on the ideas of Kale and Seshadhri [16] and Goldreich and Ron[15].

Before we describe the algorithm, we give a few useful definitions and lemmas. A *lazy random walk* on a graph $G = (V, E)$ on n vertices is a random walk on the graph, where at each vertex v the walk chooses to stay at v with probability $1/2$ and chooses a neighbor u with probability $1/(2d(v))$. The walk matrix $W = [w_{uv}]_{u,v\in[n]}$ is defined by $w_{uv} := 1/2$ if $u = v$, $w_{uv} := 1/(2d(v))$ if $u \neq v, (u, v) \in E$ and $w_{uv} := 0$ otherwise. Notice that for irregular graphs, W is not symmetric. To analyze these random walks, one can draw on the *normalized walk matrix*, which is a symmetric matrix similar to W. The normalized walk matrix N of G is $D^{-1/2} W D^{1/2}$, where D is the diagonal matrix with $D(u, u) := d(u)$.

Since N is a real symmetric matrix, it has real eigenvalues. Let $1 = \mu_1, \ldots, \mu_n \geq 0$ be its eigenvalues, and let $\{\vec{f_i}\}_{i\in[n]}$ be its orthonormal eigenbasis. We have $\vec{f_1} = \sqrt{\vec{\pi}}$, where $\vec{\pi}$ is the random walk's *stationary distribution*. In particular, it is well known that $\vec{\pi}_v = d(v)/(2m)$. For more details on spectral graph theory, refer to [4].

It is well known that graphs with high conductance have small diameter.

▶ **Lemma 6** ([3, cf. Theorem 2]). *Let $G = (V, E)$ be a graph with conductance Φ. The diameter of G is at most $(3/\Phi)\ln(m)$.*

Sinclair [24] proved that there is a tight connection between the conductance and the mixing time of random walks. In particular, the L_2 distance of any starting distribution $\vec{\pi'}$ to $\vec{\pi}$ after $\Phi^{-2} \log n$ steps is $\mathcal{O}(1/n)$.

▶ **Lemma 7** ([24, cf. Theorem 2.5]). *Let $G = (V, E)$ be a graph with conductance Φ. For any starting distribution $\vec{\pi'}$, it holds that $\|W^\ell \vec{\pi'} - \vec{\pi}\|_2 \leq \left(1 - \Phi^2/2\right)^\ell$.*

3.1 Algorithm

We discuss the algorithm from a global point of view instead of describing an algorithm $\mathcal{A}$ for a single vertex to provide a better explanation of the interactions between vertices.

Lemma 6 implies that if the graph has high conductance, then it has diameter $\mathcal{O}(\log n/\Phi)$, which we want to use as an assumption in the algorithm later. To test the diameter, we perform a BFS of depth $\mathcal{O}(\log n/\Phi)$ of the graph starting from an arbitrary vertex. Initially, every vertex chooses itself as root of the BFS and announces itself as root to all its neighbors. To break the symmetry between the vertices, a vertex accepts every vertex with a lower identifier than its current root as new root and forwards its messages. If the diameter is $\mathcal{O}(\log n/\Phi)$, a unique root has been chosen after $\mathcal{O}(\log n/\Phi)$ rounds and every vertex knows its parent and its children in the BFS tree. Otherwise, at least one of the remaining candidates will reject. Algorithm 3 gives a formal description of the BFS.

From now on, assume that the diameter is $\mathcal{O}(\log n/\Phi)$. Using the previously computed BFS tree, we can compute the number of edges in the graph by summing up vertex degrees from the leaves to the root and transmitting this number to all vertices afterwards. Algorithm 4 describes the procedure in detail.

The key technical lemma from [16] for bounded degree graphs states that if a graph is ϵ-far from having conductance $\Omega(\Phi^2)$, then there exists a $\Omega(\epsilon)$-fraction of *weak* vertices such that random walks starting from these vertices converge only slowly to the stationary distribution. Therefore, a sample $S \subset V$ of size $\mathcal{O}(1)$ will likely contain a weak vertex (technically, we sample each vertex v independently into S with probability $\Theta(d(v)/\epsilon m)$. By Markov's inequality, we may reject if S is much larger than its expected size.). We extend this lemma to unbounded degree graphs. Then, we perform $N = n^{100}$ random walks of length $\ell = 40/\Phi^2 \cdot \log n = \mathcal{O}(\log n/\Phi^2)$ starting from each of the vertices in S to approximate the rate of convergence.

The crucial point here is that in each round of the algorithm, we do not send the full trace of every random walk. Instead, for every origin $v \in S$, every vertex $u \in V$ only transmits the total number of random walks that are leaving it through an edge (u, w) to its neighbor $w \in \Gamma(u)$. Since the size of S is constant, we require $\mathcal{O}(\log n)$ bits per edge to communicate this. On the other hand, this information is sufficient because we are only interested in the distribution of endpoints of the lazy random walks for every $v \in S$. Algorithm 2 gives a formal description of this procedure. Finally, the estimated distribution of endpoints is used to approximate the distance to the stationary distribution for each $v \in S$. The whole algorithm is summarized in Algorithm 1.

First, we show that either the estimates $\widehat{W}^\ell_{v,u}$ of Algorithm 2 are good or the algorithm rejects in line 14 because G has low conductance.

▶ **Lemma 8.** *Consider Algorithm 2. For every $v, u \in V$, it holds with probability at least $1 - m^{-10}$ that (i) $|\widehat{W}^\ell_{v,u} - W^\ell(v,u)| \leq m^{-20}$ and, conditioned on the previous, (ii) if $\widehat{W}^\ell_{v,u} < m^{-2}$ then G has conductance less than Φ.*

Proof. We have $E[\widehat{W}^\ell_{v,u}] = W^\ell(v,u)$. By Hoeffding's inequality, it holds that

$$\Pr[|\widehat{W}^\ell_{v,u} - E[\widehat{W}^\ell_{v,u}]| \geq m^{-10}] \leq 2\exp\left(-\frac{N}{3m^{40}}\right) \leq m^{-10}\,. \tag{1}$$

Condition on $|\widehat{W}^\ell_{v,u} - E[\widehat{W}^\ell_{v,u}]| < m^{-20}$, which happens with probability at least $1 - 1/m^{10}$. If $\widehat{W}^\ell_{v,u} < m^{-2}$, then

$$W^\ell(v,u) = E[\widehat{W}^\ell_{v,u}] < \widehat{W}^\ell_{v,u} + m^{-20} = m^{-2} + m^{-10} < 2m^{-2}\,.$$

Let $\pi' = \mathbb{1}_v$. We bound $\|W^\ell \vec{\boldsymbol{\pi}'} - \vec{\boldsymbol{\pi}}\|_2$ from below.

$$\|W^\ell \vec{\boldsymbol{\pi}'} - \vec{\boldsymbol{\pi}}\|_2 \geq |W^\ell(v,u) - d(u)/2m| \geq -(2m^{-2} - 1/(2m)) \geq 1/(4m)\,.$$

By the contrapositive of Lemma 7, G has conductance less than Φ. ◀

Algorithm 1 Conductance tester.

procedure TestConductance($G = (V, E)$, n, Φ)
  BFS(G, $6/\Phi \ln n$) ▷ construct BFS of depth $6/\Phi \log n$, Algorithm 3
  if BFS visited less than n vertices **then reject**
  $m \leftarrow$ AggregateSum(G, $12/\Phi \ln n$, f) ▷ $f(v) := d(v)/2$, Algorithm 4
  let *every vertex* $v \in V$ **do**
    with probability $\min\{1, 10^4 d(v)/2\epsilon m\}$, mark v
  $S \leftarrow$ marked v, $r \leftarrow$ root of BFS tree
  if $|S| > 10^5/\epsilon$ **then reject**
  RandomWalk(G, S, $40/\Phi^2 \cdot \log n$, n^{100}) ▷ compute local $s_{v,u}$, Algorithm 2
  for all $v \in S$ **do** $s_v \leftarrow$ AggregateSum(G, $12/\Phi \ln n$, f) ▷ $f(u) := s_{v,u}$, Alg. 4
  let *every vertex* $v \in V$ **do**
    if $s_v \leq m^{-15}$ for all $v \in S$ **then accept**
    else reject

Algorithm 2 Perform random walks.

procedure RandomWalk(G,S,ℓ,N)
  let *every vertex* $v \in S$ **do**
    sample $u_1, \cdots, u_N$ independently according to $W\vec{e_v}$
    for all $w \in \Gamma(v)$ **do** send $(v, v, |\{i \mid w = u_i\}|)$ to w
  for ℓ rounds, **let** *every vertex* x **do**
    receive $(u_1, x'_1, k_1), (u_2, x'_2, k_2), \ldots$
    for all $v \in S$ **do**
      sample $u_1, \cdots, u_{n_v}$ independently according to $W\vec{e_x}$, where $n_v = \sum_{u_i = v} k_i$
      for all $w \in \Gamma(v)$ **do** send $(v, x, |\{i \mid w = u_i\}|)$ to w
  let *every vertex* $u \in V$ **do**
    receive $(u_1, x'_1, k_1), (u_2, x'_2, k_2), \ldots$
    for all $v \in S$ **do**
      $\widehat{W}^{\ell}_{v,u} \leftarrow \sum_{v_i = v} n_i/N$
      if $\widehat{W}^{\ell}_{v,u} \leq 2m^{-2}$ **then reject**
      $s_{v,u} \leftarrow (\widehat{W}^{\ell}_{v,u} - \frac{d(v)}{2m})^2$

Algorithm 3 Construct BFS tree.

procedure BFS(G, D)
  let *every vertex* v **do**
    $T_v \leftarrow (v, \cdot)$ ▷ set root to itself, parent to empty
    $minid \leftarrow v$
    send (v, v) to every neighbor $u \in \Gamma(v)$
  for D rounds, **let** *every vertex* w **do**
    $R_v \leftarrow \{(v', u') \text{ received} \mid u' \in \Gamma(w)\}$
    $(v, u) \leftarrow \arg\min_{(v', u') \in R_v} v'$
    if $T_w = (\cdot)$ or $v' < minid$ **then**
      $T_w \leftarrow (v, u)$ ▷ set root to v, parent to u
      send (v, w) to all neighbors $\neq u$

Algorithm 4 Aggregate sum of vertex values and propagate it to all vertices.

Require: $\forall v : v$ has local information $f(v)$
Ensure: $\forall v : v$ has information $\sum_{u \in V} f(u)$
1: **procedure** AggregateSum(G, D, $f : V \to \mathbb{R}$)
2: **for** D rounds, **let** *every vertex* v **do**
3: **if** v received partial sums s_u from all its children u in BFS tree **then**
4: $s_v \leftarrow f(v) + \sum_u s_u$
5: send s_v to parent in BFS tree
6: **let** *vertex root* r *of BFS tree* **do**
7: send total sum $s = \sum_v f(v)$ to all children
8: **for** D rounds, **let** *every vertex* v **do**
9: **if** v received total sum s from its parent **then**
10: send s_v to all children in BFS tree
11: **return** s_v ▷ consider s_v to be the output of the algorithm

Furthermore, Lemma 8 implies that the estimates s_v in Algorithm 1 (see line 10) are also good if Algorithm 2 has not rejected before.

▶ **Lemma 9.** *Consider Algorithm 1. With probability at least* $1 - m^{-8}$ *it holds for every* $v \in S$ *in line 10 that* $\left| \|W^\ell(v, \cdot) - \vec{\boldsymbol{\pi}}\|_2^2 - s_v \right| \leq 3m^{-19}$.

Proof. Let $v \in S$. We have the following equality for the discrepancy of the distribution of the random walks' endpoints that start at v and the stationary distribution:

$$\|W^\ell(v, \cdot) - \vec{\boldsymbol{\pi}}\|_2^2 = \sum_{u \in V} \left(W^\ell(v, u) - \frac{d(u)}{2m} \right)^2 . \tag{2}$$

By Lemma 8, we know that for every $u \in V$ we have $|\widehat{W}^\ell_{u,v} - W^\ell(v, u)| \leq m^{-20}$with probability $1 - 1/m^9$. Then, by the triangle inequality, $|(W^\ell(v, u) - \frac{d(u)}{2m})^2 - s_{v,u}| \leq 3m^{-20}$. Combining this with Eq. (2), a union bound over all $u \in V$ implies that with probability at least $1 - n \cdot m^{-10} \geq 1 - m^{-9}$, we have that $\left| \|W^\ell(v, \cdot) - \vec{\boldsymbol{\pi}}\|_2^2 - \sum_{u \in V} s_{v,u} \right| \leq 3m^{-19}$. A union bound over all $v \in S$ gives that with probability at least $1 - |S|/m^{-9} \geq 1 - m^{-8}$, $\left| \|W^\ell(v, \cdot) - \vec{\boldsymbol{\pi}}\|_2^2 - \sum_{u \in V} s_{v,u} \right| \leq 3m^{-19}$ for every $v \in S$. ◀

3.2 Completeness and Soundness

The proof of completeness is a straightforward application of the results from the previous section.

▶ **Lemma 10** (Completeness). *Let* $G(V, E)$ *be a graph with conductance at least* Φ. *Then, with probability at least* $2/3$, *each vertex in* G *returns* ***accept*** *when it runs Algorithm 1.*

Proof. The probability that the algorithm rejects in Line 8 of Algorithm 1 is at most $1/10$, and we assume, for the remainder of the proof, that this event did not occur. If G has conductance at least Φ, then from Lemma 7 we know that $\|W^\ell(\cdot, v) - \vec{\boldsymbol{\pi}}\|_2^2 \leq \left(1 - \Phi^2/2\right)^{2\ell} \leq \exp(-\Phi^2 \ell/2) \leq m^{-20}$ for every vertex v. Lemma 9 implies that with probability at least $9/10$, it holds that $\left| \|W^\ell(\cdot, v) - \vec{\boldsymbol{\pi}}\|_2^2 - s_v \right| \leq 3m^{-19}$. Conditioning on this event, every vertex accepts in line 12 of Algorithm 1. ◀

To complete the analysis of the tester, we show that whenever the graph is ϵ-far from having conductance $\Omega(\Phi^2)$, the tester rejects with probability at least $2/3$. To this end, we

actually show that if the volume of weak vertices is small, then the graph can be converted to another graph G' by modifying at most ϵm edges such that the conductance is $\Omega(\Phi^2)$. The idea of the analysis is due to Kale and Seshadhri [16], who analyzed a classic property tester for testing expansion in graphs with vertex degrees bounded by a constant. We deviate from their analysis where it becomes necessary to take care of arbitrary vertex degrees.

Let a vertex $v \in V$ be called *weak* if $\|W^\ell(v,\cdot) - \vec{\boldsymbol{\pi}}\|_2 > 6m^{-15}$. The following lemma states that if there exists a set of vertices S with small conductance, then there exists a set of weak vertices T whose volume is at least a constant fraction of the volume of S.

▶ **Lemma 11.** *Let $S \subset V$ be such that $\mathrm{vol}(S) \leq \mathrm{vol}(\bar{S})$ and $\mathrm{cond}(S) \leq \delta$. Then, for any $\ell \in \mathbb{N}$ and any $0 < \theta \leq 1/10$, there exists a set $T \subseteq S$ such that $\mathrm{vol}(T) \geq \theta \mathrm{vol}(S)$ and for every $v \in T$, it holds that $\|W^\ell(v,\cdot) - \vec{\boldsymbol{\pi}}\|_2^2 > \frac{1}{80m^7}(1-4\delta)^{2\ell}$.*

The proof can be found in the arXiv version [10]. We can use Lemma 11 to separate weak vertices from the remaining graph.

▶ **Lemma 12.** *Let $G = (V,E)$ be a graph. If the volume of weak vertices in G is at most $(1/100)\epsilon m$, then there is a partition of V into $P \cup \bar{P}$ such that $\mathrm{vol}(P) \leq \epsilon m/10$ and $\Phi(G[\bar{P}]) \geq \Phi^2/256$.*

Proof. We partition the graph recursively into two sets $(P, \bar{P})$. At the beginning, $P_0 = \emptyset$ and $\bar{P}_0 = V$. As long as there is a cut $(C_i, \bar{C}_i)$ in $\bar{P}_{i-1}$ in step i with $\mathrm{vol}(C_i) \leq \mathrm{vol}(\bar{C}_i)$ and $E(C_i,\bar{C}_i)/\mathrm{vol}(C_i) \leq \Phi^2/256$, we set $P_i = P_{i-1} \cup C_i$ and $\bar{P}_i = V \setminus P_i$. We continue this until we don't find such a cut or the condition $\mathrm{vol}(P_{i+1}) \leq \mathrm{vol}(\bar{P}_{i+1})$ would be violated. The number of edges going across the cut $(P,\bar{P})$ is at most $\sum_i |E(C_i,\bar{C}_i)|$. Therefore, $|E(P,\bar{P})| \leq \frac{\Phi^2}{256}\sum_i \mathrm{vol}(C_i) \leq \frac{\Phi^2}{256}\mathrm{vol}(P)$.

Now, assume that $\mathrm{vol}(P) > (1/10)\epsilon m$. Lemma 11 implies that there exists $P' \subseteq P$ such that $\mathrm{vol}(P') \geq \frac{1}{10}\mathrm{vol}(P) > \epsilon m/100$ (where $\theta = 1/10$) and for all $v \in P'$ we have $\|W^\ell(v,\cdot) - \vec{\boldsymbol{\pi}}\|_2 > \frac{1}{80m^7}(1-4\Phi^2/256)^{2\ell} > \frac{1}{80m^{10}}$. This means that P' contains only weak vertices and has volume at least $\epsilon m/100$, which contradicts our assumption that the volume of weak vertices in G is at most $\epsilon m/100$. Therefore, $\mathrm{vol}(P) \leq \epsilon m/10$ when the partitioning terminates. Hence $\Phi(G[\bar{P}]) \geq \Phi^2/256$. ◀

Finally, the following lemma states that few edge modifications in a graph with separated weak vertices are sufficient to make it a graph with high conductance.

▶ **Lemma 13** ([18, Lemma 9])**.** *Let $G = (V,E)$ be a graph. If there exists a set $P \subseteq V$ such that $\mathrm{vol}(P) \leq \epsilon m/10$ and the subgraph $G[V \setminus P]$ is a Φ'-expander, then there exists an algorithm that modifies at most ϵm edges to get a $\Phi'/3$-expander $G' = (V,E')$.*

Combining the results on the separation of weak vertices and patching the graph (Lemmas 11 to 13) and approximating the endpoint distribution (Lemmas 8 and 9), we prove the soundness of the algorithm.

▶ **Lemma 14** (Soundness)**.** *Let $G(V,E)$ be a graph. If G is ϵ-far from having conductance at least $\Phi^2/768$, then, with probability at least $2/3$, each vertex in G returns **reject** when it runs Algorithm 1.*

Proof. First we note that if the volume of weak vertices is less than $\epsilon m/100$, then by Lemmas 12 and 13, the graph is ϵ-close to having conductance at least $\Phi^2/768$. Therefore, the volume of weak vertices is at least $\epsilon m/100$. Each vertex v is contained in S with probability $\Theta(d(v)/\epsilon m)$. Hence, the expected number of weak vertices that are present in

the sample S is at least 100. Therefore, with probability at least 9/10, at least one weak vertex is sampled in S.

If $W^\ell(v,u) < m^{-2}$ for some $v \in S, u \in V$, then with probability at least 9/10, $\widehat{W}^\ell_{v,u} < 2m^{-2}$ by Lemma 8. In this case, the algorithm will reject in line 14 of Algorithm 2. If $W^\ell(v,u) \geq m^{-2}$ for all $v \in S, u \in V$, then with probability at least 9/10, it holds that $\left|\|W^\ell(v,\cdot) - \vec{\boldsymbol{\pi}}\|_2 - s_v\right| \leq 3m^{-19}$ for every $v \in S$ by Lemma 9. Since at least one vertex $v \in S$ is weak, that is, $\|W^\ell(v,\cdot) - \vec{\boldsymbol{\pi}}\|_2 > 6m^{-15}$, the algorithm rejects in line 13 of Algorithm 1. ◀

3.3 Unknown Size of the Graph

We describe how to get rid of the assumption that the size n of the graph G is known to the tester if G is connected. Note that without any prior knowledge of G, no distributed tester can distinguish between a graph with conductance Φ and two distinct copies of it (the latter is ϵ-far from being a graph with conductance Φ^c for $\epsilon < \Phi^c/2$, $c \geq 1$).

First, we describe a slightly simpler version of the final algorithm. In the setting of the simpler algorithm, we mark a single vertex that will initiate the test and will also give the final answer of the tester. We call this vertex the maintainer (of the graph). The algorithm can be easily adapted to the CONGEST model.

Let $v \in V$ be a fixed vertex. The algorithm either makes n available at all vertices and runs Algorithm 1 afterwards or v rejects because G does not have conductance Φ. If G has conductance Φ, the algorithm never rejects.

We start with an initial set $S = \{v\}$ that is grown in two phases. In the first phase, we extend S to $S \cup \Gamma(S)$ as long as $\mathrm{cond}(S) \geq \Phi$. In particular, v starts a BFS and in every round, the vertices in the last level report their degree and the number of neighbors outside of S to their parents. Similar to Algorithm 1, these are aggregated and sent to v along the edges of the BFS tree. If $\mathrm{cond}(S) < \Phi$ for the first time, the algorithm proceeds to the second phase. It continues the BFS for $-\log(\mathrm{vol}(S))/\log(1-\Phi)$ rounds and stops. If any vertex in the graph notices a neighbor that is not in S after these rounds, then $S \neq V$ and the algorithm rejects. Otherwise, we have obtained the value of $n = |S|$ that can be sent to all vertices, and we continue by executing Algorithm 1.

▶ **Lemma 15.** *Let $G = (V, E)$ be a graph and $\Phi \in [0,1]$. There is an algorithm that computes n if G has conductance at least Φ. Otherwise, it either computes n or rejects. The round complexity is $\mathcal{O}(\log m/\log(1-\Phi))$.*

Proof. It is easy to see that if the algorithm explores the whole graph, it computes n correctly, and else it rejects. Without loss of generality, let G have conductance Φ. Let S_i be the set S after i rounds and let $\bar{S}_i = V \setminus S_i$. We denote the last round of the first (second) phase by k (ℓ).

In the first phase, we have that $\mathrm{vol}(S_i) \geq (1+\Phi) \cdot \mathrm{vol}(S_{i-1})$ for every round i and by induction, $k \leq \log \mathrm{vol}(S_k)/\log(1+\Phi) \leq \log m/\log(1+\Phi)$. We also have that $\mathrm{vol}(S_k) \geq m/2 \geq \mathrm{vol}(\bar{S}_k)$ because G has conductance Φ. In the second phase, we have that $\mathrm{vol}(\bar{S}_i) \leq (1-\Phi) \cdot \mathrm{vol}(\bar{S}_{i-1})$ for every round i. By induction, $\ell - k \geq -\log m/\log(1-\Phi) \geq \log \mathrm{vol}(\bar{S}_k)^{-1}/\log(1-\Phi)$ implies that that $\mathrm{vol}(\bar{S}_\ell) = 0$. Therefore, the algorithm has explored the whole graph. Clearly, $\ell \in \mathcal{O}(\log m/\log(1-\Phi))$. ◀

To transform the algorithm into a tester in the CONGEST model, we start with each vertex being a maintainer initially. In every round every vertex chooses the vertex with the smallest id it has ever received a message from to be the maintainer and it forwards only this vertex' messages (the latter maintains the congestion bound). At the end of the algorithm, if

G has conductance Φ, then there is only one maintainer (the vertex with the smallest id) and the algorithm continues by executing Algorithm 1. Otherwise, there might be multiple vertices that are still maintainers. However, none of these vertices has explored the whole graph, so all of them send a broadcast message to reject.

4 Lower Bound

In this section, we prove a lower bound of $\Omega(\log(n+m))$ on the round complexity for testing the conductance of a graph in the LOCAL model regardless of how the final decision of the tester is derived from the single votes of the vertices.

For any $v \in V$, the k-disc of v, denoted by $\mathrm{disc}_k(G,v)$, is defined as the subgraph that is induced by the vertices that are at distance at most k to v without the edges between vertices at distance exactly k, and it is rooted at v. We refer to the isomorphism type of $\mathrm{disc}_k(G,v)$, that is, the set of all rooted graphs isomorphic to $\mathrm{disc}_k(G,v)$, by $\mathrm{disc}^*_k(G,v)$. Let $\mathrm{girth}(G)$ denote the length of the shortest cycle in G. We need the following two lemmas to obtain the distribution over graphs to prove the lower bound.

▶ **Lemma 16** ([20]; cf. [23, Section 16.8.3]). *For every $n' \in \mathbb{N}$ and every $d' \in \mathbb{N}$ there exists a d-regular graph G of size n such that G has conductance $\Phi(G) = 1/\sqrt{2d}$ and girth $2\log n/\log d$, and $n \geq n'$, $d \geq d'$.*

The second lemma states that we can sparsify an arbitrary cut $E(V_1,V_2)$ in a d-regular graph with girth $3k$ without changing $\mathrm{disc}^*_k(G,v)$ for any $v \in V$. In particular, it states that we can remove two edges in the cut and add them somewhere else, or the cut has size $\mathrm{poly}(d^k)$ only. It is obtained as a special case by observing that we can assume $L = 1$ and $\lambda = 0$ in [9, Lemma 8].

▶ **Lemma 17** ([9, Lemma 8]). *Let $G = (V,E)$ be a d-regular graph with $\mathrm{girth}(G) \geq 3k$ for $k \geq 2$ and let $V_1 \dot\cup V_2 = V$ be a partitioning of V. Then either there exists a graph $H = (V,F)$ such that (i) $\mathrm{girth}(H) \geq 3k$, (ii) $|F \cap (V_1 \times V_2)| \leq |E \cap (V_1 \times V_2)| - 2$, and (iii) $\mathrm{disc}^*_k(H,w) = \mathrm{disc}^*_k(G,w) \forall w \in V$ (that is, H is d-regular), or $E(V_1,V_2) \leq 6d^{3k}$.*

To prove the lower bound, we use an auxiliary model we call the ISO-LOCAL model. In this model, the input $I(\cdot)$ is empty but an additional oracle provides every vertex v with the ability to construct $\mathrm{disc}^*_r(G,v)$ in round r if it knows $\mathrm{disc}^*_{r-1}(G,u_i)$ of its neighbors $u_1,\dots,u_{d(v)}$. It should be noted that the ISO-LOCAL model is not a DCM due to the additional oracle.

▶ **Definition 18** (ISO-LOCAL model). *Let $\mathrm{DCM}(G,p_G,I)$ be a DCM instance such that $I(\cdot)$ maps the whole domain to the empty string. In addition to sending and receiving messages, in every round r every vertex v is provided access to a function $e_{r,v} : (\mathbb{N} \cup \{\star\})^r \times (\mathbb{N} \cup \{\star\})^r \to \{0,1\}$ during the local computation phase. The value of $e_{r,v}((i_1,\dots,i_{r'}),(j_1,\dots,j_{r''}))$ is 1 iff $p'_v(i_1,\dots,i_{r'}) = p'_v(j_1,\dots,j_{r''})$, where*

$$p'_v(i_1,\dots,i_r) := \begin{cases} v & \text{if } i_r = \star \\ p_{p'_v(\star,i_1,\dots,i_{r-1})}(i_r) & \text{otherwise.} \end{cases}$$

The instance $\mathrm{DCM}(G,p_G,I)$ equipped with such an oracle is called ISO-LOCAL.

In other words, $p'_v(\cdot)$ takes a path of length at most r that starts at v and that is defined by a sequence of port numbers as input. Then, it maps the path to its endpoint in V. Finally, $e_{r,v}(\cdot)$ tells whether two such paths end at the same vertex.

The ISO-LOCAL model is a graph where the nodes are not labeled by any strings. To argue the lower bound, we need to prove the existence of graphs that have same local neighborhoods such that one is a good expander and the other is far from having good conductance. Now it is possible that the algorithm can glean information about the different graphs based on the vertex labels even if the local neighborhoods are identical. Without ISO-LOCAL, we would need to argue that a randomized algorithm cannot deduce information from the vertex labels in the LOCAL model directly. This would be easy if for every good expander G we use, there is a bad expander H with exactly the same set of (labeled) k-discs. However, this is not the case as it would imply that G and H are isomorphic. The ISO-LOCAL model formalizes the intuition that, still, isomorphic k-discs should be sufficient to establish the lower bound even for randomized algorithms.

It is a basic observation that a distributed algorithm can only depend on information that has reached it until the moment it performs the computation in question.

▶ **Lemma 19** (folklore; cf. [19, Section 2]). *Let* $\mathrm{DNDA}(\mathcal{A}, O)$ *be a DNDA. After* r *rounds, the state of* $\mathcal{A}_v$ *may depend only on* $d(v)$, $I(v)$, *the state of* $\mathcal{A}_u$ *at time* $r - \mathrm{dist}(v, u)$ *for vertices* u *with* $\mathrm{dist}(v, u) < r$ *and the random coins of* $\mathcal{A}$.

4.1 Proof of the Lower Bound

Let $G = (V, E)$ be an expander graph obtained from applying Lemma 16 and let $k = \Theta(\log n)$. Observe that if a graph is d-regular and it has girth $3k$, then all its k-discs are pairwise isomorphic. In particular, all k-discs are full d-ary trees of depth k.

We will prove that a distributed algorithm $\mathrm{DNDA}(\mathcal{A}, O)$ with round complexity r in the ISO-LOCAL model decides based on the set of views $\mathrm{disc}^*_r(G, v)$ that the different instances of $\mathcal{A}$ have (see Lemma 20). Using Lemma 17, it will be easy to come up with a graph H that is a bad expander but whose k-discs are isomorphic to the ones of G. This implies a lower bound of $k = \Theta(\log n)$ for testing conductance in the ISO-LOCAL model (see Proposition 21). Finally, we prove that a lower bound on the round complexity of a tester in the ISO-LOCAL model implies the same bound in the LOCAL model. Actually, we prove the contrapositive: a tester in the LOCAL model implies a tester in the ISO-LOCAL model (see Proposition 22).

▶ **Lemma 20.** *Let* $\mathrm{DNDA}(\mathcal{A}, O)$ *be a deterministic DNDA in the ISO-LOCAL model. The output of* $\mathcal{A}_v$ *depends only on* $\mathrm{disc}^*_r(G, v)$ *and the port numbering* $(p_v)_{v \in V}$.

Proof. Instead of analyzing $\mathrm{DNDA}(\mathcal{A}, O)$, we analyze a canonical algorithm $\mathrm{DNDA}(\mathcal{B}, O)$ that simulates $\mathrm{DNDA}(\mathcal{A}, O)$ depending only on $\mathrm{disc}^*_r(G, v)$. Employing $\mathcal{B}$, we prove the following statement by induction: After the local computation phase of round r, the state of $\mathcal{A}_v$ depends only on $\mathrm{disc}^*_r(G, v)$.

The first local computation phase of $\mathcal{A}_v$ can only depend on the port numbering and $I(v)$ (the empty string). Therefore, $\mathcal{B}_v$ can simulate the execution of the first round of $\mathcal{A}_v$.

Let the current round be $r > 1$. Algorithm $\mathcal{B}_v$ maintains a rooted graph H_v that resembles $\mathrm{disc}^*_r(G, v)$. The adjacency lists of H_v are ordered according to $(p_v)_{v \in V}$. Let $H_v(r)$ be the value of H_v after the computation phase of round r. In the send phase, vertex v sends $H_v(r)$ to each of its neighbors. In the receive phase, vertex v receives graphs $H_{u_1}(r), \ldots, H_{u_{d(v)}}(r)$ from its neighbors $u_1, \ldots, u_{d(v)}$. In the subsequent computation phase of round $r + 1$, vertex v extends $H_v(r) = \mathrm{disc}^*_r(G, v)$ to $\mathrm{disc}^*_{r+1}(G, v) = H_v(r + 1)$ by querying $e_{r,v}$ on all pairs of vertices of $V(H_v(r)) \cup V(H_{u_1}(r)) \cup \ldots \cup V(H_{u_{d(v)}}(r))$ to identity vertices and patching the different views together.

Note that $H_v(r + 1)$ also provides the isomorphism type of $\mathrm{disc}^*_{r-\mathrm{dist}(v,u)}(G, u)$ for every vertex u at distance at most r from v. Since the adjacency lists of H_v are ordered according

to the port numbering, it is also possible to reconstruct $e_{r-\mathrm{dist}(v,u),u}(\cdot)$. By the induction hypothesis, $\mathcal{B}_v$ can now simulate round $r - \mathrm{dist}(v,u)$ of $\mathcal{A}_u$ for every such u. By Lemma 19, this is enough to simulate the local computation phase of round r of $\mathcal{A}_u$. ◀

We show that there is no tester for conductance in the ISO-LOCAL model.

▶ **Proposition 21.** *Let $G = (V, E)$ be d-regular graph on n vertices, and let $\Phi = 1/\sqrt{2d}$ be a constant. Any algorithm for testing if G has conductance at least Φ or is ϵ-far from having conductance at least $c\Phi^2$ (for constants c and ϵ) in the ISO-LOCAL model that succeeds with probability $2/3$ requires $\Omega(\log n)$ rounds of communication.*

Proof. Let $G = (V, E)$ be a d-regular graph provided by Lemma 16 and choose $k = \frac{1}{3}\log_d\left(\frac{c\Phi^2-\epsilon}{6}dn\right)$. Without loss of generality assume that n is even, and let $S \subset V$ be a set of size $n/2$. Apply Lemma 17 (with $V_1 = S$ and $V_2 = V \setminus S$) repeatedly to G until $|E(S, V \setminus S)| \leq 6d^{3k}$ holds. Let $H = (V, E')$ be the resulting graph. We have that $|E'(S, V \setminus S)| \leq (c\Phi^2 - \epsilon)dn$, and $\mathrm{vol}(S) = nd/2$. Therefore, H is ϵ-far from having conductance $c\Phi^2$. Let $\mathcal{D}_G$ ($\mathcal{D}_H$) be the uniform distribution over all ISO-LOCAL models $\mathrm{DCM}(G, p_G, I)$ ($\mathrm{DCM}(H, p_H, I)$) such that p_G (p_H) ranges over all possible mappings, that is, port numberings.

We use Yao's principle to prove the lower bound. Let $\mathrm{DNDA}(\mathcal{A}, O)$ be a tester for conductance that has round complexity smaller than k in the ISO-LOCAL model. Since G is d-regular and $\mathrm{girth}(G) \geq 3k$, $\mathrm{disc}^*_k(G, v)$ is a full d-ary tree of depth k for every $v \in V$. For any pair $u, v \in V$, we have that $\mathrm{disc}^*_k(G, u)$ is equal to $\mathrm{disc}^*_k(H, v)$ by Lemma 17. Since the port numberings of two vertices are independent of each other, $(p_v)_{v\in V}$ is a valid port numbering for $G \in \mathcal{D}_G$ iff it is valid for $H \in \mathcal{D}_H$. By Lemma 20, $\mathrm{DNDA}(\mathcal{A}, O)$ cannot distinguish between G and H. ◀

To complete the proof of the lower bound, we show that each vertex in the graph in the ISO-LOCAL model can choose an id randomly.

▶ **Proposition 22.** *Let $\mathrm{DNDA}(\mathcal{A}, O)$ be a randomized tester in the LOCAL model that succeeds with probability p. Then, there is a randomized tester $\mathrm{DNDA}(\mathcal{B}, O)$ in the ISO-LOCAL model that succeeds with probability at least $p - o(1)$, and has the same round complexity.*

Proof. We make a simple modification to $\mathcal{A}$ to obtain $\mathcal{B}$: In the first local computation phase, $\mathcal{B}_v$ draws a random number id_v uniformly from $\{1, \ldots, n^3\}$ and feeds it into $\mathcal{A}_v$ as $I(v)$. Then, $\mathcal{A}_v$ is executed as normal. For $u, v \in V$, the probability that id_u and id_v are equal is $1/n^3$. Applying a union bound, with probability $1 - o(1)$, it holds that $id_u \neq id_v$ for every $u, v \in V$. We then run algorithm $\mathcal{A}$ on this new instance and output the result. ◀

5 Open Problems

In the case of one-sided distributed testers, it is natural to define the decision rule $O(\cdot)$ of a distributed tester such that all vertices have to accept or at least one vertex has to reject. This is because in the case of rejection, the tester is required to observe a witness. However, for two-sided testers no such requirement exists. Requiring that all vertices either accept or reject simultaneously seems to be quite strong. For example, if all should vertices accept or at least one vertex should reject, one may overcome the lack of communication between connected components and test, e. g., whether a graph has more than two connected components with two-sided error. On the other hand, it might not always be possible to obtain a lower bound that is independent of the decision rule as in Theorem 2. To this end, it would be interesting to compare the power of different rules.

References

1 Zvika Brakerski and Boaz Patt-Shamir. Distributed discovery of large near-cliques. *Distributed Computing*, 24(2), 2011. doi:10.1007/s00446-011-0132-x.

2 Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. Fast distributed algorithms for testing graph properties. In *Proceedings of the 30th International Symposium on Distributed Computing (DISC)*, 2016.

3 Fan R. K. Chung. Diameters and eigenvalues. *Journal of the American Mathematical Society*, 2(2), 1989.

4 Fan R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

5 Artur Czumaj, Pan Peng, and Christian Sohler. Testing cluster structure of graphs. In *Proccedings of the 47th ACM Symposium on Theory of Computing (STOC)*, 2015.

6 Artur Czumaj and Christian Sohler. Testing expansion in bounded-degree graphs. In *Proccedings of the 48th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2007.

7 Atish Das Sarma, Danupon Nanongkai, Gopal Pandurangan, and Prasad Tetali. Distributed random walks. *Journal of the ACM (JACM)*, 60(1), 2013.

8 Guy Even, Orr Fischer, Pierre Fraigniaud, Tzlil Gonen, Reut Levi, Moti Medina, Pedro Montealegre, Dennis Olivetti, Rotem Oshman, Ivan Rapaport, and Ioan Todinca. Three Notes on Distributed Property Testing. In *Proceedings of the 31st International Symposium on Distributed Computing (DISC)*, 2017.

9 Hendrik Fichtenberger, Pan Peng, and Christian Sohler. On constant-size graphs that preserve the local structure of high-girth graphs. In *Proccedings of the 19th International Workshop on Randomization and Computation (RANDOM)*, 2015.

10 Hendrik Fichtenberger and Yadu Vasudev. A Two-Sided Error Distributed Property Tester For Conductance. *arXiv:1705.08174*, 2018. URL: http://arxiv.org/abs/1705.08174.

11 Pierre Fraigniaud, Ivan Rapaport, Ville Salo, and Ioan Todinca. Distributed testing of excluded subgraphs. In *Proceedings of the 30th International Symposium on Distributed Computing (DISC)*, 2016.

12 Oded Goldreich. Introduction to testing graph properties. In *Property Testing*. Springer, 2010.

13 Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017.

14 Oded Goldreich, Shari Goldwasser, and Dana Ron. Property Testing and Its Connection to Learning and Approximation. *Journal of the ACM (JACM)*, 45(4), 1998. doi:10.1145/285055.285060.

15 Oded Goldreich and Dana Ron. On Testing Expansion in Bounded-Degree Graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 2000. URL: https://eccc.weizmann.ac.il/report/2000/020/.

16 Satyen Kale and C. Seshadhri. An expansion tester for bounded degree graphs. *SIAM Journal on Computing (SICOMP)*, 40(3), 2011. doi:10.1137/100802980.

17 Angsheng Li, Yicheng Pan, and Pan Peng. Testing Conductance in General Graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 18(101), 2011. URL: http://eccc.hpi-web.de/report/2011/101/.

18 Angsheng Li and Pan Peng. Testing Small Set Expansion in General Graphs. In *Proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2015.

19 Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing (SICOMP)*, 21(1), 1992. doi:10.1137/0221015.

20 Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3), 1988. doi:10.1007/BF02126799.

21 Anisur Rahaman Molla and Gopal Pandurangan. Distributed Computation of Mixing Time. In *Proceedings of the 18th International Conference on Distributed Computing and Networking (ICDCN)*, 2017.

22 Asaf Nachmias and Asaf Shapira. Testing the expansion of a graph. *Information and Computation*, 208(4), 2010. doi:10.1016/j.ic.2009.09.002.

23 Uwe Naumann and Olaf Schenk, editors. *Combinatorial Scientific Computing*. CRC Press, 2012.

24 Alistair Sinclair. *Algorithms for Random Generation and Counting: A Markov Chain Approach*. Birkhauser Verlag, 1993.

Graph Similarity and Approximate Isomorphism

Martin Grohe
RWTH Aachen University, Aachen, Germany
grohe@informatik.rwth-aachen.de
https://orcid.org/0000-0002-0292-9142

Gaurav Rattan
RWTH Aachen University, Aachen, Germany
rattan@informatik.rwth-aachen.de
https://orcid.org/0000-0002-5095-860X

Gerhard J. Woeginger
RWTH Aachen University, Aachen, Germany
woeginger@informatik.rwth-aachen.de
https://orcid.org/0000-0001-8816-2693

Abstract

The *graph similarity* problem, also known as approximate graph isomorphism or graph matching problem, has been extensively studied in the machine learning community, but has not received much attention in the algorithms community: Given two graphs G, H of the same order n with adjacency matrices A_G, A_H, a well-studied measure of similarity is the *Frobenius distance*

$$\operatorname{dist}(G,H) := \min_{\pi} \|A_G^{\pi} - A_H\|_F,$$

where π ranges over all permutations of the vertex set of G, where A_G^{π} denotes the matrix obtained from A_G by permuting rows and columns according to π, and where $\|M\|_F$ is the Frobenius norm of a matrix M. The (weighted) graph similarity problem, denoted by GSim (WSim), is the problem of computing this distance for two graphs of same order. This problem is closely related to the notoriously hard *quadratic assignment problem* (QAP), which is known to be NP-hard even for severely restricted cases.

It is known that GSim (WSim) is NP-hard; we strengthen this hardness result by showing that the problem remains NP-hard even for the class of trees. Identifying the boundary of tractability for WSim is best done in the framework of linear algebra. We show that WSim is NP-hard as long as one of the matrices has unbounded rank or negative eigenvalues: hence, the realm of tractability is restricted to positive semi-definite matrices of bounded rank. Our main result is a polynomial time algorithm for the special case where the associated (weighted) adjacency graph for *one of the matrices* has a bounded number of twin equivalence classes. The key parameter underlying our algorithm is the clustering number of a graph; this parameter arises in context of the spectral graph drawing machinery.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Graph Similarity, Quadratic Assignment Problem, Approximate Graph Isomorphism

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.20

Related Version A full version of the paper is available at `https://arxiv.org/abs/1802.08509`.

1 Introduction

Graph isomorphism has been a central open problem in algorithmics for the last 50 years. The question of whether graph isomorphism is in polynomial time is still wide open, but at least we know that it is in quasi-polynomial time [4]. On the practical side, the problem is largely viewed as solved; there are excellent tools [9, 15, 21, 22] that efficiently decide isomorphism on all but very contrived graphs [25]. However, for many applications, notably in machine learning, we only need to know whether two graphs are "approximately isomorphic", or more generally, how "similar" they are. The resulting *graph similarity* problem has been extensively studied in the machine learning literature under the name *graph matching* (e.g. [1, 10, 14, 29, 30]), and also in the context of the schema matching problem in database systems (e.g. [23]). Given the practical significance of the problem, surprisingly few theoretical results are known. Before we discuss these known and our new results, let us state the problem formally.

Graph Similarity. It is not obvious how to define the distance between two graphs, but the distance measure that we study here seems to be the most straightforward one, and it certainly is the one that has been studied most. For two n-vertex graphs G and H with adjacency matrices A_G and A_H, we define the *Frobenius distance* between G and H to be

$$\operatorname{dist}(G,H) := \min_{\pi} \|A_G^{\pi} - A_H\|_F. \tag{1}$$

Here π ranges over all permutations of the vertex set of G, A_G^{π} denotes the matrix obtained from A_G by permuting rows and columns according to π, and the norm $\|M\|_F := \sqrt{\sum_{i,j} M_{ij}^2}$ is the *Frobenius norm* of a matrix $M = (M_{ij})$. Note that $\operatorname{dist}(G,H)^2$ counts the number of edge mismatches in an optimal alignment of the two graphs. The *graph similarity problem*, denoted by GSim, is the problem of computing $\operatorname{dist}(G,H)$ for graphs G, H of the same order, or, depending on the context, the decision version of this problem (decide whether $\operatorname{dist}(G,H) \le d$ for a given d). We can easily extend the definitions to weighted graphs and denote the *weighted graph similarity problem* by WSim. In practice, this is often the more relevant problem. Instead of the adjacency matrices of graphs, we may also use the Laplacian matrices of the graphs to define distances. Recall that the *Laplacian matrix* of a graph G is the matrix $L_G := D_G - A_G$, where D_G is the diagonal matrix in which the entry $(D_G)_{ii}$ is the degree of the ith vertex, or in the weighted case, the sum of the weights of the incident edges. Let $\operatorname{dist}_L(G,H) := \min_{\pi} \|L_G^{\pi} - L_H\|_F$ be the corresponding distance measure. Intuitively, in the definition of $\operatorname{dist}_L(G,H)$ we prefer permutations that map vertices of similar degrees onto one another. Technically, $\operatorname{dist}_L(G,H)$ is interesting, because the Laplacian matrices are positive semidefinite (if the weights are nonnegative). Both the (weighted) similarity problem and its version for the Laplacian matrices are special cases of the problem MSim of computing $\min_P \|A - PBP^{-1}\|_F$ for given symmetric matrices $A, B \in \mathbb{R}^{n\times n}$. In the Laplacian case, these matrices are positive semidefinite.[1]

The QAP. The graph similarity problem is closely related to *quadratic assignment problem* (QAP) [6]: given two $(n \times n)$-matrices A, B, the goal is to find a permutation $\pi \in S_n$ that minimizes $\sum_{i,j} A_{ij} B_{\pi(i)\pi(j)}$. The usual interpretation is that we have n *facilities* that we

[1] Note that the notion of similarity that we use here has nothing to do with the standard notion of "matrix similarity" from linear algebra.

want to assign to n *locations*. The entry A_{ij} is the *flow* from the ith to the jth facility, and the entry B_{ij} is the *distance* from the ith to the jth location. The goal is to find an assignment of facilities to locations that minimizes the total cost, where the cost for each pair of facilities is defined as the flow times the distance between their locations. The QAP has a large number of real-world applications, as for instance hospital planning [11], typewriter keyboard design [27], ranking of archeological data [18], and scheduling parallel production lines [13]. On the theoretical side, the QAP contains well-known optimization problems as special cases, as for instance the Travelling Salesman Problem, the feedback arc set problem, the maximum clique problem, and all kinds of problems centered around graph partitioning, graph embedding, and graph packing.

In the maximization version MAX-QAP of QAP, the objective is to maximize the quantity $\sum_{i,j} A_{ij}B_{\pi(i)\pi(j)}$ (see [19, 24]). Both QAP and MAX-QAP are notoriously hard combinatorial optimization problems, in terms of practical solvability [28] as well as in terms of theoretical hardness results even for very restricted special cases [5, 8, 7]. It is easy to see that MSIM is equivalent to MAX-QAP, because in reductions between QAP and MSIM the sign of one of the two matrices is flipped. Most of the known results for GSIM and its variants are derived from results for (MAX)QAP.

Previous Work. It seems to be folklore knowledge that GSIM is NP-complete. For example, this can be seen by a reduction from the Hamiltonian path problem: take G to be the n-vertex input graph and H a path of length n; then $\mathrm{dist}(G,H) \leq \sqrt{|E(G)|-n}$ if and only if G has a Hamiltonian path. By the same argument, we can actually reduce the subgraph isomorphism problem to GSIM. Arvind, Köbler, Kuhnert, and Vasudev [3] study several versions of what they call *approximate graph isomorphism*; their problem MIN-PGI is the same as our GSIM. They prove various hardness of approximation results. Based on an earlier QAP-approximation algorithm due to Arora, Frieze, and Kaplan [2], they also obtain a quasi-polynomial time approximation algorithm for the related problem MAX-PGI. Further hardness results were obtained by Makarychev, Manokaran, and Sviridenko [19] and O'Donnell, Wright, Wu, and Zhou [26], who prove an average case hardness result for a variant of GSIM problem that they call *robust graph isomorphism*. Keldenich [16] studied the similarity problem for a wide range matrix norms (instead of the Frobenius norm) and proved hardness for essentially all of them.

Our (hardness) results. So where does all this leave us? Well, GSIM is obviously an extremely hard optimization problem. We start our investigations by adding to the body of known hardness results: we prove that GSIM remains NP-hard even if both input graphs are trees (Theorem 8). Note that in strong contrast to this, the subgraph isomorphism problem becomes easy if both input graphs are trees [20]. The reduction from Hamiltonian path sketched above shows that GSIM is also hard if one input graph is a path. We prove that GSIM is tractable in the very restricted case that one of the input graphs is a path and the other one is a tree (Theorem 9).

As WSIM and MSIM are essentially linear algebraic problems, it makes sense to look for algebraic tractability criteria. We explore bounded rank (of the adjacency matrices) as a tractability criteria for WSIM and MSIM. Indeed, the NP-hardness reductions for GSIM involve graphs which have adjacency matrices of high rank (e.g. paths, cycles). We show that the problem GSIM (and WSIM) remains NP-hard as long as one of the matrices has unbounded rank or negative eigenvalues. (Theorems 10, 11 and 12). Consequently, the realm of tractability for WSIM (and MSIM) is restricted to the class of positive semi-definite (PSD) matrices of bounded rank.

Block Partition Structure. We feel that for a problem as hard as QAP or MSIM, identifying any somewhat natural tractable special case is worthwhile. Since the spectral structure of PSD matrices of bounded rank is quite limited, we consider combinatorial restrictions: in particular, restricting the block structure of these matrices is a natural line of investigation.

Given a weighted graph G, we call two vertices *twins* if they have identical (weighted) adjacency to every vertex of G. The *twin-equivalence partition* of $V(G)$, corresponding to this equivalence relation, induces a block structure on the adjacency matrix A_G. Indeed, if $S_1 \dot\cup \ldots \dot\cup S_p = V(G)$ are the twin-equivalence classes, the submatrix $A_G[S_i, S_j]$ is a constant matrix. Hence, the rows and columns of the matrix A_G can be simultaneously rearranged to yield a $p \times p$ block matrix. The number of twin-equivalence classes will be an important parameter of our interest: we denote this parameter by $\tau(G)$.

Our (algorithmic) results. Our main result is a polynomial time algorithm for MSIM if both input matrices are positive semidefinite and have bounded-rank, and where *one of the input matrices* has a bounded number of twin-equivalence classes. Formally, we prove the following theorem. Here, the $\widetilde{\mathcal{O}}$ notation hides factors polynomial in the input representation.

▶ **Theorem 1.** *The problem* MSIM *can be solved in* $\widetilde{\mathcal{O}}(n^{kp^2})$ *time where*
- **(i)** *the input matrices are* $n \times n$ *PSD matrices of rank at most* k*, and*
- **(ii)** *one of the input matrices has at most* p *twin-equivalence classes.*

For the proof of Theorem 1, we can re-write the (squared) objective function as $\|AP - PB\|_F^2$, where P ranges over all permutation matrices. This is a convex function, and it would be feasible to minimize it over a convex domain. The real difficulty of the problem lies in the fact that we are optimizing over the complicated discrete space of permutation matrices. Our approach relies on a linearization of the solution space, and the key insight (Lemma 19) is that the optimal solution is essentially determined by polynomially many hyperplanes. To prove this, we exploit the convexity of the objective function in a peculiar way.

2 Preliminaries

We denote the set $\{1, \ldots, n\}$ by $[n]$. Unless specified otherwise, we will always assume that the vertex set of an n-vertex graph G is $[n]$. We denote the degree of a vertex v by $d_G(v)$.

Twins. Given a $n \times n$ symmetric matrix A with real entries, let G_A denote the associated weighted adjacency graph. Two vertices are called *twins* if they have identical (weighted) adjacency to every vertex in the graph. Hence, two vertices labeled $i, j \in [n]$ are twins if and only if $A_{il} = A_{jl}$ for all $l \in [n]$. This is an equivalence relation; call the resulting partition of the vertex set as the *twin-equivalence partition*. The number of twin-equivalence classes will be an important parameter of our interest: we denote this parameter by $\tau(G)$. In these definitions, we use the matrix A and its adjacency graph A_G interchangeably. This allows us to define $\tau(A)$ for a matrix A to be $\tau(G_A)$, for the associated weighted adjacency graph G_A. The connection with block structure of the matrix is straighforward: observe that we can simultaneously rearrange the rows and columns of A to obtain a $\tau(A) \times \tau(A)$ block matrix W. If $S_1 \dot\cup \ldots \dot\cup S_p = [n]$ be the twin-equivalence partition, the block W_{lm} (where $l, m \in [\tau(A)]$) is the adjacency matrix for the induced subgraph $G_A[S_l, S_m]$. Moreover, the definition of twin-equivalence partition implies that this subgraph is a weighted complete bipartite graph.

Matrices. Given an $m \times n$ matrix M, the i^{th} row (column) of M is denoted by M^i (M_i). The multiset $\{M^1, \ldots, M^m\}$ is denoted by rows(M). Given $S \subseteq [m]$, the sum $\sum_{i \in S} M^i$ is denoted by M^S. We denote the $n \times n$ identity matrix by I_n. A real symmetric $n \times n$ matrix M is called *positive semi-definite* (PSD), denoted by $M \succeq 0$, if the scalar $z^T M z$ is non-negative for every $z \in \mathbb{R}^n$. The following conditions are well-known to be equivalent.

- **(i)** $M \succeq 0$
- **(ii)** Every eigenvalue of M is non-negative.
- **(iii)** $M = W^T W$ for some $n \times n$ matrix W. In other words, there exist n vectors $w_1, \ldots, w_n \in \mathbb{R}^n$ such that $M_{ij} = w_i^T w_j$.

Given two vectors $x, y \in \mathbb{R}^n$, their dot product $\langle x, y \rangle$ is defined to be $x^T y$. Given $M \succeq 0$, the inner product of x, y w.r.t. M, denoted by $\langle x, y \rangle_M$, is defined to be $x^T M y$. The usual dot product corresponds to the case $M = I$, the identity matrix. Every $n \times n$ symmetric matrix M has a spectral decomposition $M = U \Sigma U^T$, where the rows of U form an eigenbasis. If M has rank k, we can truncate the zero eigenvalues in Σ to obtain a truncated spectral decomposition. Now, Σ is a $k \times k$ diagonal matrix with the eigenvalues $\lambda_1, \ldots, \lambda_k \in \mathbb{R}$ on the diagonal. The matrix U is a $n \times k$ matrix with the corresponding eigenvectors $v_1, \ldots, v_k$ as the columns $U_1, \ldots, U_k$. We will always work with truncated spectral decompositions henceforth.

Frobenius Norm. The trace of a matrix M, denoted by $\mathrm{Tr}(M)$, is defined to be $\sum_{i \in [n]} M_{ii}$. The *trace inner product* of two matrices A and B, denoted by $\mathrm{Tr}(A, B)$, is the scalar $\mathrm{Tr}(A^T B)$. The *Frobenius norm* $\|M\|_F$ of a matrix M is defined to be $\sum_{i,j \in [n]} {M_{ij}}^2$. It is easy to check that $\|M\|_F^2 = \mathrm{Tr}(M, M)$. Given two n-vertex graphs G and H and a permutation $\pi \in S_n$, a *π-mismatch* between G and H is a pair $\{i, j\}$ such that $\{i, j\} \in E(G)$ and $\{i^\pi, j^\pi\} \notin E(H)$ (or vice-versa). In other words, $\pi : V(G) \to V(H)$ does not preserve adjacency for the pair $\{i, j\}$. The following claim will be useful as a combinatorial interpretation of the Frobenius norm. Let Δ denote the number of π-mismatches between G and H.

▶ **Claim 2.** $\|A_G^\pi - A_H\|_F^2 = 2\Delta$.

Proof. The only non-zero terms in the expansion of summation $\|A_G^\pi - A_H\|_F^2$ correspond to π-mismatches. Since every mismatch $\{i, j\}$ contributes 1 and is counted twice in the summation, the claim follows. ◀

Clustering Number. Spectral Graph Drawing is a well-established technique for visualizing graphs via their spectral properties (see e.g. [17]). We introduce the details necessary for our results. Let A be a $n \times n$ matrix of rank k. Let G be the corresponding adjacency graph, with the vertex set $[n]$. Given a spectral decomposition $A = U \Lambda U^T$, Σ is a $k \times k$ matrix and U is a $n \times k$ matrix. Since the spectral decomposition of a matrix is not unique, the following claim will be useful.

▶ **Claim 3.** *Given two spectral decompositions $A = U \Lambda U^T$ and $A = U' \Lambda {U'}^T$, the number of distinct elements in the multi-set* rows(U) *is equal to the number of distinct elements in the multi-set* rows(U').

Therefore, the number of *distinct* elements in the multi-set rows(U) is invariant of our choice of spectral decomposition $A = U \Lambda U^T$. This allows us to define the *clustering number* of a graph G, denoted by $\mathrm{cn}(G)$, as the number of *distinct* elements in the multi-set rows(U), for

some spectral decomposition $A = U\Lambda U^T$. The clustering number of a matrix A, denoted by $\mathrm{cn}(A)$, is defined to be the clustering number of the corresponding adjacency graph.

Let $A = U\Lambda U^T$ be a PSD matrix. The following theorem relates the clustering number $\mathrm{cn}(A)$ to the number of twin-equivalence partitions $\tau(A)$.

▶ **Theorem 4.** *Let A be a PSD matrix. The number of twin-equivalence classes $\tau(A)$ is equal to p if and only if A has p distinct elements in the set* $\mathrm{rows}(U)$ *for a spectral decomposition $A = U\Lambda U^T$.*

Hyperplanes and Convex Functions. A *hyperplane* H in the Euclidean space $\mathbb{R}^k$ is a $(k-1)$-dimensional affine subspace. The usual representation of a hyperplane is a linear equation $\langle c, x\rangle = \alpha$ for some $c \in \mathbb{R}^k, \alpha \in \mathbb{R}$. The convex sets $\{x \mid \langle c, x\rangle > \alpha\}$ and $\{x \mid \langle c, x\rangle < \alpha\}$ are called the open *half-spaces* corresponding to H, denoted by H^+, H^- respectively.

Two sets (S, T) are *weakly linearly separated* if there exists a hyperplane H such that $S \subseteq H^+ \cup H$ and $T \subseteq H^- \cup H$. In this case, we call them weakly linearly separated along H. A family of sets $S_1, \ldots, S_p$ is *weakly linearly separated* if for every $l, m \in [p]$, the sets S_l, S_m are weakly linearly separated. Let Π be a partition of a set S into p sets $S_1, \ldots, S_p$. The partition Π is said to be *mutually linearly separated* if the family of sets $S_1, \ldots, S_p$ is weakly linearly separated.

Recall that a subset $S \subseteq \mathbb{R}^k$ is called *convex* if for every $x, y \in S$, $\alpha x + (1-\alpha)y \in S$, $\alpha \in [0, 1]$. A function $f : \mathbb{R}^k \to \mathbb{R}$ is called *convex* on a convex set S if for every $x, y \in S$, $f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y)$. The following theorem about linearization of convex differentiable functions is well-known and is stated without proof. The gradient of a function $f : \mathbb{R}^k \to \mathbb{R}$, denoted by ∇f, is the vector-valued function $[\frac{\partial f}{\partial x_1} \ldots \frac{\partial f}{\partial x_k}]$. Given $X^* \in \mathbb{R}^k$, let μ^* denote the vector $\nabla f(X^*)$.

▶ **Theorem 5** (Convex function linearization)**.** *Let $f : \mathbb{R}^k \to \mathbb{R}$ be a convex function. For all $X \in \mathbb{R}^k$, $f(X) - f(X^*) \geq \langle \mu^*, X - X^*\rangle$.*

Finally, we state an important fact about the convexity of quadratic functions. Given a PSD matrix $M \in \mathbb{R}^{k\times k}$, the quadratic function $Q_M : \mathbb{R}^k \to \mathbb{R}$ is defined as $Q_M(x) = \langle x, x\rangle_M$.

▶ **Lemma 6** (Convexity of PSD)**.** *Q_M is convex on $\mathbb{R}^k$.*

3 Hardness Results

In this section, we show several new hardness results for problems GSim, WSim and MSim. As we will observe, these problems turn out to be algorithmically intractable, even for severely restricted cases. We begin by recalling the following observation.

▶ **Theorem 7** (Folklore)**.** GSim *is NP-hard for the class of simple undirected graphs.*

In fact, the problem turns out to be NP-hard even for very restricted graph classes. The following theorem is the main hardness result of this section.

▶ **Theorem 8.** GSim *is NP-hard for the class of trees.*

Proof. The proof is by a reduction from the following NP-hard variant of the Three-Partition problem [12], which is defined as follows. The input consists of integers A and $a_1, \ldots, a_{3m}$ in unary representation, with $\sum_{i=1}^{3m} a_i = mA$ and with $A/4 < a_i < A/2$ for $1 \leq i \leq 3m$. The question is to decide whether $a_1, \ldots, a_{3m}$ can be partitioned into m triples so that the elements in each triple sum up to precisely A.

We first show that the restriction of GSim to forests is NP-hard. Given an instance of Three-Partition, we compute an instance of GSim on the following two forests F_1 and F_2. Forest F_1 is the disjoint union of $3m$ paths with $a_1, \ldots, a_{3m}$ vertices, respectively. Forest F_2 is the disjoint union of m paths that each consists of A vertices. We claim that the Three-Partition instance has answer YES, if and only if there exists a permutation π such that there are at most $2m$ mismatches. If the desired partition exists, then for each triple we we can pack the three corresponding paths in F_1 into one of the paths in F_2 with two mismatches per triple. Conversely, if there exists a permutation π with at most $2m$ mismatches, then these $2m$ mismatches cut the paths in F_2 into $3m$ subpaths (we consider isolated vertices as paths of length 0). As each of these $3m$ subpaths must be matched with a path in F_1, we easily deduce from this a solution for the Three-Partition instance.

To show that GSim is NP-hard for the class of trees, we modify the above forests F_1 and F_2 into trees T_1 and T_2. Formally, we add a new vertex v_1 to $V(F_1)$ and then connect one end-point of every path in F_1 to v_1 by an edge; note that the degree of vertex v_1 in the resulting tree is $3m$. Analogously, we add a new vertex v_2 to $V(F_2)$, connect it to all paths, and thus produce a tree in which vertex v_2 has degree m. For technical reasons, we furthermore attach $8m$ newly created leaves to every single vertex in $V(F_1)$ and $V(F_2)$. k The resulting trees are denoted T_1 and T_2, respectively.

We claim that the considered Three-Partition instance has answer YES, if and only if there exists $\pi : V(T_1) \to V(T_2)$ with at most $4m$ mismatches. If the desired partition exists, the natural bijection maps every original forest edge in T_1 to an original forest edge in T_2, except for some $2m$ out of the $3m$ edges that are incident to v_1 in T_1; this yields a total number of $2m + 2m = 4m$ mismatches. Conversely, suppose that there exists a permutation π with at most $4m$ mismatches. Then π must map v_1 in T_1 to v_2 in T_2, since otherwise we pay a penalty of more than $4m$ mismatches alone for the edges incident to the vertex mapped into v_2. As the number of mismatches for edges incident to v_1 and v_2 amounts to $2m$, there remain at most $2m$ further mismatches for the remaining edges. Similarly as in our above argument for the forests, these at most $2m$ mismatches yield a solution for the Three-Partition instance. ◀

On the other hand, if we restrict one of the input instances to be a path, the problem can be solved in polynomial time. The following theorem provides a positive example of tractability of GSim.

▶ **Theorem 9.** *An input instance (G, H) of* GSim*, where G is a path and H is a tree, can be solved in polynomial time.*

The above results exhibit the hardness of GSim, and consequently, the hardness of the more general problems WSim and MSim. Since the graphs (for instance cycles and paths) involved in the hardness reductions have adjacency matrices of high rank, it is natural to ask whether MSim would become tractable for matrices of low rank. Our following theorem shows that MSim is NP-hard even for matrices of rank at most 2. The underlying reason for hardness is the well-known problem QAP, which shares the optimization domain S_n.

▶ **Theorem 10.** MSim *is NP-hard for symmetric matrices of rank at most* 2*.*

The key to the above reduction is the fact that one of the matrices has non-negative eigenvalues while the other matrix has non-positive eigenvalues. We show that the MSim is NP-hard even for positive semi-definite matrices. The main idea is to reformulate the hardness reduction in Theorem 7 in terms of Laplacian matrices.

▶ **Theorem 11.** MSim *is NP-hard for positive semi-definite matrices.*

In fact, we show that the problem remains NP-hard, even if one of the matrices is of rank 1. The proof follows by modifying the matrices in the proof of Theorem 10 so that they are positive semi-definite.

▶ **Theorem 12.** MSim *is NP-hard for positive semi-definite matrices, even if one of the matrices has rank* 1*.*

Therefore, the realm of tractability for MSim is restricted to positive definite matrices of bounded rank.

4 The QVP Problem

We proceed towards the proof of Theorem 1, our main algorithmic result about MSim. In order to prove this theorem, we need to define an intermediate problem, called the Quadratic-Vector-Partition (QVP). In this section, we study several aspects of this problem. First, we state this problem, and show an efficient reduction from MSim to QVP (Sections 4.1 and 4.2). The definition of the problem QVP is slightly technical; the ensuing reduction, from MSim to QVP, will justify the introduction of this intermediate problem. In Sections 4.3 and 4.4, we will establish strong conditions on the optimal solutions for a QVP instance. Later on, in Section 5, these conditions will allow us to design efficient algorithms for QVP, which will finish the proof of Theorem 1.

4.1 QVP, definition

Let p and k be fixed positive integers. The *input instance* to QVP is a tuple (W, K, Λ, Δ), where

- W is a set of n vectors $\{w_1, \ldots, w_n\} \subseteq \mathbb{R}^k$,
- K is a $p \times p$ PSD matrix,
- Λ is a $k \times k$ *diagonal* matrix with non-negative entries, and,
- Δ is (the unary encoding of) a p-tuple $(n_1, \ldots, n_p)$ such that $n_1 + \cdots + n_p = n$.

Some additional notation is required, before we proceed further. An ordered partition $T_1 \dot\cup \cdots \dot\cup T_p$ of $[n]$ is said to have *type* Δ if the cardinalities $|T_l| = n_l$, for all $l \in [p]$. Let $\mathcal{P}_\Delta$ denote the set of all (ordered) partitions of $[n]$ of type Δ. Let T be a subset of $[n]$. Denote the subset of W indexed by the set T as $W[T] = \{w_j \mid j \in T\}$. The centroid of the subset $W[T]$ is denoted by $\widehat{w_T}$. In other words, $\widehat{w_T} = \frac{1}{|T|} \sum_{i \in T} w_i$.

We continue with the definition of QVP. Given a partition $P = (T_1, \ldots, T_p) \in \mathcal{P}_\Delta$, the QVP objective function $F(P)$ is defined as

$$F(P) = \sum_{l,m \in [p]} K_{lm} \langle \widehat{w}_{T_l}, \widehat{w}_{T_m} \rangle_\Lambda .$$

The optimization problem QVP is to compute a partition $P^* \in \mathcal{P}_\Delta$ which is a maximizer of the objective function $F(P)$ over the domain $\mathcal{P}_\Delta$.

4.2 MSim reduces to QVP

Let k and p be fixed positive integers. Let (A, B) be an MSIM instance, as defined in Theorem 1: the PSD matrices A and B are of rank at most k, and moreover, $\tau(B) = p$. The following lemma describes a reduction from MSIM to QVP. Here, the $\widetilde{\mathcal{O}}$ notation hides factors polynomial in the size of the input representation.

▶ **Lemma 13.** *There exists an $\widetilde{\mathcal{O}}(n^3)$ running time algorithm which can transform the* MSIM *instance (A, B) into a* QVP*-instance (W, K, Λ, Δ), with the following property. Given an optimal solution for this* QVP*-instance, we can compute an optimal solution for the* MSIM *instance, in $\mathcal{O}(n)$ running time.*

Proof. Fix two spectral decompositions $A = U\Lambda U^T$ and $B = V\Gamma V^T$ of A and B respectively. Since $\tau(B) = p$, the multiset rows(V) has exactly p distinct vectors (by Theorem 4). Let these p distinct vectors be denoted by $\{\tilde{V}^1, \ldots, \tilde{V}^p\}$. Let $n_1, \ldots, n_p$ be the multiplicity of the elements $\tilde{V}^1, \ldots, \tilde{V}^p$ in the multiset rows(V). Clearly, $n_1 + \cdots + n_p = n$. Let $\tilde{P} = S_1 \cup \cdots \cup S_p$ be the partition of the set $[n]$ such that $S_l = \{i \,|\, V^i = \tilde{V}^l\}$, for $l \in [p]$. In other words, the partition $\tilde{P}$ encodes the equivalence relation $V^i = V^j$, where $i, j \in [n]$.

Let us describe the polynomial time transformation of the MSIM instance (A, B) into the QVP instance (W, K, Λ, Δ). Define W as the multiset rows(U). In other words, we can denote $W = \{w_1, \ldots, w_n\}$ where $w_i = U^i$. Define K to be the $p \times p$ matrix defined as $K_{lm} = |S_l| \cdot |S_m| \cdot \langle \tilde{V}^l, \tilde{V}^m \rangle_\Gamma$, for $l, m \in [p]$. Since we can write $K_{lm} = \langle |S_l| \cdot \tilde{V}^l, |S_m| \cdot \tilde{V}^m \rangle_\Gamma$, we can show that K is positive semi-definite. We set Λ to be the $k \times k$ diagonal matrix in the spectral decomposition $A = U\Lambda U^T$. Finally, we set Δ to be $(n_1, \ldots, n_p)$: these numbers were defined in the previous paragraph. The computation of this QVP instance can be performed in $\widetilde{\mathcal{O}}(n^3)$ time, which is the time taken to compute the spectral decompositions for A and B.

It remains to show that an optimal solution for this QVP instance yields an optimal solution for the MSIM instance in $\mathcal{O}(n)$ time. Observe that $\|A^\pi - B\|_F^2 = \mathrm{Tr}(A^\pi - B, A^\pi - B) = \mathrm{Tr}(A^\pi, A^\pi) + \mathrm{Tr}(B, B) - 2\,\mathrm{Tr}(A^\pi, B)$. Since $\mathrm{Tr}(A^\pi, A^\pi) = \|A^\pi\|_F^2 = \|A\|_F^2 = \mathrm{Tr}(A, A)$, we have $\|A^\pi - B\|_F^2 = \mathrm{Tr}(A, A) + \mathrm{Tr}(B, B) - 2\,\mathrm{Tr}(A^\pi, B)$. This derivation implies that we can equivalently maximize $\mathrm{Tr}(A^\pi, B)$ over $\pi \in S_n$. Observe that $\mathrm{Tr}(A^\pi, B)$ can be rewritten as

$$\begin{aligned}
\mathrm{Tr}(A^\pi, B) &= \sum_{i,j \in [n]} a_{i^\pi j^\pi}\, b_{ij} \\
&= \sum_{i,j \in [n]} \langle U^{i^\pi}, U^{j^\pi} \rangle_\Lambda \cdot \langle V^i, V^j \rangle_\Gamma \\
&= \sum_{l,m \in [p]} \left(\sum_{i \in S_l,\, j \in S_m} \langle U^{i^\pi}, U^{j^\pi} \rangle_\Lambda \cdot \langle V^i, V^j \rangle_\Gamma \right)
\end{aligned}$$

which can be further re-written as

$$\begin{aligned}
\mathrm{Tr}(A^\pi, B) &= \sum_{l,m \in [p]} \left(\sum_{i \in S_l,\, j \in S_m} \langle U^{i^\pi}, U^{j^\pi} \rangle_\Lambda \right) \cdot \langle \tilde{V}^l, \tilde{V}^m \rangle_\Gamma \\
&= \sum_{l,m \in [p]} \left\langle \sum_{i \in S_l} U^{i^\pi}, \sum_{j \in S_m} U^{j^\pi} \right\rangle_\Lambda \cdot \langle \tilde{V}^l, \tilde{V}^m \rangle_\Gamma \\
&= \sum_{l,m \in [p]} |S_l| \cdot |S_m| \cdot \left\langle \widehat{w}_{S_l^\pi}, \widehat{w}_{S_m^\pi} \right\rangle_\Lambda \cdot \langle \tilde{V}^l, \tilde{V}^m \rangle_\Gamma
\end{aligned}$$

Define the partition P_π of $[n]$ to be $P_\pi = (S_1^\pi, \ldots, S_p^\pi)$. Observe that P_π is of type Δ, and therefore, $P_\pi \in \mathcal{P}_\Delta$. Using the definition of the matrix K, we can thus rewrite

$$\begin{aligned}\mathrm{Tr}(A^\pi, B) &= \sum_{l,m\in[p]} |S_l| \cdot |S_m| \cdot \left\langle \widehat{w}_{S_l^\pi}, \widehat{w}_{S_m^\pi} \right\rangle_\Lambda \cdot \langle \tilde{V}^l, \tilde{V}^m \rangle_\Gamma \\ &= \sum_{l,m\in[p]} \left\langle \widehat{w}_{S_l^\pi}, \widehat{w}_{S_m^\pi} \right\rangle_\Lambda \cdot K_{lm} \\ &= F(P_\pi),\end{aligned}$$

which allows us to state the following equality.

$$\|A^\pi - B\|_F^2 = \mathrm{Tr}(A, A) + \mathrm{Tr}(B, B) - 2F(P_\pi). \tag{2}$$

We continue with the proof of the lemma. Let P^* be an optimal solution for our QVP instance. In other words, the partition $P^* = (T_1^*, \ldots, T_p^*)$ is a maximizer of $F(P)$ over the set $\mathcal{P}_\Delta$. Let π^* be a permutation which maps the sets S_l to T_l^*, for all $l \in [p]$. We claim that π^* is an optimal solution for the MSim instance. To see this, suppose π^* is not optimal. Instead, let π' be an optimal solution for the MSim instance, and hence, $\|A^{\pi^*} - B\|_F^2 > \|A^{\pi'} - B\|_F^2$. Define a related partition $P_{\pi'} = (S_1^{\pi'}, \ldots, S_p^{\pi'})$: clearly, $\pi' \in \mathcal{P}_\Delta$. Since Equation 2 implies that

$$\begin{aligned}\|A^{\pi^*} - B\|_F^2 &= \mathrm{Tr}(A, A) + \mathrm{Tr}(B, B) - 2F(P^*), \\ \|A^{\pi'} - B\|_F^2 &= \mathrm{Tr}(A, A) + \mathrm{Tr}(B, B) - 2F(P_{\pi'}),\end{aligned}$$

we use $\|A^{\pi^*} - B\|_F^2 > \|A^{\pi'} - B\|_F^2$ to obtain that $F(P^*) < F(P_{\pi'})$. This contradicts the maximality of P^*. Hence, π^* must be an optimal solution for the QVP instance.

Given such an optimal solution P^* for the QVP instance, the computation of the optimal solution π^* for the MSim instance is a straightforward $\widetilde{\mathcal{O}}(n)$ procedure: we define π^* by choosing arbitrary bijections between the sets S_l and T_l^*, for all $l \in [p]$. This finishes the proof of our lemma. ◀

4.3 Linearization of Convex Functions

We take a small detour towards the properties of convex functions. These properties will be useful for studying the optimal solutions to the QVP problem. In general, we show that the linearization of a convex function can be useful in understanding its optima over a finite domain. In this context, we prove the following lemma about convex functions, which is interesting in its own right.

▶ **Lemma 14.** *Let Ω be a finite subset of $\mathbb{R}^k \times \mathbb{R}^\ell$. Let $G : \mathbb{R}^k \to \mathbb{R}$, $H : \mathbb{R}^\ell \to \mathbb{R}$ such that H is convex, and let $F : \mathbb{R}^k \times \mathbb{R}^\ell \to \mathbb{R}$ be defined as $F(X, Y) = G(X) + H(Y)$. Let $(X^*, Y^*) \in \arg\max_{(X,Y)\in\Omega} F(X, Y)$.*

Then there exist a $\mu^* \in \mathbb{R}^\ell$ such that:

- **(i)** *$(X^*, Y^*) \in \arg\max_{(X,Y)\in\Omega} L(X, Y)$ where $L(X, Y) = G(X) + \langle \mu^*, Y \rangle$;*
- **(ii)** *$\arg\max_{(X,Y)\in\Omega} L(X, Y) \subseteq \arg\max_{(X,Y)\in\Omega} F(X, Y)$.*

Proof. Let $(X^*, Y^*) \in \arg\max_{S\in\Omega} F(S)$. Since H is convex, we can use Theorem 5 to linearize H around $Y^* \in \mathbb{R}^\ell$. Hence, there exists a $\mu^* \in \mathbb{R}^\ell$ such that $H(Y) - H(Y^*) \geq \langle \mu^*, Y - Y^* \rangle$, or equivalently,

$$H(Y) - \langle \mu^*, Y \rangle \geq H(Y^*) - \langle \mu^*, Y^* \rangle, \tag{3}$$

for all $Y \in \mathbb{R}^\ell$. Hence with $L(X,Y) = G(X) + \langle \mu^*, Y \rangle$, for all $(X,Y) \in \Omega$ we have

$$L(X^*,Y^*) = F(X^*,Y^*) - H(Y^*) + \langle \mu^*, Y^* \rangle \geq F(X,Y) - H(Y) + \langle \mu^*, Y \rangle = L(X,Y),$$

where the inequality holds by (3) and because (X^*,Y^*) maximizes F. Hence (X^*,Y^*) maximizes L as well, which proves (i).

For (ii), consider $(X^{**},Y^{**}) \in \arg\max_{(X,Y)\in\Omega} L(X,Y)$. To prove that $(X^{**},Y^{**}) \in \arg\max_{(X,Y)\in\Omega} F(X,Y)$, it suffices to prove that $F(X^{**},Y^{**}) \geq F(X^*,Y^*)$. By (i), we have $L(X^*,Y^*) = L(X^{**},Y^{**})$. Thus

$$\begin{aligned} F(X^{**},Y^{**}) = L(X^{**},Y^{**}) + H(Y^{**}) - \langle \mu^*, Y^{**} \rangle &\geq L(X^*,Y^*) + H(Y^*) - \langle \mu^*, Y^* \rangle \\ &= F(X^*,Y^*), \end{aligned}$$

where the inequality holds by (3) with $(X,Y) := (X^{**},Y^{**})$ and as (X^{**},Y^{**}) maximizes L. ◀

In other words, for every (X^*,Y^*) which maximizes F over Ω, there exists a partially "linearized" function L such that (X^*,Y^*) maximizes L over Ω. Moreover, every maximizer of L over Ω is a maximizer of F over Ω. This additional condition is necessary so that this "linearization" does not create spurious optimal solutions.

▶ **Corollary 15.** *Let Ω be a finite subset of $\mathbb{R}^{kp}$. For all $i \in [k]$, let $G_i : \mathbb{R}^k \to \mathbb{R}$ be a convex function. Let $F : \mathbb{R}^{kp} \to \mathbb{R}$ be defined as $F(X_1,\ldots,X_k) := G_1(X_1) + \ldots + G_k(X_k)$. Let $X^* = (X_1^*,\ldots,X_k^*) \in \arg\max_{X\in\Omega} F(X)$.*

Then there are $\mu_1^*,\ldots,\mu_k^* \in \mathbb{R}^p$ such that:

(i) *$X^* \in \arg\max_{X\in\Omega} L(X)$ where $L(X_1,\ldots,X_k) = \sum_{i=1}^k \langle \mu_i^*, X_i \rangle$;*

(ii) *$\arg\max_{X\in\Omega} L(X) \subseteq \arg\max_{\in\Omega} F(X)$.*

Proof. Inductively apply the lemma to the functions

$$F^i((X_1,\ldots,X_{i-1},X_{i+1},\ldots,X_k),X_i) = \underbrace{\left(\sum_{j=1}^{i-1}\langle \mu_j^*, X_j\rangle + \sum_{j=i+1}^{k} G_j(X_j)\right)}_{=:G^i(X_1,\ldots,X_{i-1},X_{i+1},\ldots,X_k)} + \underbrace{G_i(X_i)}_{=:H^i(X_i)}. \quad ◀$$

4.4 Optimal Solution Structure for QVP

Let us express the QVP objective function

$$F(P) = \sum_{l,m\in[p]} K_{lm} \langle \widehat{w}_{T_l}, \widehat{w}_{T_m} \rangle_\Lambda$$

as a restriction of a convex function to a finite domain. Using the results above for linearization of convex functions, we show that the optimal solutions for a QVP instance must satisfy certain structural constraints, specified by Lemma 19.

Formally, given a QVP instance (W,K,Λ,Δ), and a partition $P = (T_1,\ldots,T_p) \in \mathcal{P}_\Delta$, we define k vectors $X_1,\ldots,X_k$ as follows. For $q \in [k]$, let X_q be the vector of length p corresponding to the q^{th} coordinates of vectors $\widehat{w}_{T_1},\ldots,\widehat{w}_{T_p}$. Clearly, the vectors $X_1,\ldots,X_k$ are a function of the partition P. Recall that Λ is a diagonal matrix with k non-negative entries, say $\lambda_1,\ldots,\lambda_k$.

▶ **Claim 16.** $F(P) = \sum_{q=1}^{k} \lambda_q \langle X_q, X_q \rangle_K$.

Observe that the function $G : \mathbb{R}^p \mapsto \mathbb{R}$ defined by $G(Y) = \langle Y, Y \rangle_K$ is a convex function (by Lemma 6). Define a function $\tilde{F}(Y_1, \ldots, Y_p) = \lambda_1 G(Y_1) + \cdots + \lambda_k G(Y_k)$, where the vectors $Y_1, \ldots, Y_k \in \mathbb{R}^p$. This function is convex as well: it is a linear combination of convex functions, with positive co-efficients. Observe that $F(P) = \tilde{F}(X_1, \ldots, X_p)$. Therefore, the problem of maximizing F over $\mathcal{P}_\Delta$ is, essentially, a problem of maximizing a convex function $\tilde{F}(Y_1, \ldots, Y_k)$ over a finite discrete domain.

Using Corollary 15, we claim that a maximizer $P^* = (T_1^*, \ldots, T_p^*)$ of the objective function $F(P)$ over the domain $\mathcal{P}_\Delta$ must be a maximizer for some *linear* objective function $L_1(P)$ over the domain $\mathcal{P}_\Delta$.

▶ **Claim 17.** *There exist vectors $\mu_1^*, \ldots, \mu_k^* \in \mathbb{R}^p$ such that P^* is a maximizer of the function $L_1(P) = \sum_{q=1}^{k} \lambda_q \langle \mu_q^*, X_q \rangle_K$ over $\mathcal{P}_\Delta$. Moreover, every maximizer of $L_1(P)$ is a maximizer of $F(P)$.*

We can further reformulate the optimality conditions of Claim 17 as follows.

▶ **Claim 18.** *There exist vectors $\mu_1, \ldots, \mu_p \in \mathbb{R}^k$ such that P^* is a maximizer of the function $L_2(P) = \sum_{m=1}^{p} \langle \mu_m, \widehat{w}_{T_m} \rangle$ over $\mathcal{P}_\Delta$. Moreover, every maximizer of $L_2(P)$ is a maximizer of $L_1(P)$, and consequently, a maximizer of $F(P)$.*

The above claim leads to a strong geometrical restriction on the partition $W[T_1^*], \ldots, W[T_p^*]$ of W, induced by the optimal partition $P^* \in \mathcal{P}_\Delta$.

▶ **Lemma 19.** *Let $P^* = (T_1^*, \ldots, T_p^*)$ be an optimal solution for a* QVP *instance (W, K, Λ, Δ). The partition $(W[T_1^*], \ldots, W[T_p^*])$ of the set W is (weakly) mutually linearly separated.*

Proof. By Claim 18, there exist vectors $\mu_1, \ldots, \mu_p \in \mathbb{R}^k$ such that P^* is a maximizer of $L_2(P)$. Suppose there exist q, r such that $W[T_q^*]$ and $W[T_r^*]$ are not (weakly) linearly separated. We claim that this leads to a contradiction.

Let n_q and n_r be the cardinalities of the sets T_q^* and T_r^*. We use the notation W^T to denote the sum of the vectors in the set $W[T]$, for a subset $T \subseteq [n]$. Let us isolate the two terms $\langle \mu_q, \widehat{w}_{T_q^*} \rangle + \langle \mu_r, \widehat{w}_{T_r^*} \rangle$ and rewrite them as $\langle \frac{\mu_q}{n_q}, W^{T_q^*} \rangle + \langle \frac{\mu_r}{n_r}, W^{T_r^*} \rangle$. Let us denote $\frac{1}{n_q}\mu_q$ by μ'_q, and $\frac{1}{n_r}\mu_r$ by μ'_r. Therefore, the terms can be re-written as $\langle \mu'_q, W^{T_q^*} \rangle + \langle \mu'_r, W^{T_r^*} \rangle$.

Rewriting further, we can express the above terms as $\langle (\mu'_q - \mu'_r), W^{T_q^*} \rangle + \langle \mu'_r, (W^{T_q^*} + W^{T_r^*}) \rangle$. Recall that the sets $W[T_q^*]$ and $W[T_r^*]$ are not weakly linearly separated. Let us partition the set $W[T_q^*] \cup W[T_r^*]$ into two sets $W[T'_q]$ and $W[T'_r]$ such that (a) $|T'_q| = n_q$, $|T'_r| = n_r$ and (b) the sets $W[T'_q]$ and $W[T'_r]$ are weakly linearly separated along the direction $(\mu'_q - \mu'_r)$. Indeed, we can sort all elements w in $W[T_q^*] \cup W[T_r^*]$ in a descending order, according to their (signed) projection $\langle (\mu'_q - \mu'_r), w \rangle$ along $(\mu'_q - \mu'_r)$. Pick the top n_q elements in this ordering to obtain the set T'_q and collect the remaining n_r elements to form the set T'_r. Note that the sets T'_q and T'_r are weakly linearly separated along the direction $(\mu'_q - \mu'_r)$, and hence, the pair $(T_q^*, T_r^*) \neq (T'_q, T'_r)$.

Clearly, $\langle (\mu'_q - \mu'_r), W^{T'_q} \rangle > \langle (\mu'_q - \mu'_r), W^{T_q^*} \rangle$ by our construction. Moreover, $\langle \mu'_r, (W^{T'_q} + W^{T'_r}) \rangle = \langle \mu'_r, (W^{T_q^*} + W^{T_r^*}) \rangle$, because $T'_q \cup T'_r = T_q^* \cup T_r^*$. Hence, $\langle \mu'_q, W^{T'_q} \rangle + \langle \mu'_r, W^{T'_r} \rangle > \langle \mu'_q, W^{T_q^*} \rangle + \langle \mu'_r, W^{T_r^*} \rangle$, which implies that $\langle \mu_q, \widehat{w}_{T'_q} \rangle + \langle \mu_r, \widehat{w}_{T'_r} \rangle > \langle \mu_q, \widehat{w}_{T_q^*} \rangle + \langle \mu_r, \widehat{w}_{T_r^*} \rangle$. This contradicts the maximality of P^* for the function $L_2(P)$ over the domain $\mathcal{P}_\Delta$. ◀

5 Proof of Theorem 1

In this section, we prove the following algorithmic result about QVP.

▶ **Theorem 20.** *Given a* QVP *instance* (W, K, Λ, Δ)*, we can compute an optimal solution for this instance in* $\widetilde{\mathcal{O}}(n^{kp^2})$ *time.*

In this section, we will prove Theorem 20 in a restricted setting: we assume that the set W is in General Position (G.P.). The proof for the general case is not very different: using a technical tool to handle degeneracies in W, we can reduce the general case to this restricted case. We defer the proof of Theorem 20 (i.e., the general case) to the full version of the paper, and continue with the proof for this restricted setting.

Observe that the proof of Theorem 1 follows immediately from the above theorem.

Proof of Theorem 1. Let (A, B) be an MSim instance, as defined in the statement of Theorem 1. Using the reduction in Lemma 13, we can compute a QVP instance (W, K, Λ, Δ) in $\widetilde{\mathcal{O}}(n^3)$ with the following property: an optimal solution to this QVP instance can be used to compute an optimal solution for the MSim instance (A, B), in $\mathcal{O}(n)$ time. Using Theorem 20, we can compute an optimal solution for the QVP instance (W, K, Λ, Δ) in $\widetilde{\mathcal{O}}(n^{kp^2})$ time. Therefore, we can compute an optimal solution for the MSim instance in overall $\widetilde{\mathcal{O}}(n^{kp^2})$ time. ◀

5.1 Algorithm for QVP, restricted version

We proceed with the proof of Theorem 20, under the G.P. assumption. In other words, given a QVP instance (W, K, Λ, Δ), the input set W is in General Position. Recall that a set S of n points $w_1, \ldots, w_n \in \mathbb{R}^k$ is said to be in *general position* (G.P.), if there is no subset $S' \subseteq S$ with $|S'| > k$ that lies on a common hyperplane. Moreover, we can associate a unique hyperplane H_S with every k-element subset S of W. Let $\mathcal{H}$ be the set of $\binom{n}{k}$ hyperplanes, defined by each k-element subset of W. Under the G.P. assumption, we can further strengthen Lemma 19, as follows.

▶ **Lemma 21.** *Let* $P^* = (T_1, \ldots, T_p)$ *be an optimal solution for a* QVP *instance* (W, K, Λ, Δ)*. For every pair of sets* $W[T_i]$ *and* $W[T_j]$*, where* $i < j$*, there exists a hyperplane* H_{ij} *in the set* $\mathcal{H}$ *such that* $W[T_i]$ *and* $W[T_j]$ *are weakly linearly separated along* H_{ij}*.*

The proof of this lemma follows immediately from the following claim.

▶ **Claim 22.** *Let* W *be a set of* n *points* $\{w_1, \ldots, w_n\} \subset \mathbb{R}^k$ *in general position, where* $n > k$*. Suppose* W_1, W_2 *are two disjoint subsets of* W *which are weakly linearly separated by a hyperplane* H*. Then, there exists another hyperplane* $\tilde{H}$ *with the following properties: (a)* $\tilde{H}$ *passes through exactly* k *points of* W*, and (b)* $\tilde{H}$ *also weakly linearly separates* W_1, W_2*.*

Enumerative Algorithm. We proceed with an informal description of the algorithm. The overall strategy of our algorithm follows from Lemma 19 and Lemma 21. We will enumerate a particular subset $\mathcal{P}$ of $\mathcal{P}_\Delta$ defined as follows. The set $\mathcal{P}$ is the set of all weakly linearly separated partitions $P = (T_1, \ldots, T_p)$ of W with the following property (stated in Lemma 21). For every pair of sets $W[T_i]$ and $W[T_j]$, where $i < j$, there exists a hyperplane H_{ij} in $\mathcal{H}$ such that $W[T_i]$ and $W[T_j]$ are weakly linearly separated along H_{ij}. Clearly, we can maximize the objective function $F(P)$ *over the set* $\mathcal{P}$, instead of the original domain $\mathcal{P}_\Delta$:

by Lemma 21, an optimal solution must lie in the set $\mathcal{P}$. Therefore, it suffices to prove the following lemma.

▶ **Lemma 23** (Enumeration, under G.P. assumption). *Given a* QVP *instance* (W, K, Λ, Δ)*, assume that the set* W *is in General Position. Then, we can enumerate the set* $\mathcal{P}$ *in* $\widetilde{\mathcal{O}}(n^{kp^2})$ *time.*

Proof. From Lemma 21, we can deduce that a partition $P = (T_1, \ldots, T_p) \in \mathcal{P}$ can be associated with a sequence of $\binom{p}{2}$ separating hyperplanes $H_{ij} \in \mathcal{H}$, $i < j$, $i, j \in [p]$. In particular, the hyperplane H_{ij} weakly linearly separates $W[T_i]$ and $W[T_j]$.

Therefore, we enumerate the set $\mathcal{P}$ as follows. We branch over every choice of $|\mathcal{H}|^{\binom{p}{2}} \leq n^{\frac{kp^2}{2}}$ sequences of hyperplanes. We can define p convex regions $R_1, \ldots, R_p$ using these hyperplanes; the region R_i is supposed to contain the set $W[T_i]$, $i \in [p]$.

We assign the elements of W to these p disjoint convex regions $R_1, \ldots, R_p$. It is possible that an element w_j might lie on one or more of the hyperplanes H_{ij}. For such an 'ambigious' point, we brute-force try all possible p assignments of regions R_i. Since every hyperplane in $\mathcal{H}$ contains exactly k points of W, there can be at most $\binom{p}{2} \cdot k$ such ambigious points: this leads to an additional branching factor of at most $p^{\binom{p}{2} \cdot k}$. For each such branch, we obtain a partition $(W_1, \ldots, W_p)$ of W. If the type of this partition is not equal to Δ, we reject it; otherwise we add it to the list $\mathcal{P}$. The overall branching is bounded by $n^{\frac{kp^2}{2}} \cdot p^{\binom{p}{2} \cdot k}$ which is bounded by $n^{\frac{kp^2}{2}} \cdot n^{\frac{kp^2}{2}} \leq n^{kp^2}$. The overall running time is bounded by $\widetilde{\mathcal{O}}(n^{kp^2})$.

Clearly, every partition P in $\mathcal{P}$ can be discovered along some branch of our computation: we branch over all hyperplane sequences and further, over all assignments of 'ambigious' points. Moreover, every partition enumerated above belongs to $\mathcal{P}$, by our construction. Our overall branching factor of $\widetilde{\mathcal{O}}(n^{kp^2})$ is also an upper bound on the cardinality of $\mathcal{P}$. This finishes the proof of the lemma. ◀

Since we can enumerate the set $\mathcal{P}$ in $\widetilde{\mathcal{O}}(n^{kp^2})$ time, the optimal solution can be computed in a similar time as well. We summarize the above discussion as the following theorem.

▶ **Theorem 24** (QVP algorithm, G.P. assumption). QVP *can be solved in* $\widetilde{\mathcal{O}}(n^{kp^2})$ *running time.*

6 Conclusion

Through our results, we were able to gain insight into the tractibility of the problems GSim and MSim. However, there are a few open threads which remain elusive. The regime of bounded rank k and unbounded parameter $\tau(G)$ is still not fully understood for MSim, in the case of positive semi-definite matrices. It is not clear whether the problem is P-time or NP-hard in this case. Indeed, an $n^{O(k)}$ algorithm for MSim, in the case of positive semi-definite matrices, remains a possibility. From the perspective of parameterized complexity, we can ask if MSim is W[1]-hard, where the parameter of interest is the rank k. Finally, the approximability for the problems MSim deserves further examination, especially for the case of bounded rank.

References

1 H.A. Almohamad and S.O. Duffuaa. A linear programming approach for the weighted graph matching problem. *IEEE Transactions on pattern analysis and machine intelligence*, 15(5):522–525, 1993.

2 S. Arora, A. Frieze, and H. Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. *Mathematical programming*, 92(1):1–36, 2002.

3 V. Arvind, J. Köbler, S. Kuhnert, and Y. Vasudev. Approximate graph isomorphism. In B. Rovan, V. Sassone, and P. Widmayer, editors, *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science*, volume 7464 of *Lecture Notes in Computer Science*, pages 100–111. Springer Verlag, 2012.

4 L. Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC '16)*, pages 684–697, 2016.

5 R.E. Burkard, E. Cela, G. Rote, and G.J. Woeginger. The quadratic assignment problem with a monotone anti-monge and a symmetric toeplitz matrix: easy and hard cases. *Mathematical Programming*, 82:125–158, 1998.

6 E. Cela. *The Quadratic Assignment Problem: Theory and Algorithms*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

7 E. Cela, V.G. Deineko, and G.J. Woeginger. Well-solvable cases of the qap with block-structured matrices. *Discrete Applied Mathematics*, 186:56–65, 2015.

8 E. Cela, N. Schmuck, S. Wimer, and G.J. Woeginger. The wiener maximum quadratic assignment problem. *Discrete Optimization*, 8:411–416, 2011.

9 P. Codenotti, H. Katebi, K. A. Sakallah, and I. L. Markov. Conflict analysis and branching heuristics in the search for graph automorphisms. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4-6, 2013*, pages 907–914, 2013.

10 D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, 18(3):265–298, 2004.

11 A.N. Elshafei. Hospital layout as a quadratic assignment problem. *Operational Research Quarterly*, 28:167–179, 1977.

12 M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

13 A.M. Geoffrion and G.W. Graves. Scheduling parallel production lines with changeover costs: Practical application of a quadratic assignment/lp approach. *Operational Research*, 24:595–610, 1976.

14 S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on pattern analysis and machine intelligence*, 18(4):377–388, 1996.

15 T. Junttila and P. Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics*, pages 135–149. SIAM, 2007.

16 P. Keldenich. Random robust graph isomorphism. Master's thesis, Department of Compter Science, RWTH Aachen University, 2015.

17 Y. Koren. Drawing graphs by eigenvectors: theory and practice. *Computers and Mathematics with Applications*, 49(11):1867–1888, 2005.

18 J. Krarup and Pruzan P.M. Computer-aided layout design. *Mathematical Programming Study*, 9:75–94, 1978.

19 K. Makarychev, R. Manokaran, and M. Sviridenko. Maximum quadratic assignment problem: Reduction from maximum label cover and lp-based approximation algorithm. *ACM Transactions on Algorithms*, 10(4):18, 2014.

20 D.W. Matula. Subtree isomorphism in $o(n^{5/2})$. In P. H. B. Alspach and D. Miller, editors, *Algorithmic Aspects of Combinatorics*, volume 2 of *Annals of Discrete Mathematics*, pages 91–106. Elsevier, 1978.

21 B. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

22 B. D. McKay and A. Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014.

23 S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings. 18th International Conference on Data Engineering*, pages 117–128, 2002.

24 V. Nagarajan and M. Sviridenko. On the maximum quadratic assignment problem. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 516–524, 2009.

25 D. Neuen and P. Schweitzer. Benchmark graphs for practical graph isomorphism. *ArXiv (CoRR)*, arXiv:1705.03686 [cs.DS], 2017.

26 R. O'Donnell, J. Wright, C. Wu, and Y. Zhou. Hardness of robust graph isomorphism, Lasserre gaps, and asymmetry of random graphs. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1659–1677, 2014.

27 M.A. Pollatschek, N. Gershoni, and Y.T. Radday. Optimization of the typewriter keyboard by simulation. *Angewandte Informatik*, 17:438–439, 1976.

28 F. Rendl and H. Wolkowicz. Applications of parametric programming and Eigenvalue maximization to the quadratic assignment problem. *Mathematical Programming*, 53:63–78, 1992.

29 S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE transactions on pattern analysis and machine intelligence*, 10(5):695–703, 1988.

30 M. Zaslavskiy, F. Bach, and J.-P. Vert. A path following algorithm for the graph matching problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2227–2242, 2009.

Finding Short Synchronizing Words for Prefix Codes

Andrew Ryzhikov
Université Paris-Est, LIGM, Marne-la-Vallée, France
ryzhikov.andrew@gmail.com

Marek Szykuła[1]
Institute of Computer Science, University of Wrocław, Wrocław, Poland
msz@cs.uni.wroc.pl

Abstract

We study the problems of finding a shortest synchronizing word and its length for a given prefix code. This is done in two different settings: when the code is defined by an arbitrary decoder recognizing its star and when the code is defined by its literal decoder (whose size is polynomially equivalent to the total length of all words in the code). For the first case for every $\varepsilon > 0$ we prove $n^{1-\varepsilon}$-inapproximability for recognizable binary maximal prefix codes, $\Theta(\log n)$-inapproximability for finite binary maximal prefix codes and $n^{\frac{1}{2}-\varepsilon}$-inapproximability for finite binary prefix codes. By c-inapproximability here we mean the non-existence of a c-approximation polynomial time algorithm under the assumption P $\neq$ NP, and by n the number of states of the decoder in the input. For the second case, we propose approximation and exact algorithms and conjecture that for finite maximal prefix codes the problem can be solved in polynomial time. We also study the related problems of finding a shortest mortal and a shortest avoiding word.

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

Keywords and phrases synchronizing word, mortal word, avoiding word, Huffman decoder, inapproximability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.21

Acknowledgements The first author would like to thank Dominique Perrin for many useful discussions. We are also grateful to anonymous reviewers for their comments that improved presentation of the paper.

1 Introduction

Prefix codes are a simple and powerful class of variable-length codes that are widely used in information compression and transmission. A famous example of prefix codes is Huffman's codes [15]. In general, variable length codes are not resistant to errors, since one deletion, insertion or change of a symbol can desynchronize the decoder causing incorrect decoding of all the remaining part of the message. However, in a large class of codes called synchronizing codes resynchronization of the decoder is possible in such situations. It is known that almost all maximal finite prefix codes are synchronizing [12]. Synchronization of finite prefix codes has been investigated a lot [5, 7, 8, 10, 21, 22], see also the book [6] and references therein. For efficiency reasons, it is important to use as short words resynchronizing the decoder as

[1] Supported in part by the National Science Centre, Poland under project numbers 2017/25/B/ST6/01920 and 2014/15/B/ST6/00615.

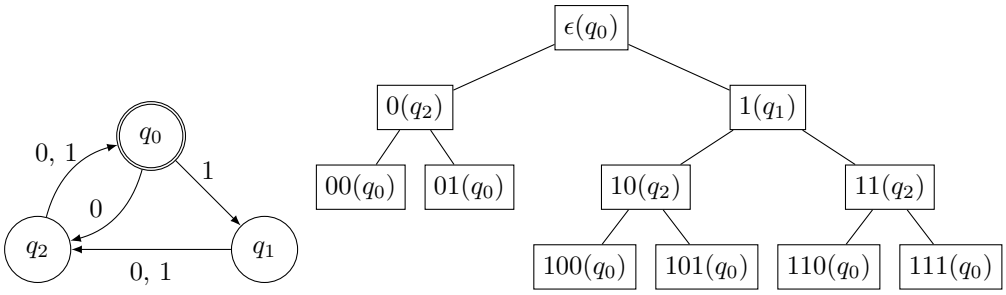


Figure 1 The Wielandt automaton on three states and the tree of the code $0\{0,1\} \cup 1\{0,1\}^2$.

possible to decrease synchronization time. However, despite the interest in synchronizing prefix codes, the computational complexity of finding short synchronizing words for them has not been studied so far. In this paper, we provide a systematic investigation of this topic.

Each recognizable (by a finite automaton) maximal prefix code can be represented by an automaton decoding the star of this code. For a finite code, this automaton can be exponentially smaller than the representation of the code by listing all its words (consider, for example, the code of all words of some fixed length). This can, of course, happen even if the code is synchronizing. An important example here is the code $0\{0,1\}^{n-1} \cup 1\{0,1\}^n$. The minimized decoder of this code is a famous Wielandt automaton with $n+1$ states (see ex. [1]), while the literal automaton contains $2^{n-1} + 2^n$ states, see Figure 1 for the case $n = 3$. In different applications, the first or the second way of representing the code can be useful. In some cases large codes having a short description may be represented by a minimized decoder, while in other applications the code can be described by simply providing the list of all codewords. The number of states of the literal decoder is equal to the number of different prefixes of the codewords, and thus the representations of a prefix code by listing all its codewords and by providing its literal automaton are polynomially equivalent. We study the complexity of problems for both arbitrary and literal decoders of finite prefix codes.

In this paper we study the existence of approximation algorithms for the problem Short Sync Word of finding a shortest synchronizing words in several classes of deterministic automata decoding prefix codes. In Section 2 we describe main definitions and survey existing results in the computational complexity of Short Sync Word. In Section 3 we provide a strong inapproximability result for this problem in the class of strongly connected automata. Section 4 is devoted to the same problem in acyclic automata, which are then used in Section 5 to show logarithmic inapproximability of Short Sync Word in the class of Huffman decoders. In Section 6 we provide a much stronger inapproximability result for partial Huffman decoders. In Section 7 we provide several algorithms for literal Huffman decoders and conjecture that Short Sync Word can be solved in polynomial time in this class. Finally, in Section 8 we apply the developed techniques to the problems of finding shortest mortal and avoiding words.

2 Main Definitions and Related Results

A *partial deterministic finite automaton* (which we simply call a *partial automaton* in this paper) is a triple $A = (Q, \Sigma, \delta)$, where Q is a set of states, Σ is a finite alphabet and $\delta : Q \times \Sigma \to Q$ is a (possibly incomplete) transition function. The function delta can be canonically extended to a function $\delta : Q \times \Sigma^* \to Q$ by defining $\delta(q, wx) = \delta(\delta(q, w), x)$ for

$x \in \Sigma$, $w \in \Sigma^*$. If δ is a complete function, the automaton is called *complete* (in this case we call it just an *automaton*). An automaton is called *strongly connected* if for every ordered pair q, q' of states there is a word mapping q to q'.

A state in a partial automaton is called *sink* if each letter either maps the state to itself or is undefined. A *simple cycle* in a partial automaton $A = (Q, \Sigma, \delta)$ is a sequence $q_1, \dots, q_k$ of its states such that all the states in the sequence are different and there exist letters $x_1, \dots, x_k \in \Sigma$ such that $\delta(q_i, x_i) = q_{i+1}$ for $1 \leq i \leq k-1$ and $\delta(q_k, x_k) = q_1$. A simple cycle is a *self-loop* if it consists of only one state. We call a partial automaton *weakly acyclic* if all its cycles are self-loops, and *strongly acyclic* if moreover all its states with self-loops are sink states. Some properties of these automata have been studied in [19].

There is a strong relation between partial automata and prefix codes [6]. A set X of words is called a *prefix code* if no word in X is a prefix of another word. The class of recognizable (by an automaton) prefix codes can be described as follows. Take a strongly connected partial automaton A and pick a state r in it. Then the set of all *first return words* of r (that is, words mapping r to itself such that each non-empty prefix does not map r to itself) is a recognizable prefix code. Moreover, each recognizable prefix code can be obtained this way. A prefix code is called *maximal* if it is not a subset of another prefix code. The class of maximal recognizable prefix codes corresponds to the class of complete automata. If a state r can be picked in an automaton in such a way that the set of all first return words is a finite prefix code, we call the automaton a *partial Huffman decoder*. If such automaton is complete (and thus the finite prefix code is maximal), we call it simply a *Huffman decoder*.

Let A be a partial automaton. A word w is called *synchronizing* for A if there exists a state q such that w maps each state of A either to q or the mapping of w is undefined for this state, and there is at least one state such that the mapping of w is defined for it. That is, a word is called synchronizing if it maps the whole set of states of the automaton to a set of size exactly one. An automaton having a synchronizing word is called *synchronizing*. A recognizable prefix code is *synchronizing* if a trim (partial) automaton recognizing the star of this code is synchronizing [6] (an automaton is called *trim* if there exists a state such that each state is accessible from this state, and there exists a state such that each state is coaccessible from this state). It can be checked in polynomial time that a strongly connected partial automaton is synchronizing (Proposition 3.6.5 of [6]).

Synchronizing automata have applications in different domains, such as synchronizing codes, symbolic dynamics, manufacturing and testing of reactive systems. They are also the subject of the Černý conjecture, one of the main open problems in automata theory. It stays that every n-state synchronizing automaton has a synchronizing word of length at most $(n-1)^2$, while the best known upper bounds are cubic [17, 24]. See [26] for a survey on this topic. The upper bound on the length of a shortest synchronizing word has been improved in particular for Huffman decoders [2] and further for literal Huffman decoders [5].

We consider the following computational problem.

> Short Sync Word
> *Input*: A synchronizing partial automaton A;
> *Output*: The length of a shortest synchronizing word for A.

Now we shortly survey existing results and techniques in the computational complexity and approximability of finding shortest synchronizing words for deterministic automata. To the best of our knowledge, there are no such results for partial automata. See [23] for an introduction to NP-completeness and [25] for an introduction to inapproximability and gap-preserving reductions.

There exist several techniques of proving that Short Sync Word is hard for different classes of automata. The very first and the most widely used idea is the one of Eppstein [11]. Here, the automaton in the reduction is composed of a set of "pipes", and transitions define the way the active states are changed inside the pipes to reach the state where synchronization takes place. This idea (sometimes extended a lot) allows to prove NP-completeness of Short Sync Word in the classes of strongly acyclic [11], ternary Eulerian [16], binary Eulerian [27], binary cyclic [16] automata. This idea is also used in the proofs of [3] for inapproximability within arbitrary constant factor for binary automata, and for $n^{1-\varepsilon}$-inapproximability for n-state binary automata [13] (the last proof uses the theory of Probabilistically Checkable Proofs). In fact, the proof in [13] holds true for binary automata with linear (in the number of states of the automaton) length of a shortest synchronizing word and a sink state.

Another idea is to construct a reduction from the Set Cover problem. It can be used to show logarithmic inapproximability of the Short Sync Word in weakly acyclic [14] and binary automata [4]. Finally, a reduction from Shortest Common Supersequence provides inapproximability of this problem within a constant factor [14].

In the class of monotonic automata Short Sync Word is solvable in polynomial time: because of the structure of these automata this problem reduces to a problem of finding a shortest words synchronizing a pair of states [20]. For general n-state automata, a $\lceil \frac{n-1}{k-1} \rceil$-approximation polynomial time algorithm exists for every k [14].

3 The Construction of Gawrychowski and Straszak

In this section we briefly recall the construction of a gadget invented by Gawrychowski and Straszak [13] to show $n^{1-\varepsilon}$-inapproximability of the Short Sync Word problem in the general class of automata. Below we will use this construction several times.

Suppose that we have a constraint satisfiablity problem (CSP) with N variables and M constraints such that each constraint is satisfied by at most K assignments (see [13] for the definitions and missing details). Following the results in [13], we can assume that $N, K \leq M^{\varepsilon}$, and also that either the CSP is satisfiable, or at most $\frac{1}{M^{1-\varepsilon}}$ fraction of all constraints can be satisfied by an assignment. It is possible to construct the following ternary automaton A_ϕ in polynomial time. For each constraint C the automaton A_ϕ contains a corresponding binary (over $\{0,1\}$) gadget T_C which is a compressed tree (that is, an acyclic digraph) of height N and the number of states at most N^2K having different leaves corresponding to satisfying and non-satisfying assignments. The automaton A_ϕ also contains a sink state s such that all the leaves corresponding to satisfying assignments are mapped to s, and all other leaves are mapped to the roots of the corresponding trees. The third letter is defined to map all the states of each gadget to its root and to map s to itself. For every $\varepsilon > 0$ it is possible to construct such an automaton with at most $MN^2K \leq M^{1+3\varepsilon}$ states in polynomial time. Moreover, for a satisfiable CSP we get an automaton with a shortest synchronizing word of length at most $N + 1 = O(M^{\varepsilon})$, and for a non-satisfiable CSP the length of a shortest synchronizing word is at least $NM^{1-\varepsilon} \geq M^{1-\varepsilon}$. Since ε can be chosen arbitrary small, this provides a gap-preserving reduction with a gap of $M^{1-\varepsilon}$.

The described construction can be modified to get the same inapproximability in the class of strongly connected automata.

▶ **Theorem 1.** *The* Short Sync Word *problem cannot be approximated in polynomial time within a factor of $n^{1-\varepsilon}$ for every $\varepsilon > 0$ for n-state binary strongly connected automata unless $P = NP$.*

Proof. Consider the automaton A_ϕ described above. Add a new letter c that cyclically permutes the roots of all gadgets, maps s to the root of one of the gadgets and acts as a self-loop for all the remaining states. Observe that thus constructed automaton A is strongly connected and has the property that every non-satisfying assignment satisfies at most the fraction of $\frac{1}{M^{1-\varepsilon}}$ of all constraints. Thus, the gap between the length of a shortest synchronizing word for a satisfying and non-satisfying assignment is still $\Theta(M^{1-\varepsilon})$.

It remains to make the automaton binary. This can be done by using Lemma 3 of [4]. This way we get a binary automaton with $\Theta(M^{1+3\varepsilon})$ states and a gap between $\Theta(M^{\varepsilon})$ and $\Theta(M^{1-\varepsilon})$ in the length of a shortest synchronizing word. By choosing appropriate small enough ε, we get a reduction with gap $n^{1-\varepsilon}$ for binary strongly connected n-state automata, which proves the statement. ◀

4 Acyclic Automata

In this section we investigate the simply-defined classes of weakly acyclic and strongly acyclic automata. The results for strongly acyclic automata are used in Section 5 to obtain inapproximability for Huffman decoders. Even though the automata in the classes of weakly and strongly acyclic automata are very restricted and have a very simple structure, the inapproximability bounds for them are quite strong. Thus we believe that these classes are of independent interest.

We will need the following problem.

> Set Cover
> *Input*: A set X of p elements and a family C of m subsets of X;
> *Output*: A subfamily of C of minimum size covering X.

A family C' of subsets of X is said to *cover* X if X is a subset of the union of the sets in C'. For every $\gamma > 0$, the Set Cover problem with $|C| \leq |X|^{1+\gamma}$ cannot be approximated in polynomial time within a factor of $c' \log p$ for some $c' > 0$ unless P = NP [4].

▶ **Theorem 2.** *The* Short Sync Word *problem cannot be approximated in polynomial time within a factor of $c \log n$ for some $c > 0$ for n-state strongly acyclic automata over an alphabet of size $n^{1+\gamma}$ for every $\gamma > 0$ unless $P = NP$.*

Proof. We reduce the Set Cover problem. Provided X and C, we construct the automaton $A = (Q, \Sigma, \delta)$ as follows. To each set c_k in C we assign a letter $k \in \Sigma$. To each element x_j in X we assign a "pipe" of states $q_1^{(j)}, \ldots, q_p^{(j)}$ in Q. Additionally, we construct a state f in Q.

For $1 \leq i \leq p-1$ and all k and j we define $\delta(q_i^{(j)}, k) = f$ if c_k contains x_j, and $\delta(q_i^{(j)}, k) = q_{i+1}^{(j)}$ otherwise. We also define $\delta(q_p^{(j)}, k) = f$ for all j and k.

We claim that the length of a shortest synchronizing word for A is equal to the minimum size of a set cover in C. Let C' be a set cover of minimum size. Then a concatenation of the letters corresponding to the elements of C' is a synchronizing word of corresponding length.

In the other direction, consider a shortest synchronizing word w for A. No letter appears in w at least twice. If the length of w is less than p, then by construction of A the subset of elements in C corresponding to the letters in w form a set cover. Otherwise we can take an arbitrary subfamily of C of size p which is a set cover (such subfamily trivially exists if C covers X).

The resulting automaton has $p^2 + 1$ states and m letters. Thus we get a reduction with gap $c' \log p \geq c'' \log \sqrt{|Q|} = \frac{1}{2} c'' \log |Q|$ for some $c'' > 0$. Because of the mentioned result of Berlinkov, we can also assume that $m < p^{1+\gamma}$ for arbitrary small $\gamma > 0$. ◀

Now we are going to extend this result to the case of binary weakly acyclic automata.

▶ **Corollary 3.** *The* Short Sync Word *problem cannot be approximated in polynomial time within a factor of* $c \log n$ *for some* $c > 0$ *for* n*-state binary weakly acyclic automata unless* $P = NP$*.*

Proof. We extend the construction from the proof of Theorem 2 by using Lemma 3 of [4]. If we start with a strongly acyclic automaton with p states and k letters, this results in a binary weakly acyclic automaton with $4pk$ states. Moreover, the length of a shortest word of the new automaton is between $\ell(\log k + 1)$ and $(\ell + 1)(\log k + 1)$, where ℓ is the length of a shortest word of the original automaton. Since we can assume $p < k < p^{1+\gamma}$ for arbitrary small $\gamma > 0$, we have $\log n = \Theta(\log p)$, where n is the number of states of the new automaton. Thus we get a gap of $\Theta(\log p) = \Theta(\log n)$. ◀

For ternary strongly acyclic automata it is possible to get $(2 - \varepsilon)$-inapproximability.

▶ **Theorem 4.** *The* Short Sync Word *problem cannot be approximated in polynomial time within a factor of* $2 - \varepsilon$ *for every* $\varepsilon > 0$ *for* n*-state strongly acyclic automata over an alphabet of size three unless* $P = NP$*.*

5 Huffman Decoders

We start with a statement relating strongly acyclic automata to Huffman decoders.

▶ **Lemma 5.** *Let* A *be a synchronizing strongly acyclic automaton over an alphabet of size* k*. Let* ℓ *be the length of a shortest synchronizing word for* A*. Then there exists a Huffman decoder* A_H *over an alphabet of size* $k + 2$ *with the same length of a shortest synchronizing word, and* A_H *can be constructed in polynomial time.*

Proof. Provided a strongly acyclic automaton $A = (Q, \Sigma, \delta)$ we construct a Huffman decoder $A_H = (Q_H, \Sigma_H, \delta_H)$.

Since A is a synchronizing strongly acyclic automaton, it has a unique sink state f. We define the alphabet Σ_H as the union of Σ with two additional letters b_1, b_2. The set of states Q_H is the union of Q with some auxiliary states defined as follows. Consider the set S of states in A having no incoming transitions. Construct a full binary tree with the root f having S as the set of its leaves (if $|S|$ is not a power of two, some subtrees of the tree can be merged). Define b_1 to map each state of this tree to the left child, and b_2 to the right child. Transfer the action of δ to δ_H for all states in Q and all letters in Σ. For all the internal states of the tree define all the letters of Σ to map these states to f. Finally, for all the states in Q define the action of b_1, b_2 in the same way as some fixed letter in Σ.

Observe that any word w over alphabet Σ synchronizing A also synchronizes A_H. In the other direction, any synchronizing word for A_H has to synchronize Q, which means that each state in Q has to be mapped to f first, so the length of a shortest synchronizing word for A_H is at least the length of a shortest synchronizing word for A. ◀

Now we use Lemma 5 to get preliminary inapproximability results for Huffman decoders.

▶ **Corollary 6.**

(i) *The* Short Sync Word *problem is NP-complete for Huffman decoders over an alphabet of size* 4*.*

(ii) *The* Short Sync Word *problem cannot be approximated in polynomial time within a factor of* $2 - \varepsilon$ *for every* $\varepsilon > 0$ *for Huffman decoders over an alphabet of size* 5 *unless* $P = NP$*.*

(iii) *For every $\gamma > 0$, the* Short Sync Word *problem cannot be approximated in polynomial time within a factor of $c \log n$ for some $c > 0$ for Huffman decoders over an alphabet of size $n^{1+\gamma}$ unless $P = NP$.*

Proof.

(i) The automaton in the Eppstein's proof of NP-completeness of Short Sync Word [11] is strongly acyclic. Then the reduction described in Lemma 5 can be applied.

(ii) A direct consequence of Theorem 4 and Lemma 5.

(iii) A direct consequence of Theorem 2 and Lemma 5. ◀

Now we show how to get a better inapproximability result for binary Huffman decoders using the composition of synchronizing prefix codes. We present a more general result for the composition of synchronizing codes which is of its own interest. This result shows how to change the size of the alphabet of a synchronizing complete code in such a way that the approximate length of a shortest synchronizing pair for it is preserved.

A set X of words over an alphabet Σ is a *code* if no word can be represented as a concatenation of elements in X in two different ways. In particular, every prefix code is a code. A pair (ℓ_X, r_X) of words in X^* is called *absorbing* if $\ell_X\Sigma^* \cap \Sigma^* r_X \subseteq X^*$. The *length* of a pair is the total length of two word. A code X over an alphabet Σ is called *complete* if every word $w \in \Sigma^*$ is a factor of some word in X^*, that is, if for every word $w \in \Sigma^*$ there exist words $v_1, v_2 \in \Sigma^*$, $u \in X^*$ such that $v_1 w v_2 = u$. In particular, every maximal (by inclusion) code is complete. A complete code having an absorbing pair is called *synchronizing*. We refer to [6] for a survey on the theory of codes.

Let Y be a code over Σ_Y and Z be a code over Σ_Z. Suppose that there exists a bijection $\beta : \Sigma_Y \to Z$. The *composition* $Y \circ_\beta Z$ is then defined as the code $X = \{\beta(y) \mid y \in Y\}$ over the alphabet Σ_Z [6]. Here $\beta(y)$ is defined as $\beta(y_1)\beta(y_2)\dots\beta(y_k)$ for $y = y_1 y_2 \dots y_k$, $y_i \in \Sigma_Y$. Sometimes β is omitted in the notation of composition.

▶ **Theorem 7.** *Let Y and Z be two synchronizing complete codes, such that Z is finite and m and M are the lengths of a shortest and a longest codeword in Z. Suppose that the composition $Y \circ Z$ is defined. Then the code $X = Y \circ Z$ is synchronizing, and the length of a shortest absorbing pair for X is between $m\ell$ and $2M\ell + 2c$, where ℓ is the length of a shortest absorbing pair for Y and c is the length of a shortest absorbing pair for Z.*

Proof. Let $Y \subseteq \Sigma_Y^*$, $X, Z \subseteq \Sigma_Z^*$, and $\beta : \Sigma_Y \to Z$ be such that $X = Y \circ_\beta Z$. First, assume that Y and Z are synchronizing, and let (ℓ_Y, r_Y), (ℓ_Z, r_Z) be shortest absorbing pairs for Y and Z. Then $\ell_Y\Sigma_Y^* \cap \Sigma_Y^* r_Y \subseteq Y^*$ and $\ell_Z\Sigma_Z^* \cap \Sigma_Z^* r_Z \subseteq Z^*$. We will show that $p_1 = (\beta(\ell_Y)\ell_Z r_Z \beta(r_Y), \beta(\ell_Y)\ell_Z r_Z \beta(r_Y))$ is an absorbing pair for X. Consider the set $\beta(\ell_Y)\ell_Z r_Z \beta(r_Y)\Sigma_Z^* \cap \Sigma_Z^*\beta(\ell_Y)\ell_Z r_Z \beta(r_Y)$. It is a subset of the set $\beta(\ell_Y)(\ell_Z\Sigma_Z^* \cap \Sigma_Z^* r_Z)\beta(r_Y) \subseteq \beta(\ell_Y)Z^*\beta(r_Y) = \beta(\ell_Y\Sigma_Y^* r_Y) \subseteq \beta(Y^*) = X^*$. Thus, p_1 is an absorbing pair for X. Moreover, the length of this pair is between $2m(|\ell_Y| + |r_Y|) + 2(|\ell_Z| + |r_Z|)$ and $2M(|\ell_Y| + |r_Y|) + 2(|\ell_Z| + |r_Z|)$.

Conversely, assume that (ℓ_X, r_X) is a shortest absorbing pair for X, hence $\ell_X\Sigma_Z^* \cap \Sigma_Z^* r_X \subseteq X^*$. Then by the definition of composition $X^* \subseteq Z^*$ and $\ell_X, r_X \in Z^*$; thus, (ℓ_X, r_X) is also absorbing for Z. Next, let $\ell_Y = \beta^{-1}(\ell_X)$, $r_Y = \beta^{-1}(r_X)$, $\ell_Y, r_Y \in Y^*$. Then $\beta(\ell_Y\Sigma_Y^* \cap \Sigma_Y^* r_Y) = \ell_X Z^* \cap Z^* r_X \subseteq \ell_X\Sigma_Z^* \cap \Sigma_Z^* r_X \subseteq X^* = \beta(Y^*)$. Since the mapping β is injective, $\ell_Y B^* \cap B^* r_Y \subseteq Y^*$. Consequently Y is synchronizing, and (ℓ_Y, r_Y) is an absorbing pair for it of length between $\frac{1}{M}(|\ell_X| + |r_X|)$ and $\frac{1}{m}(|\ell_X| + |r_X|)$.

Summarizing, we get that the length of a shortest absorbing pair for X is between $m(|\ell_Y| + |r_Y|)$ and $2M(|\ell_Y| + |r_Y|) + 2(|\ell_Z| + |r_Z|)$. ◀

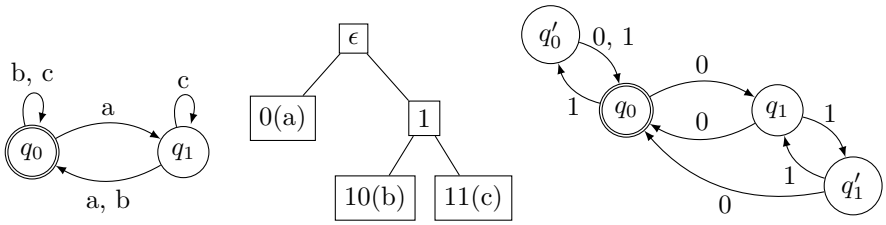


Figure 2 An automaton recognizing some infinite maximal prefix code, the tree of a finite maximal prefix code and an automaton recognizing their composition.

In the case of maximal prefix codes the first element of the absorbing pair can be taken as an empty word. For recognizable maximal prefix codes Y and Z, where Z is finite, a Huffman decoder recognizing the star of $X = Y \circ Z$ can be constructed as follows. Let H_Y be a Huffman decoder for Y. Consider the full tree T_Z for Z, where each edge is marked by the corresponding letter. For each state q in H_Y we substitute the transitions going from this state with a copy of T_Z as follows. The root of T_Z coincides with q, and the inner vertices are new states of the resulting automaton. Suppose that v is a leaf of T_Z, and the path from the root to v is marked by a word w. Let a be the letter of the alphabet of H_Y which is mapped to the word w in the composition. Then the image of q under the mapping defined by a is merged with v. In such a way we get a Huffman decoder with $\Theta(n_Y n_Z)$ states, where n_Y, n_Z is the number of states in H_Y and T_Z. By the definition of composition, this decoder has the same alphabet as Z. See Figure 2 for an example.

▶ **Corollary 8.** *The* Short Sync Word *problem cannot be approximated in polynomial time within a factor of $c \log n$ for some $c > 0$ for binary n-states Huffman decoders unless $P = NP$.*

Proof. We start with claim (iii) in Corollary 6 and use Theorem 7 to reduce the size of the alphabet. Thus, we reduce Short Sync Word for Huffman decoders over an alphabet of size $n^{1+\gamma}$ to Short Sync Word for binary Huffman decoders.

Assume that the size of the alphabet $k = n^{1+\gamma}$ is a power of two (if no, duplicate some letter the required number of times). We take the code $0\{0,1\}^{\log k - 1} \cup 1\{0,1\}^{\log k}$ as Z. This is a code where some words are of length $\log k$ and the other words are of length $\log k + 1$ (after minimization the star of this code is recognized by a Wielandt automaton with $\log k + 1$ states discussed in the introduction). This code has a synchronizing word of length $\Theta((\log k)^2)$ [1]. The number of vertices in the tree of this code is $\Theta(n)$.

Let ℓ be the length of a shortest synchronizing word for the original automaton. By Theorem 7, the length of a shortest synchronizing word for the result of the composition is between $\ell \log k$ and $2(\log k + 1)\ell + \Theta((\log k)^2) = \Theta((\ell + \log k) \log k)$.

For the Set Cover problem the inapproximability result holds even if we assume that the size of the optimal solution is of size at least $d \log |X|$ for some $d > 0$. Indeed, if d is a constant we can check all the subsets of C of size at most $d \log |X|$ in polynomial time. Thus, we can assume that $\ell \geq \log k$ implying $(\ell + \log k) \log k = \Theta(\ell \log k)$. Hence after the composition the length of a shortest synchronizing word is changed by at most constant multiplicative factor, and we we get a gap-reserving reduction with gap $\Theta(\log n)$. The resulting automaton is of size $\Theta(n^{2+\gamma})$, and the $(2 + \gamma)$ dependence is hidden in the constant c in the statement of this corollary. ◀

6 Partial Huffman Decoders

In this section we investigate automata recognizing the star of a non-maximal finite prefix code. Such codes have some noticeable properties which do not hold for maximal finite prefix codes. For example, there exist non-trivial non-maximal finite prefix codes with finite synchronization delay, which provides guarantees on the synchronization time [9]. This allows to read a stream of correctly transmitted compressed data from arbitrary position, which can be useful for audio and video decompression.

First we show that the known upper bounds and approximability for Short Sync Word hold true for strongly connected partial automata. Because of Proposition 3.6.5 of [6], Algorithm 1 of [26] works without any changes for strongly connected partial automata. The analysis of its approximation ratio is the same as in [14]. Thus we get the following.

▶ **Theorem 9.** *There exists a polynomial time algorithm (Algorithm 1 of [26]) finding a synchronizing word of length at most* $\frac{n^3-n}{6}$ *for a* n*-state strongly connected partial automaton. Moreover, this algorithm provides a* $O(n)$*-approximation for the* Short Sync Word *problem.*

Now we provide a lower bound on the approximability of the Short Sync Word problem for partial Huffman decoders by extending the idea used to prove inapproximability for Huffman decoders in the previous sections. First we prove the result for alphabet of size 5 and then use a composition with a maximal finite prefix code to get the same result for the binary case.

▶ **Theorem 10.** *The* Short Sync Word *problem cannot be approximated within a factor of* $n^{\frac{1}{2}-\varepsilon}$ *for every* $\varepsilon > 0$ *for* n*-state partial Huffman decoders over an alphabet of size* 5 *unless* $P = NP$*.*

Proof. First we prove inapproximability for the class of partial strongly acyclic automata, that is, automata having no simple cycles but loops in the sink state. We start with the CSP problem described in Section 3 with all the restriction defined there. Having an instance of this problem with N variables and M constraints such that each constraint is satisfied by at most K assignments, we construct an automaton A^b_ϕ over the alphabet $\{0, 1\}$. For each constraint j, we construct M identical compressed trees $T^1_j, \dots, T^M_j$ corresponding to this constraint (also described in Section 3). Then for $1 \le i \le M-1$ we merge the leaves of T^i_j corresponding to non-satisfying assignments with the root of T^{i+1}_j, and delete all the leaves corresponding to satisfying assignments (leaving all the transition leading to deleted states undefined). For each T^M_j, we again delete all the leaves corresponding to satisfying assignments and merge all the leaves corresponding to non-satisfying assignments with a new state s. This state is a self-loop, that is, $0, 1$ map s to itself. Now we define an additional letter a and M new states $r_1, \dots, r_M$. We define a to map r_j to the root of T^1_j. Finally, we add $N+1$ new states $c_0, \dots, c_N$ such that a maps c_0 to c_1, and $0, 1$ map c_i to c_{i+1} for $1 \le i \le N-1$, and map c_N to s. All other transitions are left undefined.

If a is applied first, the set S of states to be synchronized is c_1 together with the roots of T^1_j for all j. Observe that a cannot be applied anymore, since it would result in mapping all the active states of the automaton to void. If a letter other than a is applied first, a superset of S must be synchronized then.

If there exists a satisfying assignment $x_1, \dots, x_N$ then the word $ax_1 \dots x_N$ is synchronizing, since it maps all the states but c_0 to void. Otherwise, to synchronize the automaton we need to pass through M compressed trees, since each tree can map only at most M^ε states to void (since for every non-satisfiable CSP the maximum number of satisfiable constraints is

M^ε in the construction, see Section 3). Thus we get a gap of $M^{1-\varepsilon} = n^{\frac{1}{2}-\varepsilon}$ for the class of n-state strongly acyclic partial automata.

Now we are going to transfer this result to the case of partial Huffman decoders. We extend the idea of Lemma 5. All we need is to define transitions leading from s to the states having no incoming transitions (which are $r_1, \ldots, r_M$ together with c_0). The only difference is that now we have to make sure that a cannot be applied too early resulting in mapping all the states of the compressed trees to void leaving the state s active.

To do that, we introduce two new letters b_1, b_2 and perform branching as described in Lemma 5. Thus we get $M+1$ leaves of the constructed full binary tree. To each leave we attach a chain of states of length MN ending in the root of T_j^1 (or in c_0). This means that we introduce MN new states and define the letters b_1, b_2 to map a state in each chain to the next state in the same chain. This guarantees that if the letter a appears twice in a word of length at most MN, this word maps all the states of the automaton to void. Finally, the action of b_1, b_2 on the compressed trees and the states $c_0, \ldots, c_N$ repeats, for example, the action of the letter 0.

The number of states of the automaton in the construction is $O(M^{2+3\varepsilon})$ The gap is then $M^{1-2\varepsilon}$. By choosing small enough ε we thus get a gap of $n^{\frac{1}{2}-\varepsilon}$ as required. ◀

The next lemma shows that under some restrictions it is possible to reduce the alphabet of a non-maximal prefix code in a way that approximate length of a shortest synchronizing word is preserved. A word is called *non-mortal* for a prefix code X if it is a factor of some word in X^*.

▶ **Lemma 11.** *Let Y, Z be synchronizing prefix codes such that Z is finite and maximal. Let m and M be the lengths of a shortest and a longest codeword in Z. Suppose that $Y \circ_\beta Z$ is defined for some β. If there exists a synchronizing word w_Z for Z such that $\beta^{-1}(w)$ is a non-mortal word for Y, then the composition $X = Y \circ_\beta Z$ is synchronizing. Moreover, then the length of a shortest synchronizing word for X is between $m\ell$ and $M\ell + |w_Z|$, where ℓ is the length of a shortest synchronizing word for Y.*

Proof. Let $Y \subseteq \Sigma_Y^*$, $X, Z \subseteq \Sigma_Z^*$, and $\beta : \Sigma_Y \to Z$ be such that $X = Y \circ_\beta Z$. Let w_Y be a synchronizing word for Y. Then $w_Z\beta(w_Y)$ is a synchronizing word for X of length at most $M\ell + |w_Z|$. In the other direction, let w_X be a synchronizing word for X. Then $\beta^{-1}(w_X)$ is a synchronizing word for Y. Thus, $|w_X| \geq m\ell$. ◀

▶ **Corollary 12.** *The* Short Sync Word *problem cannot be approximated in polynomial time within a factor of $n^{\frac{1}{2}-\varepsilon}$ for every $\varepsilon > 0$ for binary n-state partial Huffman decoders unless $P = NP$.*

Proof. We use the composition of the automaton constructed in the proof of Theorem 10 with the prefix code $\{aaa, aab, ab, ba, bb\}$ having a synchronizing word $baab$. The word $baab$ is a concatenation of two different codewords, so their pre-images can be taken to be a and 0, resulting in a non-mortal word $a0$ for A, so we can use Lemma 11. ◀

7 Literal Huffman Decoders

In this section we deal with literal Huffman decoders. Given a finite maximal prefix code X over an alphabet Σ, the literal automaton recognizing X^* is an automaton $A = (Q, \Sigma, \delta)$ defined as follows. The states of A correspond to all proper prefixes of the words in X, and the transition function is defined as

$$\delta(q,x) = \begin{cases} qx & \text{if } qx \notin X, \\ \epsilon & \text{if } qx \in X \end{cases}$$

We will need the following useful lemma. The *rank* of a word $w \in \Sigma^*$ with respect to an automaton $A = (Q, \Sigma, \delta)$ is the size of the image of Q under the mapping defined by w.

▶ **Lemma 13** ([5, Lemma 16]). *For every n-state literal Huffman decoder over an alphabet of size k there exists a word of length $\lceil \log_k n \rceil$ and rank at most $\lceil \log_k n \rceil$.*

Note that if a n-state literal Huffman decoder has a synchronizing word of length at most $O(\log n)$, this word can be found in polynomial time by examining all words of length up to $O(\log n)$. Thus, in further algorithms we will assume that the length of a shortest synchronizing word is greater than this value. Lemma 13 stays that a word of rank at most $\lceil \log_k n \rceil$ can also be found in polynomial time.

▶ **Theorem 14.** *There exists a $O(\log n)$-approximation polynomial time algorithm for the* Short Sync Word *problem for literal Huffman decoders.*

Proof. Let $A = (Q, \Sigma, \delta)$ be a literal Huffman decoder, and $|\Sigma| = k$. Let w be a word of rank at most $\lceil \log_k n \rceil$ found as described above. Let Q' be the image of Q under the mapping defined by w, i.e. $Q' = \delta(Q, w)$. Define by v the word subsequently merging pairs of states in Q' with shortest possible words. Note than a shortest word synchronizing A has to synchronize every pair of states, in particular, one that requires a longest word. Thus the length of v is at most $\lceil \log_k n \rceil$ times greater than the length of a shortest word synchronizing A. Then the word wv is a $O(\log n)$-approximation for the Short Sync Word problem. ◀

▶ **Theorem 15.** *For every $\varepsilon > 0$, there exists a $(1+\varepsilon)$-approximation $O(n^{\log n})$-time algorithm for the problem* Short Sync Word *for n-state literal Huffman decoders.*

Proof. Let $A = (Q, \Sigma, \delta)$ be a literal Huffman decoder, and $|\Sigma| = k$. First we check all words of length at most $\frac{1}{\varepsilon}\lceil \log_k n \rceil$, whether they are synchronizing. The number of these words is polynomial, and the check can be performed in polynomial time. If a synchronizing word is found then we have an exact solution. Otherwise, a shortest synchronizing word must be longer than that and we proceed to the second stage.

Let w be a word of rank at most $\lceil \log_k n \rceil$ found as before. Now we construct the power automaton $A^{\leq \lceil \log_k n \rceil}$ restricted to all the subsets of size at most $\lceil \log_k n \rceil$. Using it, we find a shortest word synchronizing the subset $\delta(Q, w)$; let this word be v. We return wv.

Let w' be a shortest synchronizing word for $|A|$. Clearly, $|w'| \geq |v|$ and $\varepsilon|w'| > \lceil \log_k n \rceil$. Thus $|wv| \leq (1+\varepsilon)|w'|$, so wv is a $(1+\varepsilon)$-approximation as required. ◀

In view of the presented results we propose the following conjecture.

▶ **Conjecture 16.** *There exists an exact polynomial time algorithm for the* Short Sync Word *problem for literal Huffman decoders.*

Finally, we remark that it is possible to define the notion of the literal automaton of a non-maximal finite prefix code in the same way. In this case we leave undefined the transitions for a state w and a letter a such that w is a proper prefix of a codeword, but wa is neither a proper prefix of a codeword nor a codeword itself. However, the statement of Lemma 13 is false for partial automata. Indeed, consider a two-word prefix code $\{(0^n1^n)^n, (1^n0^n)^n\}$. Its literal automaton has $2n^2 - 1$ states, and a shortest synchronizing word for it is 0^{n+1} of length $n+1$. Every word of length at most n which is defined for at least one state is of the form 0^*1^* or 1^*0^* and thus has rank at least $n-1$.

8 Mortal and Avoiding Words

A word w is called *mortal* for a partial automaton A if its mapping is undefined for all the states of A. The techniques described in this paper can be easily adapted to get the same inapproximability for the Short Mortal Word problem defined as follows.

> Short Mortal Word
> *Input*: A partial automaton A with at least one undefined transition;
> *Output*: The length of a shortest mortal word for A.

This problem is connected for instance to the famous Restivo's conjecture [18].

▶ **Theorem 17.** *Unless $P = NP$, the* Short Mortal Word *problem cannot be approximated in polynomial time within a factor of*

(i) $n^{1-\varepsilon}$ *for every $\varepsilon > 0$ for n-state binary strongly connected partial automata;*

(ii) $c \log n$ *for some $c > 0$ for n-state binary partial Huffman decoders.*

Proof. It can be seen that in Theorem 1 and Corollary 8 we construct an automaton with a state s such that each state has to visit s before synchronization. Introduce a new state s' having all the transitions the same as s, and for s set the only defined transition (for an arbitrary letter) to map to s'. Thus we get an automaton such that every mortal word has to map each state to s before mapping it to nowhere. Thus we preserve all the estimations on the length of a shortest mortal word, which proves both statements. ◀

Moreover, it is easy to get a $O(\log n)$-approximation polynomial time algorithm for Short Mortal Word for literal Huffman decoders following the idea of Theorem 14. Indeed, it follows from Lemma 13 that either there exists a mortal word of length at most $\lceil \log_k n \rceil$, or there exists a word w of rank at most $\lceil \log_k n \rceil$. In the latter case we can find a word which is a concatenation of w and a shortest word mapping all this states to nowhere one by one. By the arguments similar to the proof of Theorem 14 we then get the following.

▶ **Proposition 18.** *There exists a $O(\log n)$-approximation polynomial time algorithm for the* Short Mortal Word *problem for n-state literal Huffman decoders. This algorithm always finds a mortal word of length $O(n \log n)$.*

Another connected and important problem is to find a shortest avoiding word. Given an automaton $A = (Q, \Sigma, \delta)$, a word w is called *avoiding* for a state $q \in Q$ if q is not contained in the image of Q, that is, $q \notin \delta(Q, w)$. Avoiding words play an important role in the recent improvement on the upper bound on the length of a shortest synchronizing word [24]. They are in some sense dual to synchronizing words.

> Short Avoiding word
> *Input*: An automaton A and its state q admitting a word avoiding q;
> *Output*: The length of a shortest word avoiding q in A.

If q is not the root of a literal Huffman decoder A (that is, not the state corresponding to the empty prefix), then a shortest avoiding word consists of just one letter. So avoiding is non-trivial only for the root state.

▶ **Proposition 19.** *For every $\varepsilon > 0$, there exists a $(1 + \varepsilon)$-approximation $O(n^{\log n})$-time algorithm for the problem* Short Sync Word *for n-state literal Huffman decoders.*

Proof. We use the same algorithm as in the proof of Theorem 15. The only difference is that we check whether the words are avoiding instead of synchronizing. ◀

9 Concluding Remarks

For prefix codes, a synchronizing word is usually required to map all the states to the root [6]. One can see that this property holds for all the constructions of the paper. Moreover, in all the constructions the length of a shortest synchronizing word is linear in the number of states of the automaton. Thus, if we restrict to this case, we still get the same inapproximability results. Also, it should be noted that all the inapproximability results are proved by providing a gap-preserving reduction, thus proving NP-hardness of approximating the Short Sync Word problem within a given factor.

References

1 Dmitry S. Ananichev, Vladimir V. Gusev, and Mikhail V. Volkov. Slowly synchronizing automata and digraphs. In *Mathematical Foundations of Computer Science, LNCS vol. 6281*, pages 55–65. Springer, 2010.

2 Marie-Pierre Béal, Mikhail V. Berlinkov, and Dominique Perrin. A quadratic upper bound on the size of a synchronizing word in one-cluster automata. *International Journal of Foundations of Computer Science*, 22(2):277–288, 2011.

3 Mikhail V. Berlinkov. Approximating the minimum length of synchronizing words is hard. *Theory of Computing Systems*, 54(2):211–223, 2014.

4 Mikhail V. Berlinkov. On two algorithmic problems about synchronizing automata. In Arseny M. Shur and Mikhail V. Volkov, editors, *DLT 2014. LNCS, vol. 8633*, pages 61–67. Springer, Cham, 2014.

5 Mikhail V. Berlinkov and Marek Szykuła. Algebraic synchronization criterion and computing reset words. *Information Sciences*, 369:718–730, 2016.

6 Jean Berstel, Dominique Perrin, and Christophe Reutenauer. *Codes and Automata*. Encyclopedia of Mathematics and its Applications 129. Cambridge University Press, 2010.

7 Marek Biskup. *Error Resilience in Compressed Data – Selected Topics*. PhD thesis, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, 2008.

8 Marek Tomasz Biskup and Wojciech Plandowski. Shortest synchronizing strings for huffman codes. *Theoretical Computer Science*, 410(38):3925–3941, 2009.

9 Véronique Bruyère. On maximal codes with bounded synchronization delay. *Theoretical Computer Science*, 204(1):11–28, 1998.

10 Renato M. Capocelli, A. A. De Santis, Luisa Gargano, and Ugo Vaccaro. On the construction of statistically synchronizable codes. *IEEE Transactions on Information Theory*, 38(2):407–414, 1992.

11 David Eppstein. Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19(3):500–510, 1990.

12 Christopher F Freiling, Douglas S Jungreis, François Théberge, and Kenneth Zeger. Almost all complete binary prefix codes have a self-synchronizing string. *IEEE Transactions on Information Theory*, 49(9):2219–2225, 2003.

13 Paweł Gawrychowski and Damian Straszak. Strong inapproximability of the shortest reset word. In F. Giuseppe Italiano, Giovanni Pighizzini, and T. Donald Sannella, editors, *MFCS 2015. LNCS, vol. 9234*, pages 243–255. Springer, Heidelberg, 2015.

14 Michael Gerbush and Brent Heeringa. *Approximating Minimum Reset Sequences*, pages 154–162. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

15 David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

16 Pavel Martyugin. Complexity of problems concerning reset words for cyclic and eulerian automata. *Theoretical Computer Science*, 450(Supplement C):3–9, 2012. Implementation and Application of Automata (CIAA 2011).

17 Jean-Eric Pin. On two combinatorial problems arising from automata theory. In C. Berge, D. Bresson, P. Camion, J.F. Maurras, and F. Sterboul, editors, *Combinatorial Mathematics Proceedings of the International Colloquium on Graph Theory and Combinatorics*, volume 75 of *North-Holland Mathematics Studies*, pages 535–548. North-Holland, 1983.

18 Antonio Restivo. Some remarks on complete subsets of a free monoid. *Quaderni de "La ricerca scientifica", CNR Roma*, 109:19–25, 1981.

19 Andrew Ryzhikov. Synchronization problems in automata without non-trivial cycles. In Arnaud Carayol and Cyril Nicaud, editors, *CIAA 2017, LNCS, vol. 10329*, pages 188–200. Springer, Cham, 2017.

20 Andrew Ryzhikov and Anton Shemyakov. Subset synchronization in monotonic automata. In Juhani Karhumäki and Aleksi Saarela, editors, *Proceedings of the Fourth Russian Finnish Symposium on Discrete Mathematics, TUCS Lecture Notes 26*, pages 154–164, 2017. Accepted to Fundamenta Informaticae.

21 Marcel-Paul Schützenberger. On the synchronizing properties of certain prefix codes. *Information and Control*, 7(1):23–36, 1964.

22 Marcel-Paul Schützenberger. On synchronizing prefix codes. *Information and Control*, 11(4):396–401, 1967.

23 Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 3rd edition, 2012.

24 Marek Szykuła. Improving the Upper Bound on the Length of the Shortest Reset Word. In *STACS 2018*, LIPIcs, pages 56:1–56:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

25 Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, 2001.

26 Mikhail V. Volkov. Synchronizing automata and the Černý conjecture. In Carlos Martín-Vide, Friedrich Otto, and Henning Fernau, editors, *LATA 2008. LNCS, vol. 5196*, pages 11–27. Springer, Heidelberg, 2008.

27 Vojtěch Vorel. Complexity of a problem concerning reset words for eulerian binary automata. *Information and Computation*, 253(Part 3):497–509, 2017. LATA 2014.

Quantum vs. Classical Proofs and Subset Verification

Bill Fefferman
Department of EECS, University of California at Berkeley, Berkeley, CA and
NIST, Gaithersburg, MD, USA
wjf@berkeley.edu

Shelby Kimmel
Department of Computer Science, Middlebury College, Middlebury, VT, USA
skimmel@middlebury.edu

Abstract

We study the ability of efficient quantum verifiers to decide properties of exponentially large subsets given either a classical or quantum witness. We develop a general framework that can be used to prove that QCMA machines, with only classical witnesses, cannot verify certain properties of subsets given implicitly via an oracle. We use this framework to prove an oracle separation between QCMA and QMA using an "in-place" permutation oracle, making the first progress on this question since Aaronson and Kuperberg in 2007 [3]. We also use the framework to prove a particularly simple standard oracle separation between QCMA and AM.

2012 ACM Subject Classification Theory of computation → Quantum complexity theory, Theory of computation → Complexity classes

Keywords and phrases Quantum Complexity Theory, Quantum Proofs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.22

Acknowledgements We are grateful for multiple discussions with Stephen Jordan regarding permutation oracle verification strategies. We appreciate the many people who discussed this project with us, including Scott Aaronson, David Gosset, Gus Gutoski, Yi-Kai Liu, Ronald de Wolf, Robin Kothari, Dvir Kafri, and Chris Umans. SK completed much of this work while at the Joint Center for Quantum Information and Computer Science (QuICS), University of Maryland.

1 Introduction

How much computational power does an efficient quantum verifier gain when given a polynomial sized quantum state to support the validity of a mathematical claim? In particular, is there a problem that can be solved in this model, that cannot be solved if the verifier were instead given a classical bitstring? This question, the so-called QMA vs. QCMA problem, is fundamental in quantum complexity theory. To complexity theorists, the question can be motivated simply by trying to understand the power of quantum nondeterminism, where both QMA and QCMA can be seen as "quantum analogues" of NP. More physically, QMA is characterized by the k-local Hamiltonian problem, in which we must decide if the ground state energy of a local Hamiltonian is above or below a specified threshold [16, 5]. In this setting, the QMA vs. QCMA question asks whether there exists a purely *classical* description of the ground state that allows us to make this decision. For instance, if the ground state of any local Hamiltonian can be prepared by an efficient quantum circuit, then QMA = QCMA, as the classical witness for the k-local Hamiltonian problem would be the

classical description of this quantum circuit. It was this intuition that caused Aharonov and Naveh to conjecture that these classes are equal, in the paper that first defined QCMA [5].

It was recently established [12] that the witness to a QMA machine may always be replaced by a subset state, where a subset state on n qubits has the form $|S\rangle = 1/\sqrt{|S|}\sum_{i\in S}|i\rangle$ for some subset $S \subset [2^n]$. However, it seems difficult to create a classical witness on n bits that captures the information in a subset state $|S\rangle$. Therefore, problems involving subsets seem like ripe ground for understanding the QMA vs. QCMA problem. We investigate the following question: under what circumstances is it possible for a quantum machine to verify properties of a subset? This question is not answered by [12]; they study general properties of languages that are in QMA and QCMA, while we attempt to prove specific languages of interest (that are related to subsets) are either in or not in QMA or QCMA.

In the hopes of further exploring these questions, we exhibit a general framework that can be used to obtain oracle separations against QCMA for subset-based problems. We use this framework to prove the existence of an "in-place" permutation oracle $\boldsymbol{\mathcal{P}}$ (a unitary that permutes standard basis states within a single register) [10, 2] for which $\mathsf{QMA}^{\boldsymbol{\mathcal{P}}} \not\subseteq \mathsf{QCMA}^{\boldsymbol{\mathcal{P}}}$, making the first progress on this problem since Aaronson and Kuperberg in 2007 [3], who attained a "quantum oracle" separation (i.e., a separation relative to an *arbitrary* black-box unitary transformation acting on a polynomial number of qubits). In this problem, for the case of QMA, the in-place permutation oracle allows us to verify that the given witness is indeed the correct subset state. On the other hand, our framework allows us to prove the language is not in QCMA. Our framework is quite general, and we are also able to use it to establish a particularly simple example of a (conventional) oracle $\boldsymbol{\mathcal{O}}$ so that $\mathsf{AM}^{\boldsymbol{\mathcal{O}}} \not\subseteq \mathsf{QCMA}^{\boldsymbol{\mathcal{O}}}$.[1]

1.1 Subset-Verifying Oracle Problems

We consider two oracle problems related to verifying properties of subsets. In *Subset Size Checking*, we are given a black box function $f : [N^2] \to \{0, 1\}$, that marks elements with either a 0 or 1. We are promised that the number of marked items is either N or $0.99N$, and we would like to decide which is the case. We want to verify the size of the subset marked by f.

In the other oracle problem, *Preimage Checking*, we are given a black box permutation on N^2 elements. We are promised that the preimage of the first N elements under the permutation is either mostly even or mostly odd, and we would like to decide which is the case. In this problem, we want to verify this parity property of a subset of the preimage of the function.

Subset Size Checking is in AM [11], and we give a procedure that proves *Preimage Checking* is in QMA when the permutation is given as an in-place quantum oracle. An in-place permutation unitary $\mathcal{P}_\sigma$ acts as $\mathcal{P}_\sigma|i\rangle = |\sigma(i)\rangle$ for a permutation σ. For *Preimage Checking*, we are interested in the set $S_{\mathrm{pre}}(\sigma) = \{i : \sigma(i) \in [N]\}$. Given the subset state $|S_{\mathrm{pre}}(\sigma)\rangle$, it is easy to verify that the correct state was sent, because $\mathcal{P}_\sigma|S_{\mathrm{pre}}(\sigma)\rangle = |[N]\rangle$, which is easy to verify using a measurement in the Hadamard basis.

However, we do not expect subset-based oracle languages like *Subset Size Checking* and *Preimage Checking* to be in QCMA because the classical witness does not have enough information to identify the relevant subset. We make this intuition more precise by providing a general recipe for proving that subset-verifying oracle languages are not in QCMA. We

[1] Note there was previously an example of an oracle separating AM from PP [20]. Since QMA $\subseteq$ PP [18], this is formally a stronger result. Nonetheless, our oracle is substantially different, and uses completely different ideas.

apply this procedure to show that both *Preimage Checking* (with a randomized in-place oracle - see Section 3 for more details) and *Subset Size Checking* are not in QCMA. The procedure involves familiar tools, like the adversary bound [6] (although adapted to our in-place oracle when necessary), as well as a new tool, the *Fixing Procedure*, which finds subsets with nice structure within a large arbitrary set. We now sketch the recipe:

1. We show that for every QCMA machine, there are more valid oracles than possible classical witnesses, so by a counting argument, there must be one classical witness w^* that corresponds to a large number of potential oracles. We then restrict ourselves to considering oracles that correspond to w^*.
2. Because we are considering subset-verifying problems, if we have a collection of black box functions that corresponds to w^*, we immediately have some set of subsets that corresponds to w^*. At this point, we know nothing about this set of subsets except its size, thanks to the counting argument. We next show (using the *Fixing Procedure*) that if we have a set of subsets of a certain size, we can always find a subset of the original set that has nice structure.
3. We apply the adversary bound to the subset with nice structure to show that the number of quantum queries needed to distinguish between YES and NO cases is exponential.
4. We finally put these pieces together in a standard diagonalization argument.

1.2 Technical Contributions

Our adversary bound for in-place permutation oracles provides a query lower-bounding technique for unitary oracles when access is *not* given to the inverse of the oracle. (While Belovs [9] created an adversary bound for arbitrary unitaries, his results assume access to an inverse.) While we typically assume quantum oracles include access to an inverse or are self-inverting, in open quantum systems it is natural to not have an inverse.

When proving that *Preimage Checking* is not in QCMA, we use an oracle that is not unitary. The oracle is a completely-positive trace-preserving (CPTP) map that at each application applies one unitary chosen uniformly at random from among a set of unitaries. Standard lower bounding techniques fail for such an oracle. The closest result is from Regev and Schiff [19] who give a lower bound on solving Grover's problem with an oracle that produces errors. Regev and Schiff deal with the non-unitarity of the map by modeling the state of the system using pure states. This strategy does not work in our case. Instead, we use the fact that every non-unitary CPTP map can be implemented as a unitary on a larger system. In our case, we simulate our random oracle using a unitary black box oracle acting on subsystem B, followed by a fixed unitary that entangles subsystems A and B. The entangling operation can not be efficiently implemented, but as we are bounding query complexity, this is acceptable. This technique may be of use for similar problems; for example, we do not know the query complexity of solving Grover's problem with an oracle that produces a depolarizing error with each application. A depolarizing map is similar to our CPTP map in that both maps can be thought of as applying a unitary at random from among a set of unitaries, and so perhaps this approach will stimulate new approaches for the Grover problem.

1.3 Impact and Directions for Future Research

While Aaronson and Kuperberg have previously proved an oracle separation between QMA and QCMA [3], their oracle seems to be especially quantum, as it is defined by a Haar random quantum state. Our in-place oracle has more of a classical feel, in that it encodes

a classical permutation function. However, it is still not a standard quantum oracle, as it is not self-inverting. Is there a standard (i.e. not in-place) oracle language that separates QMA and QCMA? Although we can only prove a separation when our in-place oracle also has randomness, we believe our techniques could be adapted to prove a similar result but without oracle randomness. While we give a recipe for showing certain subset-based problems are not in QCMA, we believe some of these problems are also not in QMA; for example, is it possible to prove *Subset Size Checking* is not in QMA?

Our contributions to techniques for lower bounding query complexity for non-standard oracles raise several questions. Is there a general adversary bound [14, 17] for in-place permutation oracles? There are examples of problems for which in-place permutation oracles require exponentially fewer queries that standard permutation oracles e.g., [7]. We conjecture that there are also examples of problems for which standard permutation oracles require exponentially fewer queries than in-place oracles. In fact, we do not believe it is possible to obtain a Grover-type speed-up with an in-place oracle; we believe the problem of determining the inverse of an element of an in-place permutation oracle requires N queries for a permutation on N elements. However, in order to prove these results, we suspect one needs a more powerful tool, like a general adversary bound for in-place oracles.

1.4 Organization

The rest of the paper is organized as follows, in Section 2, we introduce notation that will be used throughout the rest of the paper, and define QMA and QCMA. In Section 3, we define and discuss standard, in-place, and randomized in-place quantum permutations, as well as state an adversary lower bound for in-place permutation unitaries. In Section 4, we describe the *Preimage Checking* Problem and prove it is in QMA. In Section 5, we lay out the general recipe for proving subset-based languages are not in QCMA. In Section 6, we apply this procedure to the *Subset Size Checking* problem, and use it to prove an oracle separation between AM and QCMA. Finally, in Section 7, we apply the procedure to *Preimage Checking* and show this problem is not in QCMA.

2 Definitions and Notation

We use the notation $[M] = \{1, 2, \ldots, M\}$. σ will refer to a permutation. Unless otherwise specified, the sets we use, generally denoted S, will be a set of positive integers. Also, we will use bold type-face to denote a set of sets. For instance, $\mathbf{S}$ will refer to a set of sets of positive integers. To make our notation clearer, we will refer to such a set of sets as a *set family*. Likewise, we denote $\boldsymbol{\sigma}$ to be a set of permutations acting on the same set of elements.

For S a set of positive integers, a subset state $|S\rangle$ is $|S\rangle = \frac{1}{\sqrt{|S|}} \sum_{i \in S} |i\rangle$.

Throughout, we use $N = 2^n$. All logarithms are in base 2. We use $\boldsymbol{\sigma}^n$ to be the set of permutations acting on N^2 elements. That is, if $\sigma \in \boldsymbol{\sigma}^n$, $\sigma : [N^2] \to [N^2]$. For positive integers $i > j$, let $\mathbf{C}(i, j)$ be the set family containing j elements of $[i]$: $\mathbf{C}(i, j) = \{S \subset [i] : |S| = j\}$.

We use calligraphic font $\mathcal{P}, \mathcal{U}$ to denote unitary operations. We use elaborated calligraphic font $\mathscr{P}$, $\mathscr{U}$ to denote CPTP maps. For a unitary CPTP map $\mathscr{U}$ acting on a density matrix ρ, we have $\mathscr{U}(\rho) = \mathcal{U}\rho\mathcal{U}^\dagger$, where $(\cdot)^\dagger$ denotes conjugate transpose. We will use $\mathcal{O}$ to denote a unitary oracle, and $\mathscr{O}$ to denote a CPTP map oracle.

We include the following standard definition for completeness (e.g., see also [1, 3]).

▶ **Definition 1.** QMA is the set of promise problems $A = (A_{Yes}, A_{No})$ so that there exists an efficient quantum verifier V_A and a polynomial $p(\cdot)$:

1. *Completeness*: For all $x \in A_{Yes}$ there exists a $p(|x|)$-qubit pure quantum state $|\psi\rangle$ so that $\Pr[V_A(x, |\psi\rangle) = 1] \geq 2/3$
2. *Soundness*: For all $x \in A_{No}$ and any pure quantum state $|\psi\rangle$, $\Pr[V_A(x, |\psi\rangle) = 1] \leq 1/3$.

QCMA is the same class, with the witness $|\psi\rangle$ replaced by a polynomial length classical bitstring.

3 Permutation Maps

3.1 Permutations as Oracles: In-Place Permutation vs. Standard Permutation

Black box permutation unitaries have been considered previously, most notably in the collision and element distinctness problems [2, 4]. However, the unitaries considered in these works were standard oracles. A standard oracle implements the permutation $\sigma \in \boldsymbol{\sigma}^n$ as $\mathcal{P}_\sigma^{\text{stand}}|i\rangle|b\rangle = |i\rangle|b \oplus \sigma(i)\rangle$ for $i, b \in [N^2]$, where $|i\rangle$ for $i \in [N^2]$ are standard basis states and $\oplus$ denotes bitwise XOR. Note that $(\mathcal{P}_\sigma^{\text{stand}})^2 = \mathbb{I}_{N^4}$; that is, acting with the unitary twice produces the $N^4 \times N^4$ identity operation.

We consider in-place permutation unitaries, which implement the permutation $\sigma \in \boldsymbol{\sigma}^n$ as $\mathcal{P}_\sigma|i\rangle = |\sigma(i)\rangle$. In general $(\mathcal{P}_\sigma)^2 \neq \mathbb{I}_{N^2}$. Crucially, given black box access to $\mathcal{P}_\sigma$, *we do not give black box access to its inverse.* In fact, in Section 3.2, we show that given only $\mathcal{P}_\sigma$, it is hard to invert its action. Non-self-inverting permutation unitaries have been considered previously, in [10, 2].

We believe standard and in-place permutation unitaries are of incomparable computational power. That is, given one type that implements σ, you can not efficiently simulate the other type implementing the permutation σ. For example, if we have the state $\sum_{y \in S} |y\rangle|\sigma(y)\rangle$ (normalization omitted), we can create the state $\sum_{y \in S} |y\rangle|0\rangle$ with a single query to $\mathcal{P}_\sigma^{\text{stand}}$. However, if we only have access to the in-place permutation $\mathcal{P}_\sigma$ and not to $\mathcal{P}_{(\sigma)^{-1}} = (\mathcal{P}_\sigma)^{-1}$, it seems difficult to create this state.

On the other hand, suppose we want to prepare the state $\sum_{y \in [N]} |\sigma(y)\rangle$ (normalization omitted). We can create this state in one query to the in-place permutation oracle $\mathcal{P}_\sigma$ by applying the oracle to the uniform superposition $\sum_{y \in [N]} |y\rangle$. In the standard model, this problem is called "index erasure," and requires an exponential number of queries in n to $\mathcal{P}_\sigma^{\text{stand}}$ [7].

3.2 An Adversary Bound for In-Place Permutation Oracles

In Appendix A, we prove a non-weighted adversary bound for in-place permutations oracles that is identical to what Ambainis proves in Theorem 6 in [6] for standard permutation oracles.

▶ **Lemma 2.** *Let $\boldsymbol{\sigma} \subset [V] \to [V]$ be a subset of permutations acting on the elements $[V]$. Let $f : \boldsymbol{\sigma} \to \{0, 1\}$ be a function of permutations. Let $\boldsymbol{\sigma}_X \subset \boldsymbol{\sigma}$ be a set of permutations such that if $\sigma \in \boldsymbol{\sigma}_X$, then $f(\sigma) = 1$. Let $\boldsymbol{\sigma}_Y \subset \boldsymbol{\sigma}$ be a permutation family such that if $\sigma \in \boldsymbol{\sigma}_Y$ then $f(\sigma) = 0$. Let $R \subset \boldsymbol{\sigma}_X \times \boldsymbol{\sigma}_Y$ be such that*

- *For every $\sigma_x \in \boldsymbol{\sigma}_X$, there exists at least m different $\sigma_y \in \boldsymbol{\sigma}_Y$ such that $(\sigma_x, \sigma_y) \in R$.*
- *For every $\sigma_y \in \boldsymbol{\sigma}_Y$, there exists at least m' different $\sigma_x \in \boldsymbol{\sigma}_X$ such that $(\sigma_x, \sigma_y) \in R$.*

- *Let $l_{x,i}$ be the number of $\sigma_y \in \boldsymbol{\sigma}_Y$ such that $(\sigma_x, \sigma_y) \in R$ and $\sigma_x(i) \neq \sigma_y(i)$. Let $l_{y,i}$ be the number of $\sigma_x \in \boldsymbol{\sigma}_Y$ such that $(\sigma_x, \sigma_y) \in R$ and $\sigma_x(i) \neq \sigma_y(i)$. Then let $l_{max} = \max_{(\sigma_x,\sigma_y)\in R,i} l_{x,i} l_{y,i}$.*

Then given an in-place permutation oracle $\mathcal{P}_\sigma$ for $\sigma \in \boldsymbol{\sigma}$ that acts as $\mathcal{P}_\sigma|i\rangle = |\sigma(i)\rangle$, any quantum algorithm that correctly evaluates $f(\sigma)$ with probability $1-\epsilon$ for every element of $\boldsymbol{\sigma}_X$ and $\boldsymbol{\sigma}_Y$ must use $\left(1 - 2\sqrt{\epsilon(1-\epsilon)}\right)\sqrt{\frac{mm'}{l_{max}}}$ queries to the oracle.

As a corollary of Lemma 2, (using exactly the same technique as Theorem 7 in [6]), we have that the query complexity of inverting an in-place permutation oracle on V elements is $\Omega(V^{1/2})$.

3.3 Permutations with Randomness

Additionally, we consider in-place permutation oracles with internal randomness that are CPTP (completely-positive trace-preserving) maps, rather than unitaries. Oracles with internal randomness were shown to cause a complete loss of quantum speed-up in [19], while in [13], such oracles were shown to give an infinite quantum speed-up.

We consider oracles that apply an in-place permutation at random from among a family of possible permutations. Let $\boldsymbol{\sigma} \subseteq \boldsymbol{\sigma}^n$ be a set of $|\boldsymbol{\sigma}|$ permutations. Then the CPTP map $\mathscr{P}_{\boldsymbol{\sigma}}$ acts as follows:

$$\mathscr{P}_{\boldsymbol{\sigma}}(\rho) = \frac{1}{|\boldsymbol{\sigma}|}\sum_{\sigma\in\boldsymbol{\sigma}} \mathcal{P}_\sigma \rho \mathcal{P}_\sigma^\dagger. \tag{1}$$

4 Pre-Image Checking

In this section, we define a property of oracle families which we call *randomized-preimage-correct*, and construct a decision language based on such oracles that is in QMA. Essentially, the problem is to decide whether the preimage of the first N elements of a permutation is mostly even or odd.

Given a permutation $\sigma \in \boldsymbol{\sigma}^n$, we associate a preimage subset $S_{\text{pre}}(\sigma)$ to that permutation ("pre" is for "preimage"), where $S_{\text{pre}}(\sigma) = \{j : \sigma(j) \in [N]\}$. That is, $S_{\text{pre}}(\sigma)$ is the subset of elements in $[N^2]$ whose image under σ is in $[N]$. Additionally, to each subset $S \subseteq [N^2]$ with $|S| = N$, we associate a set of permutations $\boldsymbol{\sigma}_{\text{pre}}(S)$, where $\boldsymbol{\sigma}_{\text{pre}}(S) = \{\sigma : \sigma \in \boldsymbol{\sigma}^n, S_{\text{pre}}(\sigma) = S\}$. Let

$$\begin{aligned} \boldsymbol{S}^n_{\text{even}} &= \{S : S \subset [N^2], |S| = N, |S \cap \mathbb{Z}_{\text{even}}| = 2/3N\} \\ \boldsymbol{S}^n_{\text{odd}} &= \{S : S \subset [N^2], |S| = N, |S \cap \mathbb{Z}_{\text{odd}}| = 2/3N\}. \end{aligned} \tag{2}$$

▶ **Definition 3** (randomized-preimage-correct oracles)**.** Let $\boldsymbol{\mathcal{O}}$ be a countably infinite set of quantum operators (CPTP maps): $\boldsymbol{\mathcal{O}} = \{\mathscr{O}_1, \mathscr{O}_2, \dots\}$, where each $\mathscr{O}_n$ implements an operation on $(2n)$-qubits. We say that $\boldsymbol{\mathcal{O}}$ is randomized-preimage-correct if for every n, $\mathscr{O}_n = \mathscr{P}_{\boldsymbol{\sigma}_{pre}(S)}$, with $S \in \boldsymbol{S}^n_{\text{even}} \cup \boldsymbol{S}^n_{\text{odd}}$.

▶ **Theorem 4.** *For any randomized-preimage-correct $\boldsymbol{\mathcal{O}}$, the unary language $L_{\boldsymbol{\mathcal{O}}}$, which contains those unary strings 1^n such that $\mathscr{O}_n = \mathscr{P}_{\boldsymbol{\sigma}_{\text{pre}}(S)}$ with $S \in \boldsymbol{S}^n_{even}$, is in $\mathsf{QMA}^{\boldsymbol{\mathcal{O}}}$.*

Proof. We first prove completeness. We assume $1^n \in L_{\boldsymbol{\mathcal{O}}}$, so $\mathscr{O}_n = \mathscr{P}_{\boldsymbol{\sigma}_{\text{pre}}(S)}$ for some $S \in \boldsymbol{S}^n_{\text{even}}$. We consider using as a witness the subset state $|S\rangle$ on $2n$ qubits. We analyze the following verifier: with probability $1/2$, do either

Test (i) Apply $\mathscr{P}_{\boldsymbol{\sigma}_{\mathrm{pre}}(S)}$ to $|S\rangle$, and measure whether the resultant state is $|[N]\rangle$. This measurement can be done by applying $H^{\otimes n}$ to the first n qubits, and then measuring all qubits in the standard basis. If the outcome is 0, output 1; otherwise, output 0.

Test (ii) Measure $|S\rangle$ in the standard basis. Let i^* be the resulting standard basis state. If i^* is odd, output 0. Otherwise, apply $\mathscr{P}_{\boldsymbol{\sigma}_{\mathrm{pre}}(S)}$ to $|i^*\rangle$ and measure the resultant standard basis state. If the resultant state is not in $[N]$, output 0; otherwise, output 1.

If Test (i) is implemented, the verifier always outputs 1 because all the permutations that might be applied by $\mathscr{P}_{\boldsymbol{\sigma}_{\mathrm{pre}}(S)}$ transform $|S\rangle$ into $|[N]\rangle$. If Test (ii) is implemented, the verifier outputs 1 with probability 2/3. Averaging over both Tests, the verifier outputs 1 with probability 5/6.

Now we show soundness. Let $1^n \notin L_{\boldsymbol{\mathcal{O}}}$, so $\mathscr{O}_n = \mathscr{P}_{\boldsymbol{\sigma}_{\mathrm{pre}}(S)}$ for some $S \in \mathbf{S}^n_{\mathrm{odd}}$. Without loss of generality, let the witness be the $2n$-qubit state $|\psi(S)\rangle = \sum_{i=1}^{N^2} \beta_i |i\rangle$. If $p_{(i)}$ (resp. $p_{(ii)}$) is the probability the verifier outputs 1 after performing Test (i) (resp. Test (ii)), then we have

$$p_{(i)} = \frac{1}{N}\left|\sum_{i\in S}\beta_i\right|^2, \qquad p_{(ii)} = \sum_{i\in\mathbb{Z}_{\mathrm{even}}\cap S}|\beta_i|^2. \tag{3}$$

regardless of which permutation the map $\mathscr{P}_{\boldsymbol{\sigma}_{\mathrm{pre}}(S)}$ applies.

Using Cauchy-Schwarz and the triangle inequality, we have $1 \geq \left(\sqrt{3p_{(i)}} + (\sqrt{2}-1)p_{(ii)}\right)/\sqrt{2}$. Thus the total probability that the verifier outputs 1 is

$$\frac{1}{2}\left(p_{(i)} + p_{(ii)}\right) \leq \frac{1}{2}\left(\frac{2}{3}\left(1 - \frac{\sqrt{2}-1}{\sqrt{2}}p_{(ii)}\right)^2 + p_{(ii)}\right). \tag{4}$$

The derivative of the right hand side is positive for $0 \leq p_{(ii)} \leq 1$, so to maximize the right hand side we take $p_{(ii)} = 1$. Doing this, we find the probability that the verifier outputs 1 is at most 2/3. ◀

We will show that the Preimage Checking problem is not in QCMA in Section 7.

Our proof that the Preimage Checking problem is in QMA works equally well for an in-place oracle without randomness. We use the randomness in our oracle in the proof that randomized-preimage-correct languages can not be decided by QCMA. We believe the separation holds even without randomness in the oracle.

5 Strategy for Proving Subset-Based Oracle Languages are not in QCMA

In this section, we describe a general strategy for showing that certain oracle languages are not in QCMA. In particular, we consider the case when the oracles are related to sets of integers:

▶ **Definition 5** (Subset-Based Oracle). Let $\boldsymbol{\mathcal{O}} = \{\mathscr{O}_1, \mathscr{O}_2, \dots\}$ be an oracle such that each $\mathscr{O}_n$ implements a $p_1(n)$-qubit CPTP map from some set of maps $\boldsymbol{\mathcal{O}}^n$. Then we say $\boldsymbol{\mathcal{O}}$ is a subset-based oracle if there exists a set of bijective functions $\{g^1, g^2, \dots\}$ with $g^n : \boldsymbol{\mathcal{O}}^n \to \mathbf{S}^n$ where $\mathbf{S}^n$ is the union of disjoint subset families $\mathbf{S}^n_X$ and $\mathbf{S}^n_Y$.

We also use the following definition:

▶ **Definition 6.** Given a subset family $\mathbf{S}$ containing subsets of positive integers, and $\beta \in \mathbb{R}$ such that $\beta > 0$, we say $\mathbf{S}$ is β-distributed if:

(1) There exists a (possibly empty) set S_{fixed} such that $S_{fixed} \subset S$ for all $S \in \mathbf{S}$.

(2) For every element $i \in \left(\bigcup_{S\in\mathbf{S}} S\right) \setminus S_{fixed}$, i appears in at most a $2^{-\beta}$-fraction of $S \in \mathbf{S}$.

We call S_{fixed} the "fixed subset" of $\mathbf{S}$.

We use the following Recipe for proving a subset-based oracle language is not in QCMA:

▶ **Recipe 1.**

Set-up. *Fix some enumeration over all poly(n)-size quantum verifiers $M_1, M_2, ...$, which we can do because the number of such machines is countably infinite (by the Solovay-Kitaev theorem [15]). Some of these verifiers may try to decide a language by trivially "hardwiring" its outputs; for example, by returning 1 independent of the input. We start by fixing a unary language L such that no machine M_i hardwires the language. We can always do this because there are more languages than poly(n)-sized machines. Then our goal is to associate a subset-based oracle $\boldsymbol{\mathcal{O}} = \{\mathscr{O}_1, \mathscr{O}_2, \dots\}$ with L, such that $1^n \in L$ if and only if $g^n(\mathscr{O}_n) \in \mathbf{S}_X^n$, and to show that even with access to $\boldsymbol{\mathcal{O}}$, no M_i can efficiently decide L for all n.*

Consider the QCMA *machine M_i, and suppose it is given access to a subset-based oracle $\boldsymbol{\mathcal{O}}$, as well as a witness of $p_{M_i}(n)$ bits for each input 1^n. Then for each $\mathscr{O}_n \in \boldsymbol{\mathcal{O}}$ there is some subset of integers $S \in \mathbf{S}^n$ such that $g^n(\mathscr{O}_n) = S$. Since g^n is bijective, S uniquely defines $\mathscr{O}_n$, so the optimal witness that causes M_i to accept $\mathscr{O}_n$ can be thought of as a function of S. Let $w_i(S)$ be the witness that gives the highest probability of success in convincing M_i that $S \in \mathbf{S}_X^n$. Then we denote $\mathbf{S}_{i,wit}(w) = \{S : S \in \mathbf{S}_X^n, w = w_i(S)\}$.*

Using the pigeonhole principle, there exists some string $w_{i,n}$ of $p_{M_i}(n)$ bits such that

$$|\mathbf{S}_{i,wit}(w_{i,n})| \geq \frac{1}{2^{p_{M_i}(n)}} |\mathbf{S}_X^n| . \tag{5}$$

That is, there exists a witness such that a large number of subsets correspond to that witness.

1. *Prove that for $n \geq n_i^*$, there is a subset family $\mathbf{S}_X \subseteq \mathbf{S}_{i,wit}(w_{i,n})$ that is α-distributed with fixed subset S_{fixed}. Let $\mathbf{S}_Y = \{S : S \in \mathbf{S}_Y^n, S_{fixed} \subset S\}$. Show the cardinality of $\mathbf{S}_Y$ is large.*
2. *Create a relation $\mathbf{R} \subseteq \{\mathscr{O} : \mathscr{O} \in \boldsymbol{\mathcal{O}}^n, g(\mathscr{O}) \in \mathbf{S}_X\} \times \{\mathscr{O} : \mathscr{O} \in \boldsymbol{\mathcal{O}}^n, g(\mathscr{O}) \in \mathbf{S}_Y\}$ and use $\mathbf{R}$ to apply an adversary bound to prove a lower bound of $\Omega(N^{\alpha/2}) = \Omega(2^{n\alpha/2})$ on the number of queries M_i requires to distinguish some oracle $\mathscr{O}_{x,n,i} \in \boldsymbol{\mathcal{O}}^n$ such that $g^n(\mathscr{O}_{x,n,i}) \in \mathbf{S}_X^n$ from an oracle $\mathscr{O}_{y,n,i} \in \boldsymbol{\mathcal{O}}^n$ such that $g^n(\mathscr{O}_{y,n,i}) \in \mathbf{S}_Y^n$.*
3. *Apply a standard Baker-Gill-Solovay diagonalization argument [8] to complete the proof. That is, for each M_i, choose a unique $n_i \geq n_i^*$, and if $1^{n_i} \in L$, set $\mathscr{O}_{n_i} = \mathscr{O}_{x,n_i,i}$ and if $1^{n_i} \notin L$, set $\mathscr{O}_{n_i} = \mathscr{O}_{y,n_i,i}$. Then no* QCMA *machine can efficiently decide the language.*

6 Subset Size Checking

In this section, we create a subset-based oracle language $L_{\boldsymbol{\mathcal{O}}}$, such that $L_{\boldsymbol{\mathcal{O}}} \in \mathsf{AM}^{\boldsymbol{\mathcal{O}}}$, but $L_{\boldsymbol{\mathcal{O}}} \notin \mathsf{QCMA}^{\boldsymbol{\mathcal{O}}}$. We use the strategy of Section 5 to prove $L_{\boldsymbol{\mathcal{O}}} \notin \mathsf{QCMA}^{\boldsymbol{\mathcal{O}}}$.

Let f_S be a function that marks a subset $S \subset [N^2]$. That is $f_S : \{0,1\}^{2n} \to \{0,1\}$, such that $f_S(i) = 1$ if $i \in S$ and 0 otherwise. Let $\mathcal{F}_S$ be the unitary such that $\mathcal{F}_S|i\rangle = (-1)^{f_S(i)}|i\rangle$.

▶ **Definition 7.** Let $\boldsymbol{\mathcal{O}}$ be a countably infinite set of unitaries (resp. boolean functions): $\boldsymbol{\mathcal{O}} = \{\mathcal{O}_1, \mathcal{O}_2, ...\}$, where $\mathcal{O}_n$ implements a $2n$-qubit (resp. bit) unitary (function). We say $\boldsymbol{\mathcal{O}}$ is subset-gapped if for every n, $\mathcal{O}_n = \mathcal{F}_S$ (resp. $\mathcal{O}_n = f_S$) for $|S| = N$ or $|S| = 0.99N$.

Clearly $\boldsymbol{\mathcal{O}}$ is a subset-based oracle (see Definition 5), with $g^n(\mathcal{O}_n) = g^n(\mathcal{F}_S) = S$.

Then the following two lemmas give the desired oracle separation between AM and QCMA:

▶ **Lemma 8.** *For any subset-gapped $\boldsymbol{\mathcal{O}}$, the language $L_{\boldsymbol{\mathcal{O}}}$ that contains those strings 1^n such that $\mathcal{O}_n = f_S$ with $|S| = N$, is in $AM^{\boldsymbol{\mathcal{O}}}$.*

Lemma 8 is proven by Goldwasser and Sipser in [11].

▶ **Lemma 9.** *For any subset-gapped $\boldsymbol{\mathcal{O}}$, the language $L_{\boldsymbol{\mathcal{O}}}$ that contains those strings 1^n such that $\mathcal{O}_n = \mathcal{F}_S$ with $|S| = N$, is not in $\mathsf{QCMA}^{\boldsymbol{\mathcal{O}}}$.*

To prove this lemma, we follow Recipe 1. We address step 2 of the recipe in Lemma 10:

▶ **Lemma 10.** *Let $0 < \alpha < 1/2$ be a constant and $p(\cdot)$ be a polynomial function. Then there exists a positive integer $n^*(p, \alpha)$, such that for every positive integer $n > n^*(p, \alpha)$, and every subset family $\mathbf{S} \subseteq \mathbf{C}(N^2, N)$ such that $|\mathbf{S}| \geq |\mathbf{C}(N^2, N)|2^{-p(n)}$, there exists a subset family $\mathbf{S}_X \subseteq \mathbf{S}$ such that $\mathbf{S}_X$ is α-distributed with $|S_{fixed}| < .5N$.*

(Since $|S_{\text{fixed}}| < .5N$, this implies $|\{S : S \in \mathbf{S}_Y^n, S_{\text{fixed}} \subset S\}|$ is large, as desired.)

Proof Sketch. (Full proof in Appendix B.) We prove the existence of $\mathbf{S}_X$ by construction. Let $\mathbf{S}$ be any subset of $\mathbf{C}(N^2, N)$ with $|\mathbf{S}| \geq |\mathbf{C}(N^2, N)|2^{-p(n)}$. We construct $\mathbf{S}_X$ using the Fixing Procedure:

Fixing Procedure

1. Set $\mathbf{S}_X = \mathbf{S}$, and set $S_{\text{fixed}} = \emptyset$.
2. **a.** Let $\nu(i)$ be the number of subsets $S \in \mathbf{S}_X$ such that $i \in S$.
   b. If there exists some element i for which $|\mathbf{S}_X| > \nu(i) \geq |\mathbf{S}_X|N^{-\alpha}$, set $\mathbf{S}' \leftarrow \{S : S \in \mathbf{S}_X \text{ and } i \in S\}$, set $S_{\text{fixed}} \leftarrow S_{\text{fixed}} \cup i$, and return to step 2(a). Otherwise exit the Fixing Procedure.

By construction, the Fixing Procedure returns a set that is α-distributed (see Definition 6), so we only need to ensure that not too many elements are fixed. We obtain a lower bound on the final size of $\mathbf{S}_X$ because each time an element is fixed, the size of the set decreases by at most $N^{-\alpha}$. On the other hand, because $\mathbf{S}_X$ is contained in $\mathbf{C}(N^2, N)$, if a certain number of items are fixed, we have an upper bound on the size of $\mathbf{S}_X$ using the structure of $\mathbf{C}(N^2, N)$ and a combinatorial argument. We show that if more than $.5N$ items are fixed, these upper and lower bounds contradict each other, proving that less than $.5N$ items must be fixed before the Fixing Procedure terminates.

We address Step 3 of Recipe 1 with the following Lemma:

▶ **Lemma 11.** *Suppose $\mathbf{S}_X \subset \mathbf{C}(N^2, N)$ is the α-distributed subset created using the Fixing Procedure of Lemma 10, with fixed subset S_{fixed}. Let $\mathbf{S}_Y = \{S : S \in \mathbf{C}(N^2, 0.99N), S_{fixed} \subset S\}$. Then we can construct an adversary bound to prove that for every quantum algorithm G, there exists $S_x \in \mathbf{S}_X$, and $S_y \in \mathbf{S}_Y$, (that depend on G) such that, given oracle access to $\mathcal{F}_{S_x}$ or $\mathcal{F}_{S_y}$, G can not distinguish them with probability $\epsilon > .5$ without using $\left(1 - 2\sqrt{\epsilon(1-\epsilon)}\right) N^{\alpha/2}$ queries.*

Proof Sketch. (Full proof in Appendix B.) We use Theorem 6 from [6]. This result is identical to our Lemma 2, except with standard oracles rather than permutation oracles.

We let $\mathbf{R} = \mathbf{S}_X \times \mathbf{S}_Y$. To apply Theorem 6, we need to show that for elements i such that $i \in S_x$ but $i \notin S_y$ for $(S_x, S_y) \in \mathbf{R}$ that either (1) S_x is not connected to many other

sets S_y where $i \notin S_y$ or (2) S_y is not connected to many other sets S_x where $i \in S_x$. We use the α-distributed property of $\mathbf{S}_X$ to show that property (2) holds. We show a similar result for the case $i \notin S_x$ but $i \in S_y$ for $(S_x, S_y) \in \mathbf{R}$.

7 Oracle Separation of QMA and QCMA

In this section, we prove an oracle separation between QMA and QCMA. In particular, we show:

▶ **Theorem 12.** *There exists a randomized-preimage-correct oracle $\boldsymbol{\mathcal{O}}$, and a language $L_{\boldsymbol{\mathcal{O}}}$ which contains those unary strings 1^n where $\mathscr{O}_n = \mathscr{P}_{\boldsymbol{\sigma}_{\mathrm{pre}}(S)}$ with $S \in \boldsymbol{S}^n_{even}$ such that $L_{\boldsymbol{\mathcal{O}}} \notin \mathsf{QCMA}^{\boldsymbol{\mathcal{O}}}$.*

Combined with Theorem 4, this gives the desired separation between QMA and QCMA.

Really, we would like to prove a different result, one that involves preimage-correct oracles:

▶ **Definition 13** (preimage-correct oracles)**.** Let $\boldsymbol{\mathcal{O}}$ be a countably infinite set of unitaries: $\boldsymbol{\mathcal{O}} = \{\mathcal{O}_1, \mathcal{O}_2, \dots\}$, where each $\mathcal{O}_n$ implements an $(2n)$-qubit unitary. We say that $\boldsymbol{\mathcal{O}}$ is preimage-correct if for every n, $\mathcal{O}_n = \mathcal{P}_\sigma$, for some σ such that $S_{\mathrm{pre}}(\sigma) \in \mathbf{S}^n_{\mathrm{even}} \cup \mathbf{S}^n_{\mathrm{odd}}$.

The definition of preimage-correct oracles is very similar to that of randomized-preimage-correct oracles in Definition 3, except there is no randomness in preimage-correct oracles – they are unitaries. In fact, we believe:

▶ **Conjecture 1.** *There exists a preimage-correct oracle $\boldsymbol{\mathcal{O}}$, and a language $L_{\boldsymbol{\mathcal{O}}}$ which contains those unary strings 1^n where $\mathscr{O}_n = P_\sigma$ with $S_{pre}(\sigma) \in \boldsymbol{S}^n_{even}$, such that $L_{\boldsymbol{\mathcal{O}}} \notin \mathsf{QCMA}^{\boldsymbol{\mathcal{O}}}$.*

Theorem 4 applies equally well whether the oracle is preimage-correct or randomized-preimage-correct. So why is it harder to prove Conjecture 1 than Theorem 12? The answer is that Recipe 1 is much easier to use if the optimal witness depends only on a subset of integers. Note randomized-preimage-correct oracles have a one-to-one relationship with a subset of integers, and so the optimal witness only depends on that subset. However for preimage-correct oracles, the optimal witness might depend on some details of the permutation, which is more challenging to handle.

For convenience, we define the complexity class $\mathsf{QCMA}_{\mathsf{exp,poly}}$ to be the analogue of QCMA, in which the quantum verifier is allowed exponential time and space, but receives a polynomial length classical witness. While trivially bounded-error quantum exponential time, $\mathsf{BQEXP} = \mathsf{QCMA}_{\mathsf{exp,poly}}$, in general the query complexity of a $\mathsf{QCMA}_{\mathsf{exp,poly}}$ machine is not the same as the query complexity of a BQEXP machine.

Our proof works as follows. We first show that if there is a QCMA machine that decides $L_{\boldsymbol{\mathcal{O}}}$, for all randomized-preimage-correct oracles $\boldsymbol{\mathcal{O}}$, then there will be a $\mathsf{QCMA}_{\mathsf{exp,poly}}$ machine that decides $L_{\widetilde{\boldsymbol{\mathcal{O}}}}$ for any preimage-correct oracle $\widetilde{\boldsymbol{\mathcal{O}}}$, where, *crucially*, the optimal witness only depends on the pre-image subset of the permutation implemented by the oracle. Then using Recipe 1, we show that there is a language $L_{\widetilde{\boldsymbol{\mathcal{O}}}}$ for a preimage-correct oracle $\widetilde{\boldsymbol{\mathcal{O}}}$ such that no $\mathsf{QCMA}_{\mathsf{exp,poly}}$ machine that can decide the language using an efficient number of queries to $\widetilde{\boldsymbol{\mathcal{O}}}$ (with a witness that only depends on the pre-image subset). This implies that there is no QCMA machine that solves the randomized-preimage-correct oracle problem.

We first prove the reduction from deciding languages on pre-image correct oracles to languages on randomized pre-image correct oracles.

▶ **Lemma 14.** *Given a randomized-preimage-correct oracle $\boldsymbol{\mathcal{O}}$, let $1^n \in L_{\boldsymbol{\mathcal{O}}}$ if $\mathcal{O}_n = \mathscr{P}_{\boldsymbol{\sigma}_{pre}(S)}$ with $S \in \boldsymbol{S}^n_{even}$. Given a preimage-correct oracle $\tilde{\boldsymbol{\mathcal{O}}}$ let $1^n \in L_{\widetilde{\boldsymbol{\mathcal{O}}}}$ if $\mathcal{O}_n = \mathcal{P}_\sigma$ with $S_{pre}(\sigma) \in \mathbf{S}^n_{even}$. Then if there is a* QCMA *machine M that decides $L_{\boldsymbol{\mathcal{O}}}$ for every randomized-preimage-correct $\boldsymbol{\mathcal{O}}$, then there is a* $\mathsf{QCMA}_{\mathsf{exp,poly}}$ *machine $\widetilde{M}$ that decides $L_{\widetilde{\boldsymbol{\mathcal{O}}}}$ for every preimage-correct $\tilde{\boldsymbol{\mathcal{O}}}$ such that $\widetilde{M}$ uses at most a polynomial number of queries to $\tilde{\boldsymbol{\mathcal{O}}}$, and on input 1^n takes as input a classical witness w that depends only on $S_{pre}(\sigma)$.*

Proof Sketch. (Full proof in Appendix C.) Given a permutation σ, we can obtain all permutations σ' such that $S_{\text{pre}}(\sigma') = S_{\text{pre}}(\sigma)$ by first applying σ, and then permuting the first N elements and the last $N^2 - N$ elements separately. Consider a controlled-unitary that, if system A is in state $|i\rangle$, implements the i^{th} in-place permutation of the first N and last $N^2 - N$ elements on system B. If we start with system A in an equal superposition, apply $\mathcal{P}_\sigma$ to B, apply the control to A and B, and then trace out system A, the result is $\mathscr{P}_{S_{\text{pre}}(\sigma)}$ on system A. Thus, given any preimage-correct oracle $\mathcal{P}_\sigma$, we can simulate the randomized-preimage-correct oracle $\mathscr{P}_{S_{\text{pre}}(\sigma)}$.

Using this simulation trick, we can create an algorithm $\tilde{M}$ using a preimage-correct oracle that has the same outcomes as any algorithm M using a randomized-preimage-correct oracle, which uses the oracle the same number of times, and has a witness that only depends on the preimage subset. However, we do not believe the control permutation can be implemented efficiently, and that is why we must consider the class $\mathsf{QCMA}_{\mathsf{exp,poly}}$.

▶ **Lemma 15.** *There exists a preimage-correct $\boldsymbol{\mathcal{O}}$ such that there is no $\mathsf{QCMA}_{\mathsf{exp,poly}}^{\boldsymbol{\mathcal{O}}}$ machine M that decides $L_{\boldsymbol{\mathcal{O}}}$ using a polynomial number of queries, where the classical witness on input 1^n depends only on $S_{pre}(\sigma)$, when $\mathcal{O}_n = \mathcal{P}_\sigma$.*

Note that Lemma 15, combined with the contrapositive of Lemma 14, proves Theorem 12.

To prove Lemma 15, we use Recipe 1. Even though we do not have a true subset-based oracle (the function $g(\mathcal{P}_\sigma) = S_{\text{pre}}(\sigma)$ is not injective), using the constraint that the classical witness depends only on $S_{\text{pre}}(\sigma)$, we can apply the recipe.

Additionally, while Recipe 1 refers to the class QCMA, because we are only making a statement about query complexity (and say nothing about space or time complexity), the approach also applies to the query complexity of $\mathsf{QCMA}_{\mathsf{exp,poly}}$.

We prove steps 2 and 3 of Recipe 1 in Lemmas 16 and 17. These proofs are quite similar to the proofs of Lemmas 10 and 11; the full proofs can be found in Appendix D.

▶ **Lemma 16.** *Let $0 < \alpha < 1/2$ be a constant and $p(\cdot)$ be a polynomial function. Then there exists a positive integer $n^*(p, \alpha)$, such that for every $n > n^*(p, \alpha)$, and every subset family $\mathbf{S} \subset \mathbf{S}^n_{\text{even}}$ such that $|\mathbf{S}| \geq |\mathbf{S}^n_{\text{even}}|2^{-p(n)}$, there exists a subset family $\mathbf{S}_X \subset \mathbf{S}$ such that $\mathbf{S}_X$ is α-distributed. Furthermore the fixed subset S_{fixed} of $\mathbf{S}_X$ contains at most $N/3$ even elements.*

▶ **Lemma 17.** *Let $\mathbf{S}_X$ be the α-distributed set created using the Fixing Procedure from Lemma 16, with fixed subset S_{fixed}. Let $\mathbf{S}_Y = \{S : S \in \mathbf{S}^n_{odd}, S_{fixed} \subset S\}$. Then we can construct an adversary bound to prove that for every quantum algorithm G, there exists permutations $\sigma_x, \sigma_y \in \boldsymbol{\sigma}^n$ with $S_{pre}(\sigma_x) \in \mathbf{S}_X$ and $S_{pre}(\sigma_y) \in \mathbf{S}_Y$, (that depend on G) such that, given oracle access to $\mathcal{P}_{\sigma_x}$ or $\mathcal{P}_{\sigma_y}$, G can not distinguish them with probability $\epsilon > .5$ without using $\left(1 - 2\sqrt{\epsilon(1-\epsilon)}\right) N^{\alpha/2}$ queries.*

The proof strategy of Lemma 16 (like Lemma 10) involves a Fixing Procedure. However, the details are slightly more complex because we must deal with fixing even and odd elements.

The proof strategy of Lemma 17 is similar to Lemma 11, except we use a more complex relation $\mathbf{R}$ for the adversary bound. The challenge is that for two similar subsets S_x and S_y, there exist permutations σ_x and σ_y that are extremely dissimilar but for which $S_{\text{pre}}(\sigma_x) = S_x$ and $S_{\text{pre}}(\sigma_y) = S_y$. We want to create a relationship $\mathbf{R}$ that connects similar permutations, while only having information about the structure of the related subsets. To address this problem, we note that for any two subsets S_x and S_y, we can create a one-to-one matching between the elements of $\boldsymbol{\sigma}_{\text{pre}}(S_x)$ and the elements of $\boldsymbol{\sigma}_{\text{pre}}(S_y)$ such that each permutation is matched with a similar permutation. Using this one-to-one matching, we create a relationship $\mathbf{R}$ between permutations that inherits the properties of the related subsets.

As an immediate corollary of Theorem 4 and Theorem 12, there exists a randomized-preimage-correct oracle $\boldsymbol{\mathcal{O}}$ and language $L_{\boldsymbol{\mathcal{O}}}$ such that $L \notin \mathsf{QCMA}^{\boldsymbol{\mathcal{O}}}$ but $L \in \mathsf{QMA}^{\boldsymbol{\mathcal{O}}}$, and so $\mathsf{QMA}^{\boldsymbol{\mathcal{O}}} \not\subseteq \mathsf{QCMA}^{\boldsymbol{\mathcal{O}}}$.

References

1 Complexity zoo. URL: https://complexityzoo.uwaterloo.ca/Complexity_Zoo.

2 Scott Aaronson. Quantum lower bound for the collision problem. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 635–642. ACM, 2002. doi:10.1145/509907.509999.

3 Scott Aaronson and Greg Kuperberg. Quantum versus classical proofs and advice. In *Computational Complexity, 2007. CCC'07. Twenty-Second Annual IEEE Conference on*, pages 115–128. IEEE, 2007.

4 Scott Aaronson and Yaoyun Shi. Quantum lower bounds for the collision and the element distinctness problems. *J. ACM*, 51(4):595–605, 2004. doi:10.1145/1008731.1008735.

5 Dorit Aharonov and Tomer Naveh. Quantum NP – a survey. *arXiv preprint quant-ph/0210077*, 2002.

6 Andris Ambainis. Quantum lower bounds by quantum arguments. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 636–643. ACM, 2000.

7 Andris Ambainis, Loïck Magnin, Martin Roetteler, and Jérémie Roland. Symmetry-assisted adversaries for quantum state generation. In *Computational Complexity (CCC), 2011 IEEE 26th Annual Conference on*, pages 167–177. IEEE, 2011.

8 Theodore P. Baker, John Gill, and Robert Solovay. Relativizations of the P =? NP question. *SIAM J. Comput.*, 4(4):431–442, 1975. doi:10.1137/0204037.

9 Aleksandrs Belovs. Variations on quantum adversary. *arXiv preprint 1504.06943*, 2015.

10 J Niel De Beaudrap, Richard Cleve, and John Watrous. Sharp quantum versus classical query complexity separations. *Algorithmica*, 34(4):449–461, 2002.

11 Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 59–68. ACM, 1986. doi:10.1145/12130.12137.

12 Alex B Grilo, Iordanis Kerenidis, and Jamie Sikora. QMA with subset state witnesses. *arXiv preprint arXiv:1410.2882*, 2014.

13 Aram W Harrow and David J Rosenbaum. Uselessness for an oracle model with internal randomness. *Quantum Information & Computation*, 14(7&8):608–624, 2014.

14 Peter Hoyer, Troy Lee, and Robert Špalek. Negative weights make adversaries stronger. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 526–535. ACM, 2007.

15 Alexei Yu Kitaev. Quantum computation: Algorithms and error correction. *Russian Math. Surveys*, 52(6):1191–1249, 1997.

16 Alexei Yu Kitaev, Alexander Shen, and Mikhail N Vyalyi. *Classical and quantum computation*. Number 47 in Graduate Studies in Mathematics. American Mathematical Soc., 2002. doi:10.1090/gsm/047.

17 Troy Lee, Rajat Mittal, Ben W Reichardt, Robert Špalek, and Mario Szegedy. Quantum query complexity of state conversion. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 344–353. IEEE, 2011.

18 Chris Marriott and John Watrous. Quantum Arthur-Merlin games. *Comput. Complex.*, 14(2):122–152, 2005. doi:10.1007/s00037-005-0194-x.

19 Oded Regev and Liron Schiff. Impossibility of a quantum speed-up with a faulty oracle. In *Automata, Languages and Programming*, pages 773–781. Springer, 2008.

20 Nikolay K. Vereshchagin. Oracle separation of complexity classes and lower bounds for perceptrons solving separation problems. *Izvestiya: Mathematics*, 59:1103–1122, 1995. doi:10.1070/IM1995v059n06ABEH000050.

A An Adversary Bound for Permutation Oracles

We will prove Lemma 2:

▶ **Lemma 2.** *Let $\boldsymbol{\sigma} \subset [V] \to [V]$ be a subset of permutations acting on the elements $[V]$. Let $f : \boldsymbol{\sigma} \to \{0,1\}$ be a function of permutations. Let $\boldsymbol{\sigma}_X \subset \boldsymbol{\sigma}$ be a set of permutations such that if $\sigma \in \boldsymbol{\sigma}_X$, then $f(\sigma) = 1$. Let $\boldsymbol{\sigma}_Y \subset \boldsymbol{\sigma}$ be a permutation family such that if $\sigma \in \boldsymbol{\sigma}_Y$ then $f(\sigma) = 0$. Let $R \subset \boldsymbol{\sigma}_X \times \boldsymbol{\sigma}_Y$ be such that*

- *For every $\sigma_x \in \boldsymbol{\sigma}_X$, there exists at least m different $\sigma_y \in \boldsymbol{\sigma}_Y$ such that $(\sigma_x, \sigma_y) \in R$.*
- *For every $\sigma_y \in \boldsymbol{\sigma}_Y$, there exists at least m' different $\sigma_x \in \boldsymbol{\sigma}_X$ such that $(\sigma_x, \sigma_y) \in R$.*
- *Let $l_{x,i}$ be the number of $\sigma_y \in \boldsymbol{\sigma}_Y$ such that $(\sigma_x, \sigma_y) \in R$ and $\sigma_x(i) \neq \sigma_y(i)$. Let $l_{y,i}$ be the number of $\sigma_x \in \boldsymbol{\sigma}_Y$ such that $(\sigma_x, \sigma_y) \in R$ and $\sigma_x(i) \neq \sigma_y(i)$. Then let $l_{max} = \max_{(\sigma_x,\sigma_y)\in R,i} l_{x,i} l_{y,i}$.*

Then given an in-place permutation oracle $\mathcal{P}_\sigma$ for $\sigma \in \boldsymbol{\sigma}$ that acts as $\mathcal{P}_\sigma|i\rangle = |\sigma(i)\rangle$, any quantum algorithm that correctly evaluates $f(\sigma)$ with probability $1-\epsilon$ for every element of $\boldsymbol{\sigma}_X$ and $\boldsymbol{\sigma}_Y$ must use $\left(1 - 2\sqrt{\epsilon(1-\epsilon)}\right)\sqrt{\frac{mm'}{l_{max}}}$ queries to the oracle.

We note that this is identical to Ambainis' adversary bound for permutations (see Theorem 6 in [6]).

Proof. We assume that we have a control permutation oracle, that acts as

$$\mathcal{P}|x\rangle_C|i\rangle_A|z\rangle_Q = |x\rangle|\sigma_x(i)\rangle|z\rangle \tag{6}$$

where the Hilbert space $\mathcal{H}_C$ has dimension $|\boldsymbol{\sigma}|$,the Hilbert space $\mathcal{H}_A$ has dimension V and is where the permutation is carried out, and $\mathcal{H}_Q$ is a set of ancilla qubits.

Let $|\psi^t\rangle$ be the state of the system immediately after t uses of the control oracle. Let $|\varphi^t\rangle$ be the state of the system immediately before the t^{th} use of the control oracle. Let ρ^t be the reduced state of the system immediately after t uses of the control oracle, where systems A and Q have been traced out. That is, $\rho^t = \mathrm{tr}_{AQ}(|\psi^t\rangle\langle\psi^t|)$. Let $(\rho^t)_{xy}$ be the $(x,y)^{\text{th}}$ element of the density matrix. Then we will track the progress of the following measure:

$$W^t = \sum_{(\sigma_x,\sigma_y)\in R} \left|(\rho^t)_{xy}\right|. \tag{7}$$

Notice that unitaries that only act on the subsystems Q and A do not affect W^t.

If the state before the first use of the oracle is

$$|\psi^0\rangle = \left(\frac{1}{\sqrt{2|\boldsymbol{\sigma}_X|}} \sum_{\sigma_x \in \boldsymbol{\sigma}_X} |x\rangle + \frac{1}{\sqrt{2|\boldsymbol{\sigma}_Y|}} \sum_{\sigma_y \in \boldsymbol{\sigma}_Y} |y\rangle \right) \otimes |\phi\rangle_{AQ}, \tag{8}$$

then following Ambainis (e.g. Theorem 2 [6]), we have that for an algorithm to succeed with probability at least $1-\epsilon$ after T uses of the oracle, we must have

$$W^0 - W^T > \left(1 - 2\sqrt{\epsilon(1-\epsilon)}\right)\sqrt{mm'} \tag{9}$$

Now we calculate how much W^t can change between uses of the oracle. Suppose without loss of generality that

$$|\varphi^t\rangle = \frac{1}{\sqrt{2|\boldsymbol{\sigma}_X|}} \sum_{\sigma_x \in \boldsymbol{\sigma}_X} \sum_{i,z} \alpha_{x,i,z} |x,i,z\rangle_{CAQ} + \frac{1}{\sqrt{2|\boldsymbol{\sigma}_Y|}} \sum_{\sigma_y \in \boldsymbol{\sigma}_Y} \sum_{i,z} \alpha_{y,i,z} |y,i,z\rangle_{CAQ}. \tag{10}$$

Then we have

$$\begin{aligned} |\psi^t\rangle &= \frac{1}{\sqrt{2|\boldsymbol{\sigma}_X|}} \sum_{\sigma_x \in \boldsymbol{\sigma}_X} \sum_{i,z} \alpha_{x,i,z} |x,\sigma_x(i),z\rangle_{CAQ} + \frac{1}{\sqrt{2|\boldsymbol{\sigma}_Y|}} \sum_{\sigma_y \in \boldsymbol{\sigma}_Y} \sum_{i,z} \alpha_{y,i,z} |y,\sigma_y(i),z\rangle_{CAQ} \\ &= \frac{1}{\sqrt{2|\boldsymbol{\sigma}_X|}} \sum_{\sigma_x \in \boldsymbol{\sigma}_X} \sum_{i,z} \alpha_{x,\sigma_x^{-1}(i),z} |x,i,z\rangle_{CAQ} + \frac{1}{\sqrt{2|\boldsymbol{\sigma}_Y|}} \sum_{\sigma_y \in \boldsymbol{\sigma}_Y} \sum_{i,z} \alpha_{y,\sigma_y^{-1}(i),z} |y,i,z\rangle_{CAQ}. \end{aligned} \tag{11}$$

Hence for $(\sigma_x, \sigma_y) \in R$, we have

$$\begin{aligned} (\rho^t)_{xy} &= \frac{1}{2\sqrt{|\boldsymbol{\sigma}_X||\boldsymbol{\sigma}_Y|}} \sum_{i,z} \alpha_{x,\sigma_x^{-1}(i),z} \alpha^*_{y,\sigma_y^{-1}(i),z} \\ (\rho^{t-1})_{xy} &= \frac{1}{2\sqrt{|\boldsymbol{\sigma}_X||\boldsymbol{\sigma}_Y|}} \sum_{i,z} \alpha_{x,i,z} \alpha^*_{y,i,z}, \end{aligned} \tag{12}$$

where $(\cdot)^*$ signifies the complex conjugate. Now we can calculate $W^t - W^{t-1}$:

$$\begin{aligned} W^{t-1} - W^t &= \sum_{(\sigma_x,\sigma_y)\in R} |(\rho^{t-1})_{xy}| - |(\rho^t)_{xy}| \\ &\leq \sum_{(\sigma_x,\sigma_y)\in R} |(\rho^{t-1})_{xy} - (\rho^t)_{xy}|. \end{aligned} \tag{13}$$

From Eq. (12), we see that whenever $\sigma_x^{-1}(i) = \sigma_y^{-1}(i)$, we have a cancellation between the corresponding elements of $(\rho^t)_{xy}$ and $(\rho^{t-1})_{xy}$. However, when $\sigma_x^{-1}(i) \neq \sigma_y^{-1}(i)$, terms do not cancel. To see this more explicitly, we rewrite Eq. (13) as

$$\begin{aligned} W^{t-1} - W^t \leq \\ \frac{1}{2\sqrt{|\boldsymbol{\sigma}_X||\boldsymbol{\sigma}_Y|}} \sum_{(\sigma_x,\sigma_y)\in R} \Bigg| \sum_z \Bigg[\sum_{i:\sigma_x(i)=\sigma_y(i)} \alpha_{x,i,z}\alpha^*_{y,i,z} + \sum_{i:\sigma_x(i)\neq\sigma_y(i)} \alpha_{x,i,z}\alpha^*_{y,i,z} \\ - \sum_{i:\sigma_x^{-1}(i)=\sigma_y^{-1}(i)} \alpha_{x,\sigma_x^{-1}(i),z}\alpha^*_{y,\sigma_y^{-1}(i),z} - \sum_{i:\sigma_x^{-1}(i)\neq\sigma_y^{-1}(i)} \alpha_{x,\sigma_x^{-1}(i),z}\alpha^*_{y,\sigma_y^{-1}(i),z} \Bigg] \Bigg|. \end{aligned} \tag{14}$$

Consider the sets $T_{x,y} = \{i : \sigma_x(i) = \sigma_y(i)\}$ and $U_{x,y} = \{\sigma_x^{-1}(i) : \sigma_x^{-1}(i) = \sigma_y^{-1}(i)\}$. We will show $U_{x,y} = T_{x,y}$. Suppose $i \in T_{x,y}$. Then $\sigma_x(i) = \sigma_y(i) = i'$, for some i'. But that implies

$\sigma_x^{-1}(i') = \sigma_y^{-1}(i') = i$, so $\sigma_x^{-1}(i') = i \in U_{x,y}$, and thus $T_{x,y} \subset U_{x,y}$. The opposite direction is shown similarly. Therefore, those two sums in Eq. (14) cancel, and, moving the summation over z and i outside the absolute values by the triangle inequality, we are left with

$$W^{t-1} - W^t \leq \frac{1}{2\sqrt{|\boldsymbol{\sigma}_X||\boldsymbol{\sigma}_Y|}} \sum_{z,(\sigma_x,\sigma_y)\in R} \left(\sum_{i:\sigma_x(i)\neq\sigma_y(i)} \left|\alpha_{x,i,z}\alpha^*_{y,i,z}\right| + \sum_{i:\sigma_x^{-1}(i)\neq\sigma_y^{-1}(i)} \left|\alpha_{x,\sigma_x^{-1}(i),z}\alpha^*_{y,\sigma_y^{-1}(i),z}\right| \right). \tag{15}$$

Now we use the AM-GM to bound the terms in the absolute values:

$$\begin{aligned} W^{t-1} - W^t \leq & \frac{1}{2} \sum_{z,(\sigma_x,\sigma_y)\in R} \left(\sum_{i:\sigma_x(i)\neq\sigma_y(i)} \left(\sqrt{\frac{l_{y,i}}{l_{x,i}}} \frac{|\alpha_{x,i,z}|^2}{2|\boldsymbol{\sigma}_X|} + \sqrt{\frac{l_{x,i}}{l_{y,i}}} \frac{|\alpha^*_{y,i,z}|^2}{2|\boldsymbol{\sigma}_Y|} \right) \right) \\ & + \frac{1}{2} \sum_{z,(\sigma_x,\sigma_y)\in R} \left(\sum_{i:\sigma_x^{-1}(i)\neq\sigma_y^{-1}(i)} \left(\sqrt{\frac{l_{y,i}}{l_{x,i}}} \frac{\left|\alpha_{x,\sigma_x^{-1}(i),z}\right|^2}{2|\boldsymbol{\sigma}_X|} + \sqrt{\frac{l_{x,i}}{l_{y,i}}} \frac{\left|\alpha^*_{y,\sigma_y^{-1}(i),z}\right|^2}{2|\boldsymbol{\sigma}_Y|} \right) \right) \end{aligned} \tag{16}$$

We now show that for $(\sigma_x, \sigma_y) \in R$,

$$\begin{aligned} \sum_{i:\sigma_x^{-1}(i)\neq\sigma_y^{-1}(i)} |\alpha_{x,\sigma_x^{-1}(i),z}|^2 &= \sum_{i:\sigma_x(i)\neq\sigma_y(i)} |\alpha_{x,i,z}|^2, \\ \sum_{i:\sigma_x^{-1}(i)\neq\sigma_y^{-1}(i)} |\alpha_{y,\sigma_y^{-1}(i),z}|^2 &= \sum_{i:\sigma_x(i)\neq\sigma_y(i)} |\alpha_{y,i,z}|^2. \end{aligned} \tag{17}$$

We prove the first equality, and the second is proven similarly. We define

$$\begin{aligned} T'_{x,y} &= [V] \setminus T_{x,y}, \\ U'_{x,y} &= [V] \setminus U_{x,y}. \end{aligned} \tag{18}$$

Looking at the definition of $T_{x,y}$ and $U_{x,y}$, we see that

$$\begin{aligned} T'_{x,y} &= \{i : \sigma_x(i) \neq \sigma_y(i)\} \\ U'_{x,y} &= \{\sigma_x^{-1}(i) : \sigma_x^{-1}(i) \neq \sigma_y^{-1}(i)\}. \end{aligned} \tag{19}$$

We previously showed $T_{x,y} = U_{x,y}$, so we have $T'_{x,y} = U'_{x,y}$. Therefore

$$\sum_{i:\sigma_x^{-1}(i)\neq\sigma_y^{-1}(i)} |\alpha_{x,\sigma_x^{-1}(i),z}|^2 = \sum_{i:U'_{x,y}} |\alpha_{x,j,z}|^2 = \sum_{i:T'_{x,y}} |\alpha_{x,i,z}|^2 \sum_{i:\sigma_x(i)\neq\sigma_y(i)} |\alpha_{x,i,z}|^2. \tag{20}$$

Thus, Eq. (16) becomes

$$W^{t-1} - W^t \leq \sum_{z,(\sigma_x,\sigma_y)\in R} \left(\sum_{i:\sigma_x(i)\neq\sigma_y(i)} \left(\sqrt{\frac{l_{y,i}}{l_{x,i}}} \frac{|\alpha_{x,i,z}|^2}{2|\boldsymbol{\sigma}_X|} + \sqrt{\frac{l_{x,i}}{l_{y,i}}} \frac{|\alpha^*_{y,i,z}|^2}{2|\boldsymbol{\sigma}_Y|} \right) \right). \tag{21}$$

Now we switch the order of summation and then use the definition of $l_{x,i}$ and $l_{y,i}$ to get

$$\begin{aligned} & W^{t-1} - W^t \\ \leq & \sum_{i\in[V],z} \left(\sum_{(\sigma_x,\sigma_y)\in R:\sigma_x(i)\neq\sigma_y(i)} \left(\sqrt{\frac{l_{y,i}}{l_{x,i}}} \frac{|\alpha_{x,i,z}|^2}{2|\boldsymbol{\sigma}_X|} + \sqrt{\frac{l_{x,i}}{l_{y,i}}} \frac{|\alpha^*_{y,i,z}|^2}{2|\boldsymbol{\sigma}_Y|} \right) \right) \\ \leq & \sum_{i\in[V],z} \left(\sum_{\sigma_x\in\boldsymbol{\sigma}_X} \sqrt{l_{x,i} \max_{\sigma_y:(\sigma_x,\sigma_y)\in R} l_{y,i}} \frac{|\alpha_{x,i,z}|^2}{2|\boldsymbol{\sigma}_X|} + \sum_{\sigma_y\in\boldsymbol{\sigma}_Y} \sqrt{l_{y,i} \max_{\sigma_x:(\sigma_x,\sigma_y)\in R} l_{x,i}} \frac{|\alpha^*_{y,i,z}|^2}{2|\boldsymbol{\sigma}_Y|} \right) \end{aligned} \tag{22}$$

Finally, using the definition of l_{max} we have

$$\begin{aligned} W^{t-1} - W^t \leq & \sqrt{l_{max}} \sum_{i\in[V],z} \left(\sum_{x\in\boldsymbol{\sigma}_X} \frac{|\alpha_{x,i,z}|^2}{2|\boldsymbol{\sigma}_X|} + \sum_{\sigma_y\in\boldsymbol{\sigma}_Y} \frac{|\alpha^*_{y,i,z}|^2}{2|\boldsymbol{\sigma}_Y|} \right) \\ \leq & \sqrt{l_{max}}, \end{aligned} \tag{23}$$

where we have used that Eq. (10) is a normalized state. ◀

B Proofs of Lemmas 10 and 11

▶ **Lemma 10.** *Let $0 < \alpha < 1/2$ be a constant and $p(\cdot)$ be a polynomial function. Then there exists a positive integer $n^*(p,\alpha)$, such that for every positive integer $n > n^*(p,\alpha)$, and every subset family $\mathbf{S} \subseteq \mathbf{C}(N^2, N)$ such that $|\mathbf{S}| \geq |\mathbf{C}(N^2,N)|2^{-p(n)}$, there exists a subset family $\mathbf{S}_X \subseteq \mathbf{S}$ such that $\mathbf{S}_X$ is α-distributed with $|S_{fixed}| < .5N$.*

Proof. We prove the existence of $\mathbf{S}'$ by construction. Let $\mathbf{S}$ be any subset family of $\mathbf{C}(N^2,N)$ such that $|\mathbf{S}| \geq |\mathbf{C}(N^2,N)|2^{-p(n)}$. We construct $\mathbf{S}'$ using the following procedure:

Fixing Procedure

1. Set $\mathbf{S}' = \mathbf{S}$, and set $S_{\text{fixed}} = \emptyset$.

2. a. Let $\nu(i)$ be the number of subsets $S \in \mathbf{S}_X$ such that $i \in S$.

b. If there exists some element i for which

$$|\mathbf{S}_X| > \nu(i) \geq |\mathbf{S}_X| N^{-\alpha} \tag{24}$$

set $\mathbf{S}' \leftarrow \{S : S \in \mathbf{S}_X \text{ and } i \in S\}$, set $S_{\text{fixed}} \leftarrow S_{\text{fixed}} \cup i$, and return to step 2(a). Otherwise exit the Fixing Procedure.

The Fixing Procedure by construction will always return a set that satisfies Definition 6. Now we just need to bound the size of S_{fixed}.

Let's suppose that at some point in the Fixing Procedure, for sets $\mathbf{S}'$ and S_{fixed}, we have $.5N$ items fixed. Suppose for contradiction there is some element $i^* \notin S_{\text{fixed}}$ that appears in greater than $N^{-\alpha}$ fraction of $S \in \mathbf{S}'$.

Let us look at the set family $\mathbf{S}'' = \{S : S \in \mathbf{S}', i^* \in S\}$. Because $(S_{\text{fixed}} \cup i^*) \subset S$ for all $S \in \mathbf{S}''$, there are $.5N - 1$ elements in each $S \in \mathbf{S}''$ that can be chosen freely from the remaining $N^2 - .5N - 1$ un-fixed elements. Thus, we have

$$|\mathbf{S}''| \leq \binom{N^2 - .5N - 1}{.5N - 1}. \tag{25}$$

By assumption $|\mathbf{S}''| \geq |\mathbf{S}'|N^{-\alpha}$, so

$$\begin{aligned}
|\mathbf{S}'| &\leq \binom{N^2 - .5N - 1}{.5N - 1} N^{\alpha} \\
&\leq \binom{N^2}{.5N} N^{\alpha} \\
&\leq (2Ne)^{N/2} N^{\alpha} \\
&= 2^{N/2(\log(2e)+\log N)+\log(N)\alpha} \\
&= 2^{O(N)+(N/2)\log N}. \qquad (26)
\end{aligned}$$

However, we can also bound the size of $\mathbf{S}'$ from the Fixing Procedure. Notice that at every step of the Fixing Procedure, the size of $\mathbf{S}'$ is reduced by at most a factor $N^{-\alpha}$. Since we are assuming $.5N$ elements are in S_{fixed}, the Fixing Procedure can reduce the original set $\mathbf{S}$ by at most a factor $N^{-\alpha N/2}$. Since $|\mathbf{S}| \geq |\mathbf{C}(N^2, N)|2^{-p(n)}$, we have that at this point in the Fixing Procedure

$$\begin{aligned}
|\mathbf{S}'| &\geq |\mathbf{C}(N^2, N)|2^{-p(n)} N^{-\alpha N/2} \\
&= \binom{N^2}{N} 2^{-p(n)} N^{-\alpha N/2} \\
&\geq N^N 2^{-p(n)} N^{-\alpha N/2} \\
&= 2^{N\log N - p(n) - \log(N)\alpha N/2} \\
&= 2^{-O(N)+N\log N(1-\alpha/2)}. \qquad (27)
\end{aligned}$$

Notice that as long as $\alpha < 1$, for large enough N (in particular, for $N > 2^{n^*}$ for some positive integer n^*, where n^* depends on α and $p(\cdot)$), the bound of Eq. (27) will be larger than the bound of Eq. (26), giving a contradiction. Therefore, our assumption must have been false, and more than $N/2$ elements can not have been fixed during the Fixing Procedure. Therefore, the final set produced by the Fixing Procedure will satisfy point (2) of Definition 6. ◀

▶ **Lemma 11.** *Suppose $\mathbf{S}_X \subset \mathbf{C}(N^2, N)$ is the α-distributed subset created using the Fixing Procedure of Lemma 10, with fixed subset S_{fixed}. Let $\mathbf{S}_Y = \{S : S \in \mathbf{C}(N^2, 0.99N), S_{fixed} \subset S\}$. Then we can construct an adversary bound to prove that for every quantum algorithm G, there exists $S_x \in \mathbf{S}_X$, and $S_y \in \mathbf{S}_Y$, (that depend on G) such that, given oracle access to $\mathcal{F}_{S_x}$ or $\mathcal{F}_{S_y}$, G can not distinguish them with probability $\epsilon > .5$ without using $\left(1 - 2\sqrt{\epsilon(1-\epsilon)}\right) N^{\alpha/2}$ queries.*

Proof. Note $\mathbf{S}_Y$ is non-empty, since only $.5N$ elements are in S_{fixed}.

We will use Theorem 6 from [6]. This result is identical to our Lemma 2, except with standard oracles rather than permutation oracles. We define the relation $\mathbf{R}$ as:

$$\mathbf{R} = \{(S_x, S_y) : S_x \in \mathbf{S}_X, S_y \in \mathbf{S}_Y\}. \qquad (28)$$

Notice that each $S_x \in \boldsymbol{S}_X$ is paired with every element of $\mathbf{S}_Y$. Thus $m = |\mathbf{S}_Y|$. Likewise $m' = |\mathbf{S}_X|$.

Now consider $(S_x, S_y) \in \mathbf{R}$. We first consider the case of some element j such that $j \in S_x$ but $j \notin S_y$. By our construction of $\mathbf{S}_Y$, $j \notin S_{\text{fixed}}$. We upper bound $l_{x,j}$, the number of $S_{y'}$ such that $(S_x, S_{y'}) \in \mathbf{R}$ and $j \notin S_{y'}$. We use the trivial upper bound $l_{x,j} \leq |\mathbf{S}_Y|$, which is sufficient for our purposes. Next we need to upper bound $l_{y,j}$, the number of $S_{x'}$ such that

$(S_{x'}, S_y) \in \mathbf{R}$ and $j \in S_x$. Since S_y is paired with every element of $\mathbf{S}_X$ in $\mathbf{R}$, we just need to determine the number of sets in $\mathbf{S}_X$ that contain j. Because $\mathbf{S}_X$ is α-distributed, there can be at most $N^{-\alpha}|\mathbf{S}_X|$ elements of $\mathbf{S}_X$ that contain j. In this case we have

$$l_{x,j} l_{y,j} \leq |\mathbf{S}_X||\mathbf{S}_Y| N^{-\alpha}. \tag{29}$$

We now consider the case that $j \in S_y$ but $j \notin S_x$. (Note this case only occurs when S_{fixed} contains less than $0.99N$ elements.) We upper bound $l_{y,j}$, the number of $S_{x'}$ such that $(S_{x'}, S_y) \in \mathbf{R}$ and $j \notin S_{x'}$. Again, we use the trivial upper bound of $l_{y,j} \leq |\mathbf{S}_X|$, which is sufficient for our analysis. Next we upper bound $l_{x,j}$, the number of $S_{y'}$ such that $(S_x, S_{y'}) \in \mathbf{R}$ and $j \in S_{y'}$. In our choice of $\mathbf{R}$, S_x is paired with every $S_y \in \mathbf{S}_Y$, so we need to count the number of $S \in \mathbf{S}_Y$ that contain j. We have

$$\begin{aligned} l_{x,j} =& \binom{N^2 - S_{\text{fixed}} - 1}{0.99N - S_{\text{fixed}} - 1} \\ =& \frac{0.99N - S_{\text{fixed}}}{N^2 - S_{\text{fixed}}} |\mathbf{S}_Y| \\ \leq& \frac{1}{N} |\mathbf{S}_Y|. \end{aligned} \tag{30}$$

Therefore in this case, we have

$$l_{x,j} l_{y,j} \leq |\mathbf{S}_X||\mathbf{S}_Y| N^{-1}. \tag{31}$$

Looking at Eq. (29) and Eq. (31), we see that because $\alpha < 1$, the bound of Eq. (29) dominates, and so we have that

$$\sqrt{\frac{mm'}{l_{x,j} l_{y,j}}} \geq \sqrt{\frac{|\mathbf{S}_X||\mathbf{S}_Y|}{|\mathbf{S}_X||\mathbf{S}_Y| N^{-\alpha}}} = N^{\alpha/2}. \tag{32}$$

Using the contrapositive of Lemma 2, if an algorithm G makes less than q queries to an oracle $\mathcal{F}_S$ where S is promised to be in $\mathbf{S}_X$ or $\mathbf{S}_Y$, there exists at least one element of $\mathbf{S}_X$ and one element of $\mathbf{S}_Y$ such that the probability of distinguishing between the corresponding oracles less than is $1/2 + \epsilon$, where

$$\frac{1}{2} \sqrt{2 N^{-\alpha/2} q} > \epsilon. \tag{33}$$

Equivalently, there exists at least one element of $\boldsymbol{S}_X$ and one element of $\boldsymbol{S}_Y$ such that in order for $\mathcal{A}$ to distinguish them with constant bias, one requires $\Omega(N^{\alpha/2})$ queries. ◀

C Proof of Lemma 14

▶ **Lemma 14.** *Given a randomized-preimage-correct oracle $\boldsymbol{\mathcal{O}}$, let $1^n \in L_{\boldsymbol{\mathcal{O}}}$ if $\mathcal{O}_n = \mathscr{P}_{\boldsymbol{\sigma}_{pre}(S)}$ with $S \in \boldsymbol{S}^n_{even}$. Given a preimage-correct oracle $\tilde{\boldsymbol{\mathcal{O}}}$ let $1^n \in L_{\tilde{\boldsymbol{\mathcal{O}}}}$ if $\mathcal{O}_n = \mathcal{P}_\sigma$ with $S_{pre}(\sigma) \in \mathbf{S}^n_{even}$. Then if there is a* QCMA *machine M that decides $L_{\boldsymbol{\mathcal{O}}}$ for every randomized-preimage-correct $\boldsymbol{\mathcal{O}}$, then there is a* $\mathsf{QCMA}_{\mathsf{exp,poly}}$ *machine $\widetilde{M}$ that decides $L_{\tilde{\boldsymbol{\mathcal{O}}}}$ for every preimage-correct $\tilde{\boldsymbol{\mathcal{O}}}$ such that $\widetilde{M}$ uses at most a polynomial number of queries to $\tilde{\boldsymbol{\mathcal{O}}}$, and on input 1^n takes as input a classical witness w that depends only on $S_{pre}(\sigma)$.*

Proof. We denote the composition of two CPTP maps with $\circ$, so $\mathscr{E} \circ \mathscr{F}$ means apply $\mathscr{F}$ first, and then $\mathscr{E}$.

For each input 1^n, M applies an algorithm that takes as input a standard basis state. Because S completely characterizes $\mathscr{P}_{\boldsymbol{\sigma}_{\mathrm{pre}(S)}}$, the optimal witness will depend only on S.

Suppose on input 1^n to M, the algorithm is the following:

$$\mathscr{L}_{AB} \circ (\mathscr{O})_A \circ (\mathscr{U}_t)_{AB} \circ \cdots \circ (\mathscr{U}_2)_{AB} \circ (\mathscr{O})_A \circ (\mathscr{U}_1)_{AB}(|w\rangle\langle w| \otimes |\psi_0\rangle\langle\psi_0|)_{AB} \tag{34}$$

where $|w\rangle\langle w|$ is the witness state (that depends only on S) in the standard basis and $\mathscr{U}_i$ are fixed unitaries and $\mathscr{L}$. The two subspaces A and B refer to the subset where the oracle acts (A) and the rest of the workspace (B). The two subspaces do *not* refer to the tensor product structure of the initial state.

For $i \in [N!(N^2-N)!]$ let $\boldsymbol{\tau}^n = \{\tau_i\}$ be the set of permutations on the elements of $[N^2]$ that do not mix the first N elements with the last $N^2 - N$ elements. Then let $\mathcal{P}_n^{\mathrm{C}}$ be the following control-permutation:

$$\mathcal{P}_n^{\mathrm{C}}|i\rangle|j\rangle = \begin{cases} |i\rangle|\tau_i(j)\rangle \text{ for } i \in [N!(N^2-N)!] \\ |i\rangle|j\rangle \text{ otherwise.} \end{cases} \tag{35}$$

$\mathscr{P}_n^{\mathrm{C}}$ is the respective CPTP map.

$\mathcal{P}_n^{\mathrm{C}}$ is a completely known unitary that is independent of the oracle, however, we do not know how to implement this unitary in polynomial time. This unitary is the reason we consider the class $\mathsf{QCMA}_{\mathsf{exp,poly}}$ in this proof rather than the more standard QCMA. Ultimately, we care about query complexity - not the complexity of the unitaries that occur between the oracle applications.

Let

$$|\chi_n\rangle = \frac{1}{\sqrt{N!(N^2-N)!}} \sum_{i=1}^{N!(N^2-N)!} |i\rangle \tag{36}$$

Then on input 1^n we have $\tilde{M}$ implement the algorithm

$$\begin{aligned} &\mathscr{L}_{AB} \circ (\mathscr{P}_n^{\mathrm{C}})_{C_t A} \circ (\mathscr{O})_A \circ (\mathscr{U}_t)_{AB} \circ \cdots \\ &\quad \circ (\mathscr{U}_2)_{AB} \circ (\mathscr{P}_n^{\mathrm{C}})_{C_1 A} \circ (\mathscr{O})_A \circ (\mathscr{U}_1)_{AB} \left(|\chi_n\rangle\langle\chi_n|_C^t \otimes (|w\rangle\langle w| \otimes |\psi_0\rangle\langle\psi_0|)_{AB}\right) \end{aligned} \tag{37}$$

where $(\mathscr{P}_\tau^{\mathrm{C}})_{C_i A}$ means the $C_i{}^{\mathrm{th}}$ register controls the A^{th} register, and initially, the $C_i{}^{\mathrm{th}}$ register is the i^{th} copy of $|\chi_n\rangle$, and $\mathscr{O}$ is the CPTP version of the oracle $\mathcal{O}$.

Let $\rho_i(\mathscr{O})$ (resp. $\tilde{\rho}_i(\mathcal{O})$) be the state of the system during the algorithm M (resp. $\tilde{M}$) after the i^{th} use of the oracle. Let $\rho_0(\mathscr{O})$ (resp. $\tilde{\rho}_0(\mathcal{O})$) be the initial state of the respective algorithms. Then we will show that

$$\rho_i(\mathscr{P}_{\boldsymbol{\sigma}_{\mathrm{pre}(S_{\mathrm{pre}}(\sigma))}}) = \mathrm{tr}_{C_1,\ldots,C_t}\left(\tilde{\rho}_i(\mathcal{P}_\sigma)\right). \tag{38}$$

As a consequence of this, the probability distribution of measurement outcome of the two algorithms will be identical.

We prove this by induction. For the initial step, we have

$$\rho_0(\mathscr{P}_{\boldsymbol{\sigma}_{\mathrm{pre}(S_{\mathrm{pre}}(\sigma))}}) = |w\rangle\langle w| \otimes |\psi_0\rangle\langle\psi_0| \tag{39}$$

while

$$\begin{aligned} \mathrm{tr}_C(\tilde{\rho}_0(\mathcal{P}_\sigma)) &= \mathrm{tr}_C\left(|\chi_n\rangle\langle\chi_n|_C^t \otimes (|w\rangle\langle w| \otimes |\psi_0\rangle\langle\psi_0|)_{AB}\right) \\ &= |w\rangle\langle w| \otimes |\psi_0\rangle\langle\psi_0|. \end{aligned} \tag{40}$$

For the induction step, we need to show

$$\rho_k(\mathscr{P}_{\boldsymbol{\sigma}_{\mathrm{pre}}(S_{\mathrm{pre}}(\sigma))}) = \mathrm{tr}_{C_1,\ldots,C_t}\left(\tilde{\rho}_k(\mathcal{P}_\sigma)\right). \tag{41}$$

Because $\mathscr{P}_{\boldsymbol{\sigma}_{\mathrm{pre}}(S_{\mathrm{pre}}(\sigma))}$ has an equal probability of applying $\mathcal{P}_\sigma$ for each σ such that $S(\sigma) = S$, we have

$$\begin{aligned}\rho_k(\mathscr{P}_{\boldsymbol{\sigma}_{\mathrm{pre}}(S_{\mathrm{pre}}(\sigma))}) =& \mathscr{P}_{\boldsymbol{\sigma}_{\mathrm{pre}}(S_{\mathrm{pre}}(\sigma))}\left(\mathcal{U}_k\rho_{k-1}(\mathscr{P}_{\boldsymbol{\sigma}_{\mathrm{pre}}(S_{\mathrm{pre}}(\sigma))})\mathcal{U}_k^\dagger\right)\\ =& \frac{1}{N!(N^2-N)!}\sum_{i=1}^{N!(N^2-N)!}\mathcal{P}_{\tau_i}\mathcal{P}_\sigma\mathcal{U}_k\rho_{k-1}(\mathscr{P}_{\boldsymbol{\sigma}_{\mathrm{pre}}(S_{\mathrm{pre}}(\sigma))})\mathcal{U}_k^\dagger\mathcal{P}_\sigma^\dagger\mathcal{P}_{\tau_i}^\dagger. \end{aligned}\tag{42}$$

On the other hand

$$\begin{aligned}\mathrm{tr}_C\left(\tilde{\rho}_k(\mathcal{P}_\sigma)\right) =& \mathrm{tr}_C\left((\mathcal{P}_\tau^{\mathrm{C}})_{C_kA}\mathcal{P}_\sigma\mathcal{U}_k(\tilde{\rho}_{k-1}(\mathcal{P}_\sigma))\mathcal{U}_k^\dagger\mathcal{P}_\sigma^\dagger(\mathcal{P}_\tau^{\mathrm{C}})_{C_kA}^\dagger\right)\\ =& \frac{1}{N!(N^2-N)!}\sum_{i=1}^{N!(N^2-N)!}\mathcal{P}_{\tau_i}\mathcal{P}_\sigma\mathcal{U}_k\,\mathrm{tr}_C\left(\tilde{\rho}_{k-1}(\mathcal{P}_\sigma)\right)\mathcal{U}_k^\dagger\mathcal{P}_\sigma^\dagger\mathcal{P}_{\tau_i}^\dagger \end{aligned}\tag{43}$$

Now we need to show $\tilde{M}$ decides $L_{\boldsymbol{\mathcal{O}}}$ for a preimage-correct oracle $\boldsymbol{\mathcal{O}}$. Let's consider an input 1^n. Suppose $\mathcal{O}_n = \mathcal{P}_\sigma$, where $S_{\mathrm{pre}}(\sigma) \in \mathbf{S}^n_{\mathrm{even}}$. Then because M decides $L_{\boldsymbol{\mathcal{O}}}$ for any randomized-preimage-correct, there exists a witness w that depends only on $S_{\mathrm{pre}}(\sigma)$ such that when the oracle is $\mathscr{P}_{\boldsymbol{\sigma}_{\mathrm{pre}}(S_{\mathrm{pre}}(\sigma))}$ the output of M is 1 with probability at least $2/3$. Using the same witness w, $\tilde{M}$ will therefore produce output 1 with probability at least $2/3$.

Now consider an input 1^n such that $\mathcal{O}_n = \mathcal{P}_\sigma$ where $S_{\mathrm{pre}}(\sigma) \in \mathbf{S}^n_{\mathrm{odd}}$. Because M decides $L_{\boldsymbol{\mathcal{O}}}$ for a randomized-preimage-correct oracle $\boldsymbol{\mathcal{O}}$, when M is run with the oracle $\mathscr{P}_{\boldsymbol{\sigma}_{\mathrm{pre}}(S_{\mathrm{pre}}(\sigma))}$, for any witness w, M will output 1 with probability at most $1/3$. But because $\tilde{M}$ will have the same probability distribution of outcomes, this means that for any witness w to $\tilde{M}$, with oracle $\mathcal{P}_\sigma$, $\tilde{M}$ will output 1 with probability at most $1/3$. ◀

D Proofs of Lemmas 16 and 17

▶ **Lemma 16.** *Let $0 < \alpha < 1/2$ be a constant and $p(\cdot)$ be a polynomial function. Then there exists a positive integer $n^*(p,\alpha)$, such that for every $n > n^*(p,\alpha)$, and every subset family $\mathbf{S} \subset \mathbf{S}^n_{\mathrm{even}}$ such that $|\mathbf{S}| \geq |\mathbf{S}^n_{\mathrm{even}}|2^{-p(n)}$, there exists a subset family $\mathbf{S}_X \subset \mathbf{S}$ such that $\mathbf{S}_X$ is α-distributed. Furthermore the fixed subset S_{fixed} of $\mathbf{S}_X$ contains at most $N/3$ even elements.*

Proof. We prove the existence of $\mathbf{S}'$ by construction. Let $\mathbf{S}$ be any subset of $\mathbf{S}^n_{\mathrm{even}}$ such that $|\mathbf{S}| \geq |\mathbf{S}^n_{\mathrm{even}}|2^{-p(n)}$. We construct $\mathbf{S}_X$ using the following procedure, which we call the fixing procedure.

Fixing Procedure

1. Set $\mathbf{S}_X = \mathbf{S}$, and set $S_{\mathrm{fixed}} = \emptyset$.
2. **a.** Let $\nu(i)$ be the number of subsets $S \in \mathbf{S}_X$ such that $i \in S$.
   b. If there exists some element i for which

$$|\mathbf{S}_X| > \nu(i) \geq |\mathbf{S}_X|N^{-\alpha} \tag{44}$$

set $\mathbf{S}_X \leftarrow \{S : S \in \mathbf{S}_X \text{ and } i \in S\}$, set $S_{\mathrm{fixed}} \leftarrow S_{\mathrm{fixed}} \cup i$, and return to step 2(a). Otherwise exit the Fixing Procedure.

By construction, $\mathbf{S}_X$ will satisfy Definition 6. Now we need to check that the Fixing Procedure stops before fixing more than $N/3$ even items.

Let's suppose that at some point in the Fixing Procedure, for sets $\mathbf{S}_X$ and S_{fixed}, we have $N/3$ even items fixed. Suppose for contradiction, that at this point, there is some even element i^* such that i^* appears in greater than $N^{-\alpha}$ fraction of $S \in \mathbf{S}_X$. Let's also assume without loss of generality that $|S_{\text{fixed}} \cap \mathbb{Z}_{\text{odd}}| = k_{\text{odd}} \leq N/3$.

Let us look at the set

$$\mathbf{S}'' = \{S : S \in \mathbf{S}_X, i^* \in S\}. \tag{45}$$

Because $(S_{\text{fixed}} \cup i^*) \subset S$ for all $S \in \mathbf{S}''$, there are $N/3 - 1$ even elements that can be freely chosen for $S \in \mathbf{S}''$ and $N/3 - k_{\text{odd}}$ odd elements that can be freely chosen. Thus, we have

$$|\mathbf{S}''| \leq \binom{N^2/2 - N/3 - 1}{N/3 - 1}\binom{N^2/2 - k_{\text{odd}}}{N/3 - k_{\text{odd}}}. \tag{46}$$

By assumption

$$|\mathbf{S}''| \geq |\mathbf{S}_X| N^{-\alpha}, \tag{47}$$

so

$$\begin{aligned}
|\mathbf{S}_X| &\leq \binom{N^2/2 - N/3 - 1}{N/3 - 1}\binom{N^2/2 - k_{\text{odd}}}{N/3 - k_{\text{odd}}} N^{\alpha} \\
&\leq \binom{N^2/2}{N/3}\binom{N^2/2}{N/3} N^{\alpha} \\
&\leq (3Ne/2)^{2N/3} N^{\alpha} \\
&= 2^{2N/3(\log(3e/2) + \log N) + \log(N)\alpha} \\
&= 2^{O(N) + (2N/3)\log N}.
\end{aligned} \tag{48}$$

However, we can also bound the size of $\mathbf{S}_X$ from the Fixing Procedure. Notice that at every step of the Fixing Procedure, the size of $\mathbf{S}_X$ is reduced by at most a factor $N^{-\alpha}$. Since we are assuming $N/3$ even elements are in S_{fixed} and $k_{\text{odd}} \leq N/3$ odd elements are in S_{fixed}, the Fixing Procedure can reduce the original set family $\mathbf{S}$ by at most a factor $N^{-\alpha(2N/3)}$. Since $|\mathbf{S}| \geq |\mathbf{S}^n_{\text{even}}| 2^{-p(n)}$, we have that at this point in the Fixing Procedure

$$\begin{aligned}
|\mathbf{S}_X| &\geq |\mathbf{S}^n_{\text{even}}| 2^{-p(n)} N^{-\alpha(2N/3)} \\
&= \binom{N^2/2}{2N/3}\binom{N^2/2}{N/3} 2^{-p(n)} N^{-\alpha(2N/3)} \\
&\leq (3N/4)^{2N/3} (3N/4)^{N/3} 2^{-p(n)} N^{-\alpha(2N/3)} \\
&= 2^{2N/3(\log(3/4) + \log N) + N/3(\log(3/4) + \log N) - p(n) - \log(N)\alpha 2N/3} \\
&= 2^{N \log N(1 - 2\alpha/3) + N\log(3/4) - p(\log(N))} \\
&= 2^{-O(N) + N\log N(1 - 2\alpha/3)}.
\end{aligned} \tag{49}$$

Notice that as long as $\alpha < 1/2$, for large enough N (in particular, for $N > 2^{n^*}$ for some positive integer n^*, where n^* depends on α and $p(\cdot)$), the bound of Eq. (49) will be larger than the bound of Eq. (48), giving a contradiction. Therefore, our assumption, must have been false, and at this point in the Fixing Procedure, all even elements $i \in [N^2]/S_{\text{fixed}}$ will appear in at most a fraction $N^{-\alpha}$ of $S \in \mathbf{S}_X$. Thus at the next step of the Fixing Procedure, an even element will not be added to S_{fixed}, and the number of even elements in S_{fixed} will stay bounded by $N/3$. The same logic can be reapplied at future steps of the Fixing Procedure, even if additional odd items are added. ◀

▶ **Lemma 17.** *Let $\mathbf{S}_X$ be the α-distributed set created using the Fixing Procedure from Lemma 16, with fixed subset S_{fixed}. Let $\mathbf{S}_Y = \{S : S \in \mathbf{S}^n_{odd}, S_{fixed} \subset S\}$. Then we can construct an adversary bound to prove that for every quantum algorithm G, there exists permutations $\sigma_x, \sigma_y \in \boldsymbol{\sigma}^n$ with $S_{pre}(\sigma_x) \in \mathbf{S}_X$ and $S_{pre}(\sigma_y) \in \mathbf{S}_Y$, (that depend on G) such that, given oracle access to $\mathcal{P}_{\sigma_x}$ or $\mathcal{P}_{\sigma_y}$, G can not distinguish them with probability $\epsilon > .5$ without using $\left(1 - 2\sqrt{\epsilon(1-\epsilon)}\right) N^{\alpha/2}$ queries.*

Proof. Since $\mathbf{S}_X$ is α-distributed, there exists a set S_{fixed} of elements such that $S_{\text{fixed}} \subset S$ for all $S \in \mathbf{S}_X$, where S_{fixed} contains at most $N/3$ odd elements and at most $N/3$ even elements. (Otherwise Condition (2) of Definition 6 will not be satisfied.) We choose

$$\begin{aligned}\boldsymbol{\sigma}_Y &= \{\sigma : S_{\text{pre}}(\sigma) \in \mathbf{S}_Y\},\\ \boldsymbol{\sigma}_X &= \{\sigma : S_{\text{pre}}(\sigma) \in \mathbf{S}_X\}.\end{aligned} \tag{50}$$

We now define the relation $\mathbf{R}$ needed to apply our adversary bound. For each $(S_x, S_y) \in \mathbf{S}_X \times \mathbf{S}_Y$, we will create a one-to-one matching in $\mathbf{R}$ between the elements of $\boldsymbol{\sigma}_{\text{pre}}(S_x)$ and $\boldsymbol{\sigma}_{\text{pre}}(S_y)$. We first choose any element $\sigma^*_x \in \boldsymbol{\sigma}_{\text{pre}}(S_x)$. Then we choose a permutation $\sigma^*_y \in \boldsymbol{\sigma}_{\text{pre}}(S_y)$ such that

- $\forall j \in (S_x \cap S_y), \sigma^*_x(j) = \sigma^*_y(j)$,
- $\forall j \in [N^2] \setminus (S_x \cup S_y), \sigma^*_x(j) = \sigma^*_y(j)$,
- $\forall j \in S_x \setminus (S_x \cap S_y), \exists i \in S_y \setminus (S_x \cap S_y)$ such that $\sigma^*_x(j) = \sigma^*_y(i)$ and $\sigma^*_x(i) = \sigma^*_y(j)$.

Since every permutation corresponding to S_y is in $\boldsymbol{\sigma}_{\text{pre}}(S_y)$, there will always be such a σ^*_y that satisfies the above criterion. We choose $(\sigma^*_x, \sigma^*_y) \in \mathbf{R}$.

For $i \in [N!(N^2 - N)!]$ let $\boldsymbol{\tau}^n = \{\tau_i\}$ be the set of permutations on the elements of $[N^2]$ that do not mix the first N elements with the last $N^2 - N$ elements. By $\sigma_a \circ \sigma_b$, we mean apply first permutation σ_b, and then permutation σ_a. Notice that

$$\begin{aligned}\boldsymbol{\sigma}_{\text{pre}}(S_x) =& \{\tau \circ \sigma^*_x : \tau \in \boldsymbol{\tau}^n\}\\ \boldsymbol{\sigma}_{\text{pre}}(S_y) =& \{\tau \circ \sigma^*_y : \tau \in \boldsymbol{\tau}^n\}.\end{aligned} \tag{51}$$

Furthermore given $\tau \in \boldsymbol{\tau}^n$, we have

- $\forall j \in (S_x \cap S_y), \tau \circ \sigma^*_x(j) = \tau \circ \sigma^*_y(j)$,
- $\forall j \in [N^2] \setminus (S_x \cup S_y), \tau \circ \sigma^*_x(j) = \tau \circ \sigma^*_y(j)$,
- $\forall j \in S_x \setminus (S_x \cap S_y), \exists i \in S_y \setminus (S_x \cap S_y)$ such that $\tau \circ \sigma^*_x(j) = \tau \circ \sigma^*_y(i)$ and $\sigma \circ \sigma^*_x(i) = \sigma \circ \sigma^*_y(j)$.

For every $\tau \in \boldsymbol{\tau}^n$, we set $(\tau \circ \sigma^*_x, \tau \circ \sigma^*_y) \in \mathbf{R}$. In doing so, we create a one-to-one correspondance in $\mathbf{R}$ between the elements of $\boldsymbol{\sigma}_{\text{pre}}(S_x)$ and $\boldsymbol{\sigma}_{\text{pre}}(S_y)$. We then repeat this process for all pairs $(S_x, S_y) \in \mathbf{S}_X \times \mathbf{S}_Y$. The end result is the $\mathbf{R}$ that we will use.

Now we need to analyze the properties of this $\mathbf{R}$. Notice that each $\sigma_x \in \boldsymbol{\sigma}_X$ is paired to exactly one element of $\boldsymbol{\sigma}_{\text{pre}}(S_y)$ for each $S_y \in \mathbf{S}_Y$. Thus $m = |\mathbf{S}_Y|$. Likewise $m' = |\mathbf{S}_X|$.

Now consider $(\sigma_x, \sigma_y) \in \mathbf{R}$. We consider some element j such that $\sigma_x(j) \neq \sigma_y(j)$. We first consider the case that $j \in S_x$. We upper bound $l_{x,j}$, the number of $\sigma_{y'}$ such that $(\sigma_x, \sigma_{y'}) \in \mathbf{R}$ and $\sigma_{y'}(j) \neq \sigma_x(j)$. To simplify analysis, we use the simple upper bound $l_{x,j} \leq |\mathbf{S}_Y|$, which is sufficient for our purposes. Next we need to upper bound $l_{y,j}$, the number of $\sigma_{x'}$ such that $(\sigma_{x'}, \sigma_y) \in \mathbf{R}$ and $\sigma_{x'}(j) \neq \sigma_y(j)$. By our construction of $\mathbf{R}$, we have $j \notin S_y$. Also, by construction, if $j \notin S_y$, $\sigma_{x'}(j) \neq \sigma_y(j)$ if and only if $j \in S_{x'}$. Since σ_y is paired to only one element σ_x for each set S_x, $l_{y,j}$ is bounded by the number of sets

$S_x \in \mathbf{S}_X$ such that $j \in S_x$. Because $\mathbf{S}_X$ is α-distributed, at most a fraction $N^{-\alpha}$ of the sets of $\mathbf{S}_X$ can contain j, so $l_{y,j} \leq |\mathbf{S}_X| N^{-\alpha}$. In this case we have

$$l_{x,j} l_{y,j} \leq |\mathbf{S}_X||\mathbf{S}_Y| N^{-\alpha}. \tag{52}$$

We now consider the case that $j \in S_y$. We upper bound $l_{y,j}$, the number of $\sigma_{x'}$ such that $(\sigma_{x'}, \sigma_y) \in \mathbf{R}$ and $\sigma_{x'}(j) \neq \sigma_y(j)$. To simplify analysis, we use the upper bound of $l_{y,j} \leq |\mathbf{S}_X|$, which is sufficient for our analysis. Next we need to upper bound $l_{x,j}$, the number of $\sigma_{y'}$ such that $(\sigma_x, \sigma_{y'}) \in \mathbf{R}$ and $\sigma_{y'}(j) \neq \sigma_x(j)$. By our construction of $\mathbf{R}$, we have $j \notin S_x$. Also, by construction, if $j \notin S_x$, $\sigma_{y'}(j) \neq \sigma_x(j)$ if and only if $j \in S_{y'}$. Since σ_x is paired to only one element σ_x for each set S_x, $l_{x,j}$ is bounded by the number of sets $S_y \in \mathbf{S}_Y$ such that $j \in S_y$. Suppose S_{fixed} contains k_{even} even elements and k_{odd} odd elements. If j is odd, we have

$$\begin{aligned} l_{x,j} &= \binom{N^2/2 - k_{\text{odd}} - 1}{2N/3 - k_{\text{odd}} - 1} \binom{N^2/2 - k_{\text{even}}}{N/3 - k_{\text{even}}} \\ &\leq \frac{2N/3}{N^2/2 - N/3} |\mathbf{S}_Y|, \end{aligned} \tag{53}$$

while if j is even (in that case, we must have $k_{\text{even}} < N/3$), we have

$$\begin{aligned} l_{x,j} &= \binom{N^2/2 - k_{\text{odd}}}{2N/3 - k_{\text{odd}}} \binom{N^2/2 - k_{\text{even}} - 1}{N/3 - k_{\text{even}} - 1} \\ &\leq \frac{N/3}{N^2/2 - N/3} |\mathbf{S}_Y|, \end{aligned} \tag{54}$$

where we've used that

$$|\mathbf{S}_Y| = \binom{N^2/2 - k_{\text{odd}}}{2N/3 - k_{\text{odd}}} \binom{N^2/2 - k_{\text{even}}}{N/3 - k_{\text{even}}}. \tag{55}$$

Therefore in this case, we have

$$l_{x,j} l_{y,j} = |\mathbf{S}_X||\mathbf{S}_Y| O(N^{-1}). \tag{56}$$

Looking at Eq. (52) and Eq. (56), we see that because $\alpha < 1$, the bound of Eq. (52) dominates, and so we have that

$$\sqrt{\frac{mm'}{l_{x,j} l_{y,j}}} \geq \sqrt{\frac{|\mathbf{S}_X||\mathbf{S}_Y|}{|\mathbf{S}_X||\mathbf{S}_Y| N^{-\alpha}}} = N^{\alpha/2}. \tag{57}$$

Using the contrapositive of Lemma 2, if an algorithm G makes less than q queries to an oracle $\mathcal{O}_{\sigma_x}$ where σ_x is promised to be in $\boldsymbol{\sigma}_X$ or $\boldsymbol{\sigma}_Y$, there exists at least one element of $\boldsymbol{\sigma}_X$ and one element of $\boldsymbol{\sigma}_Y$ such that the probability of distinguishing between the corresponding oracles less than is $1/2 + \epsilon$, where

$$\frac{1}{2} \sqrt{2 N^{-\alpha/2} q} > \epsilon. \tag{58}$$

Equivalently, there exists at least one element of $\boldsymbol{\sigma}_X$ and one element of $\boldsymbol{\sigma}_Y$ such that in order for $\mathcal{A}$ to distinguish them with constant bias, one requires $\Omega(N^{\alpha/2})$ queries. ◀

Timed Network Games with Clocks

Guy Avni[1]
IST Austria, Klosterneuburg, Austria
guy.avni@ist.ac.at

Shibashis Guha[2]
Université Libre de Bruxelles, Brussels, Belgium
shibashis.guha@ulb.ac.be

Orna Kupferman[3]
Hebrew University, Jerusalem, Israel
orna@cs.huji.ac.il

Abstract

Network games are widely used as a model for selfish resource-allocation problems. In the classical model, each player selects a path connecting her source and target vertices. The cost of traversing an edge depends on the *load*; namely, number of players that traverse it. Thus, it abstracts the fact that different users may use a resource at different times and for different durations, which plays an important role in determining the costs of the users in reality. For example, when transmitting packets in a communication network, routing traffic in a road network, or processing a task in a production system, actual sharing and congestion of resources crucially depends on time.

In [13], we introduced *timed network games*, which add a time component to network games. Each vertex v in the network is associated with a cost function, mapping the load on v to the price that a player pays for staying in v for one time unit with this load. Each edge in the network is guarded by the time intervals in which it can be traversed, which forces the players to spend time in the vertices. In this work we significantly extend the way time can be referred to in timed network games. In the model we study, the network is equipped with *clocks*, and, as in timed automata, edges are guarded by constraints on the values of the clocks, and their traversal may involve a reset of some clocks. We argue that the stronger model captures many realistic networks. The addition of clocks breaks the techniques we developed in [13] and we develop new techniques in order to show that positive results on classic network games carry over to the stronger timed setting.

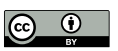
2012 ACM Subject Classification Theory of computation → Algorithmic game theory, Theory of computation → Network formation, Theory of computation → Timed and hybrid models

Keywords and phrases Network games, Timed automata, Nash equilibrium, Equilibrium inefficiency

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.23

Related Version A full version of the paper is available at [14], `http://arxiv.org/abs/1808.04882`.

[1] Supported by the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE/SHiNE), Z211-N23 (Wittgenstein Award), and M2369-N33 (Meitner fellowship).
[2] Supported partially by the ARC project "Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond" (Fédération Wallonie-Bruxelles).
[3] Supported by the European Research Council (FP7/2007-2013) / ERC grant agreement no 278410.



1 Introduction

Network games (NGs, for short) [10, 48, 49] constitute a well studied model of non-cooperative games. The game is played among selfish players on a network, which is a directed graph. Each player has a source and a target vertex, and a strategy is a choice of a path that connects these two vertices. The cost a player pays for an edge depends on the *load* on it, namely the number of players that use the edge, and the total cost is the sum of costs of the edges she uses. In *cost-sharing* games, load has a positive effect on cost: each edge has a cost and the players that use it split the cost among them. Then, in *congestion* games[4], load has a negative effect on cost: each edge has a non-decreasing *latency function* that maps the load on the edge to its cost.

One limitation of NGs is that the cost of using a resource abstracts the fact that different users may use the resource at different times and for different durations. This is a real limitation, as time plays an important role in many real-life settings. For example, in a road or a communication system, congestion only affects cars or messages that use a road or a channel simultaneously. We are interested in settings in which congestion affects the quality of service (QoS) or the way a price is shared by entities using a resource at the *same time* (rather than affecting the travel time). For example, discomfort increases in a crowded train (in congestion games) or price is shared by the passengers in a taxi (in cost-sharing games).

The need to address temporal behaviors has attracted a lot of research in theoretical computer science. Formalisms like temporal logic [46] enable the specification of the temporal ordering of events. Its refinement to formalisms like real-time temporal logic [7], interval temporal logic [42], and timed automata (TAs, for short) [6] enables the specification of *real-time* behaviors. Extensions of TAs include *priced timed automata* (PTAs, for short) that assign costs to real-time behaviors. Thus, PTAs are suitable for reasoning about quality of real-time systems. They lack, however, the capability to reason about multi-agent systems in which the players' choices affect the incurred costs.

We study *timed network games* (TNGs, for short) – a new model that adds a time component to NGs. A TNG is played on a *timed-network* in which edges are labeled by *guards* that specify time restrictions on when the edge can be traversed. Similar to NGs, each player has a source and target vertex, but a strategy is now a *timed path* that specifies, in addition to which vertices are traversed, the amount of time that is spent in each vertex. Players pay for staying in vertices, and the cost of staying in a vertex v in a time interval $I \subseteq \mathbb{R}_{\geq 0}$ is affected by the load in v during I. In [13], we studied a class of TNGs that offered a first extension of NGs to a timed variant in which the reference to time is restricted: the guards on the edges refer only to *global* time, i.e., the time that has elapsed since the beginning of the game. In the model in [13], it is impossible to refer to the duration of certain events that occur during the game, for example, it is not possible to express constraints that require staying exactly one time unit in a vertex. Accordingly, we refer to that class as *global TNGs* (GTNGs, for short).

In this work, we significantly extend the way time can be referred to in TNGs. We do this by adding *clocks* that may be reset along the edges, and by allowing the guards on the edges to refer to the values of all clocks. GTNGs can be viewed as a fragment in which there is only a single clock that is never reset. We demonstrate our model in the following example.

[4] The name *congestion games* is sometimes used to refer to games with general latency functions. We find it more appropriate to use it to refer to games with non-decreasing functions.

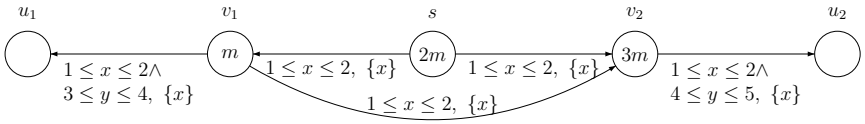


Figure 1 A congestion TNG.

▶ **Example 1.** Consider a setting in which messages are sent through a network of routers. Messages are owned by selfish agents who try to avoid congested routes, where there is a greater chance of loss or corruption. The owners of the messages decide how much time they spend in each router. Using TNGs, we can model constraints on these times, as well as constraints on global events, in particular, arrival time. Note that in some applications, c.f., advertising or security, messages need to patrol the network with a lower bound on their arrival time.

Consider the TNG appearing in Figure 1. The vertices in the TNG model the routers. There are two players that model two agents, each sending a message. The source of both messages is s and the targets are u_1 and u_2, for messages 1 and 2, respectively. The latency functions are described in the vertices, as a function of the load m; e.g., the latency function in v_2 is $\ell_{v_2}(m) = 3m$. Thus, when a single message stays in v_2 the cost for each time unit is 3, and when the two messages visit v_2 simultaneously, the cost for each of them is 6 per unit time. The network has two clocks, x and y. Clock x is reset in each transition and thus is used to impose restrictions on the time that can be spent in each router: since all transitions can be taken when $1 \leq x \leq 2$, a message stays between 1 and 2 time units in a router. Clock y is never reset, thus it keeps track of the global time. The guards on clock y guarantee that message 1 reaches its destination by time 4 but not before time 3 and message 2 reaches its destination by time 5 but not before time 4.

Suppose the first agent chooses the timed path $(s, 2), (v_1, 1), u_1$, thus message 1 stays in s for two time units and in v_1 for one time unit before reaching its destination u_1. Suppose the second agent chooses the path $(s, 2), (v_1, 2), (v_2, 1), u_2$. Note that crossing an edge is instantaneous. Since both messages stay in the same vertices during the intervals $I_1 = [0, 2]$ and $I_2 = [2, 3]$, the load in the corresponding vertices is 2. During interval I_1, each of the agents pays $|I_1| \cdot \ell_s(2) = 2 \cdot 4$ and during I_2, each pays $|I_2| \cdot \ell_{v_1}(2) = 1 \cdot 2$. Message 2 stays in v_1 alone during the interval $[3, 4]$ and in v_2 during the interval $[4, 5]$, for which it pays 1 and 3, respectively. The total costs are thus 10 and 14.

Before we elaborate on our contribution, let us survey relevant works, namely, extensions of NGs with temporal aspects and extensions of timed-automata to games. Extensions of NGs that involve reasoning about time mostly study a cost model in which the players try to minimize the time of arrival at their destinations (c.f., [36, 39, 47, 45]), where, for example, congestion affects the duration of crossing an edge. These works are different from ours since we consider a QoS cost model. An exception is [36], which studies the QoS costs. A key difference in the models is that there, time is discrete and the players have finitely many strategies. Thus, reductions to classical resource allocation games is straightforward while for TNGs it is not possible, as we elaborate below. Games on timed automata were first studied in [11] in which an algorithm to solve timed games with timed reachability objective was given. The work was later generalized and improved [4, 20, 35, 23]. Average timed games, games with parity objectives, mean-payoff games and energy games have also been studied in the context of timed automata [2, 37, 27, 21, 34]. All the timed games above are two-player zero-sum ongoing games. Prices are fixed and there is no notion of load. Also, the questions

studied on these games concern their decidability, namely finding winners and strategies for them. TNGs are not zero-sum games, so winning strategies do not exist. Instead, the problems we study here concern rationality and stability.

The first question that arises in the context of non-zero-sum games is the existence of *stable outcomes*. In the context of NGs, the most prominent stability concept is that of a (pure) *Nash equilibrium* (NE, for short) [43] – a profile such that no player can decrease her cost by unilaterally deviating from her current strategy.[5] Decentralized decision-making may lead to solutions that are sub-optimal for the society as a whole. The standard measures to quantify the inefficiency incurred due to selfish behavior is the *price of stability* (PoS) [10] and the *price of anarchy* (PoA) [38]. In both measures we compare against the *social optimum* (SO, for short), namely a profile that minimizes the sum of costs of all players. The PoS (PoA, respectively) is the best-case (worst-case) inefficiency of an NE; that is, the ratio between the cost of a best (worst) NE and the SO.

The picture of stability and equilibrium inefficiency for standard NGs is well understood. Every NG has an NE, and in fact these games are *potential games* [48], which have the following stronger property: a *best response sequence* is a sequence of profiles $P_1, P_2, \ldots$ such that, for $i \geq 1$, the profile P_{i+1} is obtained from P_i by letting some player deviate and decrease her personal cost. In finite potential games, every best-response sequence converges to an NE. For k-player cost-sharing NGs, the PoS and PoA are $\log k$ and k, respectively [10]. For congestion games with affine cost functions, PoS ≈ 1.577 [29, 1] and PoA $= \frac{5}{2}$ [30].

In [13], we showed that these positive results carry over to GTNGs. A key technical feature of GTNGs is that since guards refer to global time, it is easy to find an upper bound T on the time by which all players reach their destinations. Proving existence of NE follows from a reduction to NGs, using a zone-like structure [5, 18]. The introduction of clocks with resets breaks the direct reduction to NGs and questions the existence of a bound by which the players arrive at their destinations.[6] To see the difficulty in finding such a bound, consider, for example, a cost-sharing game in which all players, on their paths to their targets, need to stay for one time unit in a "gateway" vertex v that costs 1 (see details in Section 6). Assume also that, for $1 \leq i \leq k$, Player i can only reach v in times that are multiples of p_i, for relatively prime numbers $p_1, \ldots, p_k$. The SO is obtained when all players synchronize their visits to v, and such a synchronization forces them to wait till time $p_1 \cdot \ldots \cdot p_k$, which is exponential in the TNG.

The lack of an upper bound on the global time in TNGs demonstrates that we need a different approach to obtain positive results for general TNGs. We show that TNGs are guaranteed to have an NE. Our proof uses a combination of techniques from real-time models and resource allocation games. Recall that a PTA assigns a price to a timed word. We are able to reduce the *best-response* and the *social-optimum* problems to and from the problem of finding cheapest runs in PTAs [19], showing that the problems are PSPACE-complete. Next, we show that TNGs are potential games. Note that since players have uncountably many strategies, the fact that TNGs are potential games does not immediately imply existence of an NE, as a best-response sequence may not be finite. We show that there is a best-response sequence that terminates in an NE. For this, we first need to show the existence of an integral best-response, which is obtained from the reduction to PTAs. Finally, given a TNG, we find a time T such that there exists an NE in which all players reach their destination by time T.

Due to lack of space, some of the proofs appear in the full version [14].

---

[5] Throughout this paper, we consider *pure* strategies, as is the case for the vast literature on NGs.

[6] In the full version we show that even with an upper bound on time, a reduction from TNGs to NGs is not likely.

2 Preliminaries

2.1 Resource allocation games and network games

For $k \in \mathbb{N}$, let $[k] = \{1, \ldots, k\}$. A *resource allocation game* (RAG, for short) is $R = \langle k, E, \{\Sigma_i\}_{i\in[k]}, \{\ell_e\}_{e\in E}\rangle$, where $k \in \mathbb{N}$ is the number of players; E is a set of resources; for $i \in [k]$, the set strategies of Player i is $\Sigma_i \subseteq 2^E$; and, for $e \in E$, the *latency function* $\ell_e : [k] \to \mathbb{Q}_{\geq 0}$ maps a load on e to its cost under this load. A *profile* is a choice of a strategy for each player. The set of profiles of R is $\mathit{profiles}(R) = \Sigma_1 \times \ldots \times \Sigma_k$. For $e \in E$, we define the *load* on e in a profile $P = \langle \sigma_1, \ldots, \sigma_k\rangle$, denoted $\mathit{load}_P(e)$, as the number of players using e in P, thus $\mathit{load}_P(e) = |\{i \in [k] : e \in \sigma_i\}|$. The cost a player pays in profile P, denoted $\mathit{cost}_i(P)$, depends on the choices of the other players. We define $\mathit{cost}_i(P) = \sum_{e\in\sigma_i} \ell_e(\mathit{load}_P(e))$.

Network games (NGs, for short) can be viewed as a special case of RAGs where strategies are succinctly represented by means of paths in graphs. An NG is $\mathcal{N} = \langle k, V, E, \{\langle s_i, u_i\rangle\}_{i\in[k]}, \{\ell_e\}_{e\in E}\rangle$, where $\langle V, E\rangle$ is a directed graph; for $i \in [k]$, the vertices s_i and u_i are the *source* and *target* vertices of Player i; and the latency functions are as in RAGs. The set of strategies for Player i is the set of simple paths from s_i to u_i in $\mathcal{N}$. Thus, in NGs, the resources are the edges in the graph.

We distinguish between two types of latency functions. In *cost-sharing games*, the players that visit a vertex share its cost equally. Formally, every $e \in E$ has a cost $c_e \in \mathbb{Q}_{\geq 0}$ and its latency function is $\ell_e(l) = \frac{c_e}{l}$. Note that these latency functions are decreasing, thus the load has a positive effect on the cost. In contrast, in *congestion games*, the cost functions are non-decreasing and so the load has a negative effect on the cost. Typically, the latency functions are restricted to simple functions such as linear latency functions, polynomials, and so forth.

2.2 Timed networks and timed network games

A *clock* is a variable that gets values from $\mathbb{R}_{\geq 0}$ and whose value increases as time elapses. A *reset* of a clock x assigns value 0 to x. A *guard* over a set C of clocks is a conjunction of *clock constraints* of the form $x \sim m$, for $x \in C$, $\sim \in \{\leq, =, \geq\}$, and $m \in \mathbb{N}$. Note that we disallow guards that use the operators $<$ and $>$ (see Remark 4). A guard of the form $\bigwedge_{x\in C} x \geq 0$ is called *true*. The set of guards over C is denoted $\Phi(C)$. A *clock valuation* is an assignment $\kappa : C \to \mathbb{R}_{\geq 0}$. A clock valuation κ *satisfies* a guard g, denoted $\kappa \models g$, if the expression obtained from g by replacing each clock $x \in C$ with the value $\kappa(x)$ is valid.

A *timed network* is a tuple $\mathcal{A} = \langle C, V, E\rangle$, where C is a set of clocks, V is a set of vertices, and $E \subseteq V \times \Phi(C) \times 2^C \times V$ is a set of directed edges in which each edge e is associated with a guard $g \in \Phi(C)$ that should be satisfied when e is traversed and a set $R \subseteq C$ of clocks that are reset along the traversal of e.

When traversing a path in a timed network, time is spent in vertices, and edges are traversed instantaneously. Accordingly, a *timed path* in $\mathcal{A}$ is a sequence $\eta = \langle \tau_1, e_1\rangle, \ldots, \langle \tau_n, e_n\rangle \in (\mathbb{R}_{\geq 0} \times E)^*$, describing edges that the path traverses along with their traversal times. The timed path η is *legal* if the edges are successive and the guards associated with them are satisfied. Formally, there is a sequence $\langle v_0, t_0\rangle, \ldots, \langle v_{n-1}, t_{n-1}\rangle, v_n \in (V \times \mathbb{R}_{\geq 0})^* \cdot V$, describing the vertices that η visits and the time spent in these vertices, such that for every $1 \leq j \leq n$, the following hold: (1) $t_{j-1} = \tau_j - \tau_{j-1}$, with $\tau_0 = 0$, (2) there is $g_j \in \Phi(C)$ and $R_j \subseteq C$, such that $e_j = \langle v_{j-1}, g_j, R_j, v_j\rangle$, (3) there is a clock valuation κ_j that describes the values of the clocks before the incoming edge to vertex v_j is traversed. Thus, $\kappa_1(x) = t_0$, for all $x \in C$, and for $1 < j \leq n$, we distinguish between clocks that are reset when e_{j-1}

is traversed and clocks that are not reset: for $x \in R_{j-1}$, we define $\kappa_j(x) = t_{j-1}$, and for $x \in (C \setminus R_{j-1})$, we define $\kappa_j(x) = \kappa_{j-1}(x) + t_{j-1}$, and (4) for every $1 \leq j \leq n$, we have that $\kappa_j \models g_j$. We sometimes refer to η also as the sequence $\langle v_0, t_0\rangle, \ldots, \langle v_{n-1}, t_{n-1}\rangle, v_n$.

Consider a finite set $T \subseteq \mathbb{R}_{\geq 0}$ of time points. We say that a timed path η is a *T-path* if all edges in η are taken at times in T. Formally, for all $1 \leq j \leq n$, we have that $\tau_j \in T$. We refer to the time at which η *ends* as the time τ_n at which the destination is reached. We say that η is *integral* if $T \subseteq \mathbb{N}$.

A *timed network game* (TNG, for short) extends an NG by imposing constraints on the times at which edges may be traversed. Formally, $\mathcal{T} = \langle k, C, V, E, \{\ell_v\}_{v\in V}, \langle s_i, u_i\rangle_{i\in[k]}\rangle$ includes a set C of clocks, and $\langle C, V, E\rangle$ is a timed network. Recall that while traversing a path in a timed network, time is spent in vertices. Accordingly, the latency functions now apply to vertices, thus $\ell_v : [k] \to \mathbb{Q}_{\geq 0}$ maps a load on vertex v to its cost under this load. Traversing an edge is instantaneous and is free of charge. A strategy for Player i, for $i \in [k]$, is then a legal timed path from s_i to u_i. We assume all players have at least one strategy.

▶ Remark. A possible extension of TNGs is to allow costs on edges. Since edges are traversed instantaneously, these costs would not be affected by load. Such an extension does not affect our results and we leave it out for sake of simplicity. Another possible extension is allowing strict time guards, which we discuss in Remark 4.

The cost Player i pays in profile P, denoted $cost_i(P)$, depends on the vertices in her timed path, the time spent on them, and the load during the visits. In order to define the cost formally, we need some definitions. For a finite set $T \subseteq \mathbb{R}_{\geq 0}$ of time points, we say that a timed path is a T-strategy if it is a T-path. Then, a profile P is a T-profile if it consists only of T-strategies. Let $t_{max} = \max(T)$. For $t \in T$ such that $t < t_{max}$, let $next_T(t)$ be the minimal time point in T that is strictly larger than t. We partition the interval $[0, t_{\max}]$ into a set Υ of sub-intervals $[m, next_T(m)]$ for every $m \in (T \cup \{0\}) \setminus \{t_{max}\}$. We refer to the sub-intervals in Υ as *periods*. Suppose T is the minimal set such that P is a T-profile. Note that Υ is the coarsest partition of $[0, t_{\max}]$ into periods such that no player crosses an edge within a period in Υ. We denote this partition by Υ_P.

For a player $i \in [k]$ and a period $\gamma \in \Upsilon_P$, let $visits_P(i, \gamma)$ be the vertex that Player i visits during period γ. That is, if $\pi_i = \langle v_0^i, t_0^i\rangle, \ldots, \langle v_{n_i-1}^i, t_{n_i-1}^i\rangle, v_{n_i}^i$ is a legal timed path that is a strategy for Player i and $\gamma = [m_1, m_2]$, then $visits_P(i, \gamma)$ is the vertex v_j^i for the index $1 \leq j < n_i$ such that $\tau_j^i \leq m_1 \leq m_2 \leq \tau_{j+1}^i$, and $visits_P(i, \gamma)$ is the vertex v_0^i if $0 = m_1 \leq m_2 \leq \tau_1^i$. Note that since P is a T-profile, for each period $\gamma \in \Upsilon_P$, the number of players that stay in each vertex v during γ is fixed. Let $load_P(v, \gamma)$ denote this number. Formally $load_P(v, \gamma) = |\{i : visits_P(i, \gamma) = v\}|$. Finally, for a period $\gamma = [m_1, m_2]$, let $|\gamma| = m_2 - m_1$ be the *duration* of γ. Suppose Player i's path ends at time τ^i. Let $\Upsilon_P^i \subseteq \Upsilon_P$ denote the periods that end by time τ_i.

Recall that the latency function $\ell_v : [k] \longrightarrow \mathbb{Q}_{\geq 0}$ maps the number of players that simultaneously visit vertex v to the price that each of them pays per time unit. If $visits_P(i, \gamma) = v$, then the cost of Player i in P, over the period γ is $cost_{\gamma,i}(P) = \ell_v(load_P(v, \gamma)) \cdot |\gamma|$. We define $cost_i(P) = \sum_{\gamma\in\Upsilon_P^i} cost_{\gamma,i}(P)$. The cost of the profile P, denoted $cost(P)$, is the total cost incurred by all the players, i.e., $cost(P) = \sum_{i=1}^{k} cost_i(P)$.

A T-strategy is called an *integral strategy* when $T \subseteq \mathbb{N}$, and similarly for *integral* profile.

A profile $P = \langle \pi_1, \ldots, \pi_k\rangle$ is said to *end* by time τ if for each $i \in [k]$, the strategy π_i ends by time τ. Consider a TNG $\mathcal{T}$ that has a cycle such that a clock x of $\mathcal{T}$ is reset on the cycle. It is not difficult to see that this may lead to $\mathcal{T}$ having infinitely many integral profiles that end by different times. A TNG $\mathcal{T}$ is called *global* if it has a single clock x that is never reset. We use GTNG to indicate that a TNG is global.

As in RAGs, we distinguish between cost-sharing TNGs that have cost-sharing latency functions and congestion TNGs in which the latency functions are non-decreasing.

2.3 Stability and efficiency

Consider a game G. For a profile P and a strategy π of player $i \in [k]$, let $P[i \leftarrow \pi]$ denote the profile obtained from P by replacing the strategy of Player i in P by π. A profile P is said to be a *(pure) Nash equilibrium* (NE) if none of the players in $[k]$ can benefit from a unilateral deviation from her strategy in P to another strategy. Formally, for every Player i and every strategy π for Player i, it holds that $cost_i(P[i \leftarrow \pi]) \geq cost_i(P)$.

A *social optimum* (SO) of a game G is a profile that attains the infimum cost over all profiles. We denote by $SO(G)$ the cost of an SO profile; i.e., $SO(G) = \inf_{P \in profiles(G)} cost(P)$. It is well known that decentralized decision-making may lead to sub-optimal solutions from the point of view of the society as a whole. We quantify the inefficiency incurred due to self-interested behavior by the *price of anarchy* (PoA) [38, 44] and *price of stability* (PoS) [10] measures. The PoA is the worst-case inefficiency of a Nash equilibrium, while the PoS measures the best-case inefficiency of a Nash equilibrium. Note that unlike resource allocation games in which the set of profiles is finite, in TNGs there can be uncountably many NEs, so both PoS and PoA need to be defined using infimum/supremum rather than min/max. Formally,

▶ **Definition 2.** Let $\mathcal{G}$ be a family of games, and let $G \in \mathcal{G}$ be a game in $\mathcal{G}$. Let $\Gamma(G)$ be the set of Nash equilibria of the game G. Assume that $\Gamma(G) \neq \emptyset$.

- The *price of anarchy* of G is $PoA(G) = \sup_{P \in \Gamma(G)} cost(P)/SO(G)$. The *price of anarchy* of the family of games $\mathcal{G}$ is $PoA(\mathcal{G}) = sup_{G \in \mathcal{G}} PoA(G)$.
- The *price of stability* of G is $PoS(G) = \inf_{P \in \Gamma(G)} cost(P)/SO(G)$. The *price of stability* of the family of games $\mathcal{G}$ is $PoS(\mathcal{G}) = sup_{G \in \mathcal{G}} PoS(G)$.

3 The Best-Response and the Social-Optimum Problems

Consider a TNG $\mathcal{T} = \langle k, C, V, E, \{\ell_v\}_{v \in V}, \langle s_i, u_i \rangle_{i \in [k]} \rangle$. In the *best-response problem* (BR problem, for short), we ask how a player reacts to a choice of strategies of the other players. Formally, let $\pi_1, \ldots, \pi_{k-1}$ be a choice of integral[7] strategies for Players $1, \ldots, k-1$ in $\mathcal{T}$. We look for a strategy π_k that minimizes $cost_k(\langle \pi_1, \ldots, \pi_k \rangle)$. The choice of allowing Player k to react is arbitrary and is done for convenience of notation. In the *social optimum problem* (SOPT problem, for short), we seek a profile that maximizes the social welfare, or in other words, minimizes the sum of players' costs.

In this section we describe *priced timed automata* (PTAs, for short) [9, 17] and show that while they are different from TNGs both in terms of the model and the questions asked on it, they offer a useful framework for reasoning about TNGs. In particular, we solve the BR and SOPT problems by reductions to problems about PTAs.

3.1 From TNGs to priced timed automata

A PTA [9, 17] is $\mathcal{P} = \langle C, V, E, \{r_v\}_{v \in V} \rangle$, where $\langle C, V, E \rangle$ is a timed network and $r_v \in \mathbb{Q}_{\geq 0}$ is the *rate* of vertex $v \in V$. Intuitively, the rate r_v specifies the cost of staying in v for a duration of one time unit. Thus, a timed path $\eta = \langle v_0, t_0 \rangle, \ldots, \langle v_n, t_n \rangle, v_{n+1}$ in a PTA has a

[7] We choose integral strategies since strategies with irrational times cannot be represented as part of the input; for strategies that use rational times, the best response problem can be solved with little modification in the proof of Theorem 4.

price, denoted $price(\eta)$, which is $\sum_{0 \leq j \leq n} r_v \cdot t_v$. The size of $\mathcal{P}$ is $|V| + |E|$ plus the number of bits needed in the binary encoding of the numbers appearing in guards and rates in $\mathcal{P}$. [8]

Consider a PTA $\mathcal{P}$ and two vertices s and u. Let $paths(s, u)$ be the set of timed paths from s to u. We are interested in cheapest timed paths in $paths(s, u)$. A priori, there is no reason to assume that the minimal price is attained, thus we are interested in the optimal price, denoted $opt(s, u)$, which we define to be $\inf\{price(\eta) : \eta \in paths(s, u)\}$. The corresponding decision problem, called the *cost optimal reachability problem* (COR, for short) takes in addition a threshold μ, and the goal is to decide whether $opt(s, t) \leq \mu$. Recall that we do not allow the guards to use the operators $<$ and $>$.

▶ **Theorem 3.** *[19, 32] The COR problem is PSPACE-complete for PTAs with two or more clocks. Moreover, the optimal price is attained by an integral path, i.e., there is an integral path $\eta \in paths(s, u)$ with $price(\eta) = opt(s, u)$.*

In Sections 3.2 and 3.3 below, we reduce problems on TNGs to problems on PTAs. The reductions allow us to obtain properties on strategies and profiles in TNGs using results on PTAs, which we later use in combination with techniques for NGs in order to solve problems on TNGs.

3.2 The best-response problem

▶ **Theorem 4.** *Consider a TNG $\mathcal{T}$ with n clocks and integral strategies $\pi_1, \ldots, \pi_{k-1}$ for Players $1, \ldots, k-1$. There is a PTA $\mathcal{P}$ with $n+1$ clocks and two vertices v and u such that there is a one-to-one cost-preserving correspondence between strategies for Player k in $\mathcal{T}$ and timed paths from v to u: for every strategy π_k in $\mathcal{T}$ and its corresponding path η in $\mathcal{P}$, we have $cost_k(\langle \pi_1, \ldots, \pi_k \rangle) = price(\eta)$.*

Proof. We describe the intuition of the reduction and the details can be found in the full version. Consider a TNG $\mathcal{T} = \langle k, V, E, C, \{\ell_v\}_{v \in V}, \langle s_i, u_i \rangle_{i \in [k]} \rangle$, where $C = \{x_1, \ldots, x_m\}$. Let $Q = \langle \pi_1, \ldots, \pi_{k-1} \rangle$ be a choice of timed paths for Players $1, \ldots, k-1$. Note that Q can be seen as a profile in a game that is obtained from $\mathcal{T}$ by removing Player k, and we use the definitions for profiles on Q in the expected manner. Let $T \subseteq \mathbb{Q}$ be the minimal set of time points for which all the strategies in Q are T-strategies. Consider two consecutive time points $a, b \in T$, i.e., there is no $c \in T$ with $a < c < b$. Then, there are players that cross edges at times a and b, and no player crosses an edge at time points in the interval (a, b). Moreover, let t_{max} be the latest time in T, then t_{max} is the latest time at which a player reaches her destination. Let Υ_Q be a partition of $[0, t_{max}]$ according to T. We obtain Υ'_Q from Υ_Q by adding the interval $[t_{\max}, \infty)$.

A key observation is that the load on all the vertices is unchanged during every interval in Υ'_Q. For a vertex $v \in V$ and $\delta \in \Upsilon_Q$, the cost Player k pays per unit time for using v in the interval δ is $\ell_v(load_Q(v, \delta) + 1)$. On the other hand, since all $k-1$ players reach their destination by time t_{max}, the load on v after t_{max} is 0, and the cost Player k pays for using it then is $\ell_v(1)$.

The PTA $\mathcal{P}$ that we construct has $|\Upsilon'_Q|$ copies of $\mathcal{T}$, thus its vertices are $V \times \Upsilon'_Q$. Let $\delta_0 = [0, b] \in \Upsilon'_Q$ be the first interval. We consider paths from the vertex $v = \langle s_k, \delta_0 \rangle$, which is the copy of Player k's source in the first copy of $\mathcal{T}$, to a target u, which is a new vertex we add and whose only incoming edges are from vertices of the form $\langle u_k, \delta \rangle$, namely,

[8] In general, PTAs have rates on transitions and strict time guards, which we do not need here.

the copies of the target vertex u_k of Player k. We construct $\mathcal{P}$ such that each such path η from v to u in $\mathcal{P}$ corresponds to a legal strategy π_k for Player k in $\mathcal{T}$, and such that $cost_k(\langle \pi_1, \ldots, \pi_{k-1}, \pi_k \rangle) = price(\eta)$. The main difference between the copies are the vertices' costs, which depend on the load as in the above. We refer to the n clocks in $\mathcal{T}$ as *local clocks*. In each copy of $\mathcal{P}$, we use the local clocks and their guards in $\mathcal{T}$ as well as an additional *global* clock that is never reset to keep track of global time. Let $\delta = [a, b] \in \Upsilon_Q$ and $\delta' = [b, c] \in \Upsilon'_Q$ be the following interval. Let $\mathcal{T}_\delta$ and $\mathcal{T}_{\delta'}$ be the copies of $\mathcal{T}$ that corresponds to the respective intervals. The local clocks guarantee that a path in $\mathcal{T}_\delta$ is a legal path in $\mathcal{T}$. The global clock allows us to make sure that (1) proceeding from $\mathcal{T}_\delta$ to $\mathcal{T}_{\delta'}$ can only occur precisely at time b, and (2) proceeding from $\langle u_k, \delta \rangle$ in $\mathcal{T}_\delta$ to the target u can only occur at a time in the interval δ. ◀

We conclude with the computational complexity of the BR problem. The decision-problem variant gets as input a TNG $\mathcal{T}$, integral strategies $\pi_1, \ldots, \pi_{k-1}$ for Players $1, \ldots, k-1$, and a value μ, and the goal is to decide whether Player k has a strategy π_k such that $cost_k(\langle \pi_1, \ldots, \pi_k \rangle) \leq \mu$. Theorem 4 implies a reduction from the BR problem to the COR problem and a reduction in the other direction is easy since PTAs can be seen as TNGs with a single player. For one-clock instances, we show that the BR problem is NP-hard by a reduction from the subset-sum problem. Note the contrast with the COR problem in one-clock instances, which is NLOGSPACE-complete [41]. The proof of the following theorem can be found in the full version.

▶ **Theorem 5.** *The BR problem is PSPACE-complete for TNGs with two or more clocks. For one-clock cost-sharing and congestion TNGs it is in PSPACE and NP-hard.*

Proof. We reduce the BR problem to and from the COR problem, which is PSPACE-complete for PTAs with at least two clocks [19]. A PTA can be seen as a one-player TNG, thus the BR problem for TNGs with two or more clocks is PSPACE-hard. For the upper bound, given a TNG $\mathcal{T}$, strategies $Q = \langle \pi_1, \ldots, \pi_{k-1} \rangle$ for Players $1, \ldots, k-1$, and a threshold μ, we construct a PTA $\mathcal{P}$ as in the proof of Theorem 4. Note that the size of $\mathcal{P}$ is polynomial in the size of the input and that $\mathcal{P}$ has one more clock than $\mathcal{T}$. An optimal path in $\mathcal{P}$ is a best response for Player k, and such a path can be found in PSPACE.

The final case to consider is TNGs with one clock. We show that the BR problem is NP-hard for such instances using a reduction from the *subset-sum* problem. The input to that problem is a set of natural numbers $A = \{a_1, \ldots, a_n\}$ and $\mu \in \mathbb{N}$, and the goal is to decide whether there is a subset of A whose sum is μ. We start with the cost-sharing case. The game we construct is a two-player game on a network that is depicted in Figure 2. Player 2 has a unique strategy that visits vertex v_{n+1} in the time interval $[\mu, \mu + 1]$. A Player 1 strategy π corresponds to a choice of a subset of A. Player 1's source is v_1 and her target is u_2. The vertex v_{n+1} is the only vertex that has a cost, which is 1, and the other vertices cost 0. For $1 \leq i \leq n$, Player 1 needs to choose between staying in vertex v_i for a duration of a_i time units, and exiting the vertex through the top edge, or staying 0 time units, and exiting the vertex through the bottom edge. Finally, she must stay in v_{n+1} for exactly one time unit. The cost Player 1 pays for v_{n+1} depends on the load. If she stays there in the global time interval $[\mu, \mu + 1]$, she pays $1/2$, and otherwise she pays 1. Thus, Player 1 has a strategy with which she pays $1/2$ iff there is a subset of A whose sum is μ, and we are done.

The reduction for congestion games is similar. Recall that in congestion games, the cost increases with the load, thus a player would aim at using a vertex together with as few other players as possible. The network is the same as the one used above. Instead of two

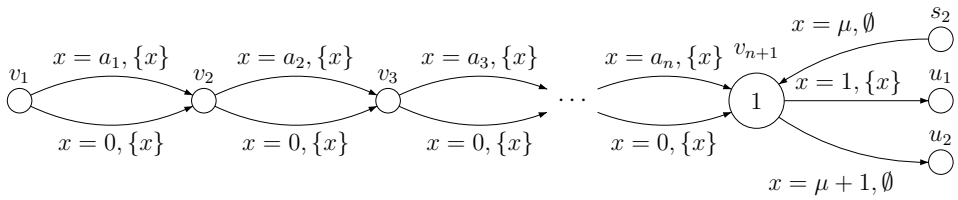


Figure 2 NP-hardness proof of best response problem in one clock TNG.

players, we use three players, where Players 2 and 3 have a unique strategy each. Player 2 must stay in v_{n+1} in the time interval $[0, \mu]$ and Player 3 must stay there during the interval $[\mu + 1, \sum_{1 \leq i \leq n} a_i]$. As in the above, Player 1 has a strategy in which she uses v_{n+1} alone in the time interval $[\mu, \mu + 1]$ iff there is a subset of A whose sum is μ. ◀

3.3 The social-optimum problem

▶ **Theorem 6.** *Consider a TNG $\mathcal{T} = \langle k, C, V, E, \{\ell_v\}_{v \in V}, \langle s_i, u_i \rangle_{i \in [k]} \rangle$. There is a PTA $\mathcal{P}$ with $k \cdot |C|$ clocks, $|V|^k$ vertices, and two vertices $\bar{s}$ and $\bar{u}$ such that there is a one-to-one cost-preserving correspondence between profiles in $\mathcal{T}$ and paths from $\bar{s}$ to $\bar{u}$; namely, for a profile P and its corresponding path η_P, we have $cost(P) = price(\eta_P)$.*

Proof. We describe the intuition of the construction and the details can be found in the full version. Recall that the social optimum is obtained when the players do not act selfishly, rather they cooperate to find the profile that minimizes their sum of costs. Let $\mathcal{T} = \langle k, C, V, E, \{\ell_v\}_{v \in V}, \langle s_i, u_i \rangle_{i \in [k]} \rangle$. We construct a PTA $\mathcal{P}$ by taking k copies of $\mathcal{T}$. For $i \in [k]$, the i-th copy is used to keep track of the timed path that Player i uses. We need k copies of the clocks of $\mathcal{T}$ to guarantee that the individual paths are legal. Recall that the players' goal is to minimize their total cost, thus for each point in time, the price they pay in $\mathcal{P}$ is the sum of their individual costs in $\mathcal{T}$. More formally, consider a vertex $\bar{v} = \langle v_1, \ldots, v_k \rangle$ in $\mathcal{P}$ and let $S_{\bar{v}} \subseteq V$ be the set of vertices that appear in $\bar{v}$. Then, the load on a vertex $v \in S_{\bar{v}}$ in $\bar{v}$ is $load_{\bar{v}}(v) = |\{i : v_i = v\}|$, and the rate of $\bar{v}$ is $\sum_{v \in S_{\bar{v}}} \ell_v(load_{\bar{v}}(v))$. The cost of the social optimum in $\mathcal{T}$ coincides with the price of the optimal timed path in $\mathcal{P}$ from $\langle s_1, \ldots, s_k \rangle$ to the vertex $\langle u_1, \ldots, u_k \rangle$, i.e., the vertices that respectively correspond to the sources and targets of all players. ◀

We turn to study the complexity of the SOPT problem. In the decision-problem variant, we are given a TNG $\mathcal{T}$ and a value μ and the goal is to decide whether there is a profile P in $\mathcal{T}$ with $cost(P) \leq \mu$. Theorem 6 implies a reduction from the SOPT problem to the COR problem, and, as in the BR problem, the other direction is trivial. For one-clock instances, we use the same NP-hardness proof as in the BR problem. The details can be found in the full version.

▶ **Theorem 7.** *The SOPT problem is PSPACE-complete for at least two clocks and it is NP-hard for TNGs with one clock.*

4 Existence of a Nash Equilibrium

The first question that arises in the context of games is the existence of an NE. In [13], we showed that GTNGs are guaranteed to have an NE by reducing every GTNG to an NG. We strengthen the result by showing that every TNG has an NE.

In order to prove existence, we combine techniques from NGs and use the reduction to PTA in Theorem 4. A standard method for finding an NE is showing that a *best-response sequence* converges: Starting from some profile $P = \langle \pi_1, \ldots, \pi_k \rangle$, one searches for a player that can benefit from a unilateral deviation. If no such player exists, then P is an NE and we are done. Otherwise, let π'_i be a beneficial deviation for Player i, i.e., $cost_i(P) > cost_i(P[i \leftarrow \pi'_i])$. The profile $P[i \leftarrow \pi'_i]$ is considered next and the above procedure repeats.

A *potential function* for a game is a function Ψ that maps profiles to costs, such that the following holds: for every profile $P = \langle \pi_1, \ldots, \pi_k \rangle$, $i \in [k]$, and strategy π'_i for Player i, we have $\Psi(P) - \Psi(P[i \leftarrow \pi'_i]) = cost_i(P) - cost_i(P[i \leftarrow \pi'_i])$, i.e., the change in potential equals the change in cost of the deviating player. A game is a *potential game* if it has a potential function. In a potential game with finitely many profiles, since the potential of every profile is non-negative and in every step of a best-response sequence the potential strictly decreases, every best-response sequence terminates in an NE. It is well-known that RAGs are potential games [48] and since they are finite, this implies that an NE always exists.

The idea of our proof is as follows. First, we show that TNGs are potential games, which does not imply existence of NE since TNGs have infinitely many profiles. Then, we focus on a specific best-response sequence that starts from an integral profile and allows the players to deviate only to integral strategies. Finally, we define *normalized TNGs* and show how to *normalize* a TNG in a way that preserves existence of NE. For normalized TNGs, we show that the potential reduces at least by 1 along each step in the best-response sequence, thus it converges to an NE.

▶ **Theorem 8.** *TNGs are potential games.*

Proof. Consider a TNG $\mathcal{T} = \langle k, C, V, E, \{\ell_v\}_{v \in V}, \langle s_i, u_i \rangle_{i \in [k]} \rangle$. Recall that for a profile P, the set of intervals that are used in P is Υ_P. We define a potential function Ψ that is an adaptation of Rosenthal's potential function [48] to TNGs. We decompose the definition of Ψ into smaller components, which will be helpful later on. For every $\gamma \in \Upsilon_P$ and $v \in V$, we define $\Psi_{\gamma,v}(P) = \sum_{j=1}^{load_P(v,\gamma)} |\gamma| \cdot \ell_v(j)$, that is, we take the sum of $|\gamma| \cdot \ell_v(j)$ for all $j \in [load_P(v, \gamma)]$. We define $\Psi_\gamma(P) = \sum_{v \in V} \Psi_{\gamma,v}(P)$, and we define $\Psi(P) = \sum_{\gamma \in \Upsilon_P} \Psi_\gamma(P)$. Let for some $i \in [k]$, we have P' to be a profile that is obtained by an unilateral deviation of Player i to a strictly beneficial strategy π'_i from her current strategy in P, that is $P' = P[i \leftarrow \pi']$ for some $i \in [k]$. In the full version, we show that $\Psi(P) - \Psi(P') = cost_i(P) - cost_i(P')$. ◀

Recall from Theorem 4, that given a TNG, a profile P and an index i, we find the best response of Player i by constructing a PTA. If P is an integral profile, from Theorem 3, we have that the best response of Player i also leads to an integer profile. Thus we have the following lemma.

▶ **Lemma 9.** *Consider a TNG $\mathcal{T}$ and an integral profile P. For $i \in [k]$, if Player i has a beneficial deviation from P, then she has an integral beneficial deviation.*

The last ingredient of the proof gives a lower bound for the difference in cost that is achieved in a beneficial integral deviation for some player $i \in [k]$, which in turn bounds the change in potential.

We first need to introduce a *normalized* form of TNGs. Recall that the latency function in a TNG $\mathcal{T}$ is of the form $\ell_v : [k] \to \mathbb{Q}_{\geq 0}$. In a normalized TNG all the latency functions map loads to natural numbers, thus for every vertex $v \in V$, we have $\ell_v : [k] \to \mathbb{N}$. Constructing a normalized TNG from a TNG is easy. Let L be the least common multiple of the denominators of the elements in the set $\{\ell_v(l) : v \in V \text{ and } l \in [k]\}$. For every latency function ℓ_v and every $l \in [k]$, we construct a new latency function ℓ'_v by $\ell'_v(l) = \ell_v(l) \cdot L$.

Consider a TNG $\mathcal{T}$ and let $\mathcal{T}'$ be the normalized TNG that is constructed from $\mathcal{T}$. It is not hard to see that for every profile P and $i \in [k]$, we have $cost_i(P)$ in $\mathcal{T}'$ is $L \cdot cost_i(P)$ in $\mathcal{T}$. We can thus restrict attention to normalized TNGs as the existence of NE and convergence of best-response sequence in $\mathcal{T}'$ implies the same properties in $\mathcal{T}$. In order to show that a best-response sequence converges in TNGs, we bound the change of potential in each best-response step by observing that in normalized TNGs, the cost a player pays is an integer.

▶ **Lemma 10.** *Let $\mathcal{T}$ be a normalized TNG, $P = \langle \pi_1, \ldots, \pi_k \rangle$ be an integral profile in $\mathcal{T}$, and π_i' be a beneficial integral deviation for Player i, for some $i \in [k]$. Then, $cost_i(P) - cost_i(P[i \leftarrow \pi_i']) \geq 1$.*

We can now prove the main result in this section.

▶ **Theorem 11.** *Every TNG has an integral NE. Moreover, from an integral profile P, there is a best-response sequence that converges to an integral NE.*

Proof. Lemma 9 allows us to restrict attention to integral deviations. Indeed, consider an integral profile P. Lemma 9 implies that if no player has a beneficial integral deviation from P, then P is an NE in $\mathcal{T}$. We start best-response sequence from some integral profile P_I and allow the players to deviate with integral strategies only. Consider a profile P and let P' be a profile that is obtained from P by a deviation of Player i. Recall from Theorem 8 that $cost_i(P) - cost_i(P') = \Psi(P) - \Psi(P')$. Lemma 10 implies that when the deviation is beneficial, we have $\Psi(P) - \Psi(P') \geq 1$. Since the potential is non-negative, the best-response sequence above converges within $\Psi(P_I)$ steps. ◀

▶ Remark. A TNG that allows $<$ and $>$ operators on the guards is not guaranteed to have an NE. Indeed, in a PTA, which can be seen as a one-player TNG, strict guards imply that an optimal timed path may not be achieved. In turn, this means that an NE does not exist. To overcome this issue, we use ϵ-NE, for $\epsilon > 0$; an ϵ-deviation is one that improves the payoff of a player at least by ϵ, and an ϵ-NE is a profile in which no player has a ϵ-deviation. Our techniques can be adapted to show that ϵ-NE exist in TNGs with strict guards. The proof uses the results of [19] that show that an ϵ-optimal timed path exists in PTAs. The proof technique for existence of NE in TNGs with non-strict guards can then be adapted to the strict-guard case.

5 Equilibrium Inefficiency

In this section we address the problem of measuring the degradation in social welfare due to selfish behavior, which is measured by the PoS and PoA measures. We show that the upper bounds from RAGs on these two measures apply to TNGs. For cost-sharing TNGs, we show that the PoS and PoA are at most $\log k$ and k, respectively, as it is in cost-sharing RAGs. Matching lower bounds were given in [13] already for GTNGs. For congestion TNGs with affine latency functions, we show that the PoS and PoA are $1 + \sqrt{(3)}/3 \approx 1.577$ and $\frac{5}{2}$, respectively, as it is in congestion RAGs. Again, a matching lower bound for PoA is shown in [13] for GTNGs, and a matching lower bound for the PoS remains open. Let $\mathcal{F}$ denote a family of latency functions and $\mathcal{F}$-TNGs and $\mathcal{F}$-RAGs denote, respectively, the family of TNGs and RAGs that use latency functions from this family.

▶ **Theorem 12.** *Consider a family of latency functions $\mathcal{F}$. We have $PoS(\mathcal{F}\text{-}TNGs) \leq PoS(\mathcal{F}\text{-}RAGs)$ and $PoA(\mathcal{F}\text{-}TNGs) \leq PoA(\mathcal{F}\text{-}RAGs)$. In particular, the PoS and PoA for cost-sharing TNGs with k players is at most $\log(k)$ and k, respectively, and for congestion TNGs with affine latency functions it is at most roughly 1.577 and $\frac{5}{2}$ respectively.*

Proof. We prove for PoS in cost-sharing games and the other proofs are similar. Consider a TNG $\mathcal{T}$ and let $N^1, N^2, \ldots$ be a sequence of NEs whose cost tends to $c^* = \inf_{P \in \Gamma(\mathcal{T})} cost(P)$. Let O be a social optimum profile in $\mathcal{T}$, which exists due to Theorem 6. Thus, $PoS(\mathcal{T}) = \lim_{j \to \infty} cost(N^j)/\ cost(O)$. We show that each element in the sequence is bounded above by PoS(cost-sharing RAGs), which implies that $PoS(\mathcal{T}) \leq PoS$(cost-sharing RAGs), and hence PoS(cost-sharing TNGs) $\leq PoS$(cost-sharing RAGs). In the full version, for each $j \geq 1$, we construct a RAG $\mathcal{R}_j$ that has $PoS(\mathcal{R}_j) = cost(N^j)/cost(O)$, and since $\mathcal{R}_j$ is a cost-sharing RAG, we have $PoS(\mathcal{R}_j) \leq PoS$(cost- sharing RAGs), and we are done. ◀

6 Time Bounds

Recall that due to resets of clocks, the time by which a profile ends can be potentially unbounded. It is interesting to know, given a TNG, whether there are time bounds within which some interesting profiles like an NE and an SO are guaranteed to exist. Earlier we showed that every TNG is guaranteed to have an integral NE (Theorem 11) and an integral SO (Theorem 6). In this section we give bounds on the time by which such profiles end. That is, given a TNG $\mathcal{T}$, we find $t_{NE}(\mathcal{T}), T_{SO}(\mathcal{T}) \in \mathbb{Q}_{\geq 0}$ such that an integral NE N and an integral SO O exist in $\mathcal{T}$ in which the players reach their destinations by time $t_{NE}(\mathcal{T})$ and $T_{SO}(\mathcal{T})$ respectively.

We start by showing a time bound on an optimal timed path in a PTA, and then proceed to TNGs.

▶ **Lemma 13.** *Consider a PTA $\mathcal{P} = \langle C, V, E, \{r_v\}_{v \in V} \rangle$, and let χ be the largest constant appearing in the guards on the edges of $\mathcal{P}$. Then, for every $s, u \in V$, there is an integral optimal timed path from s to u that ends by time $|V| \cdot (\chi + 2)^{|C|}$.*

Proof. Consider an optimal integral timed path η in $\mathcal{P}$ that ends in the earliest time and includes no loop that is traversed instantaneously. Let $v_0, \ldots, v_n$ be the sequence of vertices that η traverses, and, for $0 \leq i < n$, let κ_i be the clock valuation before exiting the vertex v_i. Since η is integral, κ_i assigns integral values to clocks. Note that since the largest constant appearing in a guard in $\mathcal{P}$ is χ, the guards in $\mathcal{P}$ cannot differentiate between clock values greater than χ. We abstract away such values and define the *restriction* of a clock valuation κ_i to be $\beta_i : C \to (\{0\} \cup [\chi] \cup \{\top\})$ by setting, for $x \in C$, the value $\beta_i(x) = \kappa_i(x)$, when $\kappa_i(x) \leq \chi$, and $\beta_i(x) = \top$, when $\kappa_i(x) > \chi$. Assume towards contradiction that η ends after time $|V| \cdot (\chi + 2)^{|C|}$. Then, there are $0 \leq i < j < n$ such that $\langle v_i, \beta_i \rangle = \langle v_j, \beta_j \rangle$. Let $\eta = \eta_1 \cdot \eta_2 \cdot \eta_3$ be a partition of η such that η_2 is the sub-path between the i-th and j-th indices. Consider the path $\eta' = \eta'_1 \cdot \eta'_3$ that is obtained from η by removing the sub-path η_2. First, note that η' is a legal path. Indeed, the restrictions of the clock valuations in η_1 and η_3 match these in η'_1 and η'_3, that is, $\eta' = \eta_1 \cdot \eta_3$. Second, since we assume that traversing the loop η_2 is not instantaneous, we know that η' ends before η. Moreover, since the rates in $\mathcal{P}$ are non-negative, we have $price(\eta') \leq price(\eta)$, and we reach a contradiction to the fact that η is an optimal timed path that ends earliest. ◀

▶ **Theorem 14.** *For a k-player TNG $\mathcal{T}$ with a set V of vertices and a set C of clocks, there exists an SO that ends by time $\mathcal{O}(|V|^k \cdot \chi^{k|C|})$, where χ is the maximum constant appearing in $\mathcal{T}$. For every $k \geq 1$, there is a k-player (cost-sharing and congestion) TNG $\mathcal{T}_k$ such that $\mathcal{T}_k$ has $\mathcal{O}(k)$ states, the boundaries in the guards in $\mathcal{T}_k$ are bounded by $\mathcal{O}(k \log k)$, and any SO in $\mathcal{T}_k$ requires time $2^{\Omega(k)}$.*

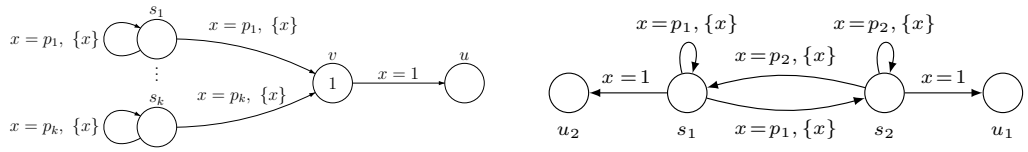


Figure 3 The time required for the SO is not polynomial.

Proof. We start with the upper bound. Consider a TNG $\mathcal{T}$ with a set V of vertices and a set C of clocks. By Theorem 6, we can construct a PTA $\mathcal{P}$ with $|V|^k$ vertices and $k|C|$ clocks such that a social optimum of $\mathcal{T}$ is an optimal timed path in $\mathcal{P}$. Applying Lemma 13, we are done.

We turn to the lower bounds. We show that for every $k \geq 1$, there is a k-player (cost-sharing and congestion) TNG $\mathcal{T}_k$ such that $\mathcal{T}_k$ has $\mathcal{O}(k)$ states, the boundaries in the guards in $\mathcal{T}_k$ are bounded by $\mathcal{O}(k \log k)$, and any SO in $\mathcal{T}_k$ requires time $2^{\Omega(k)}$.

Consider the k-player cost-sharing TNG appearing on the left of Figure 3. Let $p_1, \ldots, p_k$ be relatively prime (e.g., the the first k prime numbers). All the vertices in the TNG have cost 0, except for v, which has some positive cost function. Each player i has to spend one time unit in v in her path from s_i to u. In an SO, all k players spend this one time unit simultaneously, which forces them all to reach v at time $\prod_{1 \leq i \leq k} p_i$. Since the i-th prime number is $O(i \log i)$ and the product of the first i prime numbers is $2^{\Omega(i)}$, we are done. We note that we could define the TNG also with no free vertices, that is vertics with 0 cost, by setting the cost in v to be much higher than those in the source vertices.

For congestion games, the example is more complicated. We start with the case of two players. Consider the congestion TNG appearing on the right of Figure 3. Assume that p_1 and p_2 are relatively prime, $r_{s_1}(1) = r_{s_2}(1) = 0$, and $r_{s_1}(2) = r_{s_2}(2) = 1$. In the SO, the two players avoid each other in their paths from s_i to u_i, and the way to do so is to wait $p_1 \cdot p_2$ time units before the edge from s_i to s_{3-i} is traversed. In the full version,we generalize this example to k players. Again, we could define the TNG with no free vertices. ◀

We proceed to derive a time bound for the existence of an NE. For a TNG $\mathcal{T}$, let $L_{\mathcal{T}} \in \mathbb{N}$ be the smallest number such that multiplying the latency functions by $L_{\mathcal{T}}$ results in a normalized TNG. Recall the $SO(\mathcal{T})$ is the cost of a social optimum in $\mathcal{T}$.

▶ **Theorem 15.** *Consider a TNG $\mathcal{T}$ with k players, played on a timed network $\langle V, E, C \rangle$, and let χ be the maximum constant appearing in a guard. Then, there is an NE in $\mathcal{T}$ that ends by time $\mathcal{O}(\varphi \cdot |V| \cdot \chi^{|C|} + |V|^k \cdot \chi^{k|C|})$, where $\varphi = L_{\mathcal{T}} \cdot SO(\mathcal{T})$ for congestion TNGs and $\varphi = L_{\mathcal{T}} \cdot \log(k) \cdot SO(\mathcal{T})$ for cost-sharing TNGs.*

Proof. Recall the proof of Theorem 11 that shows that every TNG has an integral NE: we choose an initial integral profile P and perform integral best-response moves until an NE is reached. The number of iterations is bounded by the potential $\Psi(P)$ of P. We start the best-response sequence from a social-optimum profile O that ends earliest. By Theorem 14, there is such a profile that ends by time $\mathcal{O}(|V|^k \cdot \chi^{k|C|})$. Let $\varphi = L_{\mathcal{T}} \cdot SO(\mathcal{T})$ in the case of congestion TNGs and $\varphi = L_{\mathcal{T}} \cdot (\ln(k) + 1) \cdot SO(\mathcal{T})$ in the case of cost-sharing TNGs. It is not hard to show that $\Psi(O) \leq \varphi$.

Next, we bound the time that is added in a best-response step. We recall the construction in Theorem 4 of the PTA $\mathcal{P}$ for finding a best-response move. Consider a TNG $\mathcal{T}$ and a profile of strategies P, where, w.l.o.g., we look for a best-response for Player k. Suppose the strategies of Players $1, \ldots, k-1$ take transitions at times $\tau_1, \ldots, \tau_n$. We construct a PTA $\mathcal{P}$

with $n+1$ copies of $\mathcal{T}$. For $1 \leq i \leq n+1$, an optimal path in $\mathcal{P}$ starts in the first copy and moves from copy i to copy $(i+1)$ at time τ_i. We use the additional "global" clock to enforce these transitions. A key observation is that in the last copy, this additional clock is never used. Thus, the largest constant in a guard in the last copy coincides with χ, the largest constant appearing in $\mathcal{T}$. Let η be an optimal path in $\mathcal{P}$ and π_k the corresponding strategy for Player k. We distinguish between two cases. If η does not enter the last copy of $\mathcal{P}$, then it ends before time τ_n, namely the latest time at which a player reaches her destination. Then, the profile $P[k \leftarrow \pi_k]$ ends no later than P. In the second case, the path η ends in the last copy of $\mathcal{P}$. We view the last copy of $\mathcal{P}$ as a PTA. By Lemma 13, the time at which η ends is within $|V| \cdot (\chi+2)^{|C|}$ since its entrance into the copy, which is τ_n. Then, $P[i \leftarrow \pi_k]$ ends at most $|V| \cdot (\chi+2)^{|C|}$ time units after P. To conclude, the best-response sequence terminates in an NE that ends by time $\mathcal{O}(\varphi \cdot |V| \cdot (\chi+2)^{|C|} + |V|^k \cdot \chi^{k|C|})$. ◀

7 Discussion and Future Work

The model of TNGs studied in this paper extends the model of GTNGs introduced in [13] by adding clocks. From a practical point of view, the addition of clocks makes TNGs significantly more expressive than GTNGs and enables them to model the behavior of many systems that cannot be modeled using GTNGs. From a theoretical point of view, the analysis of TNGs poses different and difficult technical challenges. In the case of GTNGs, a main tool for obtaining positive results is a reduction between GTNGs and NGs. Here, in order to obtain positive results we need to combine techniques from NGs and PTAs.

We left several open problems. In Theorem 11, we describe a method for finding an integral NE through a sequence of BR moves. We leave open the complexity of finding an NE in TNGs. For the upper bound, we conjecture that there is a PSPACE algorithm for the problem. For the lower bound, we would need to find an appropriate complexity class of search problems and show hardness for that class. For example, PLS [31], which lies "close" to P, and includes the problem of finding an NE in NGs, consists of search problems in which a local search, e.g., a BR sequence, terminates. Unlike NGs, where a BR can be found in polynomial time, in TNGs, the problem is PSPACE-complete. To the best of our knowledge, complexity classes for search problems that are higher than PLS were not studied. Further we show that the BR and SO problems for one-clock TNGs is in PSPACE and is NP-hard, leaving open the tight complexity.

This work belongs to a line of works that transfer concepts and ideas between the areas of formal verification and algorithmic game theory: logics for specifying multi-agent systems [8, 26], studies of equilibria in games related to synthesis and repair problems [25, 24, 33, 3], and of non-zero-sum games in formal verification [28, 22]. This line of work also includes efficient reasoning about NGs with huge networks [40, 12], an extension of NGs to objectives that are richer than reachability [16], and NGs in which the players select their paths dynamically [15]. For future work, we plan to apply the real-time behavior of TNGs to these last two concepts; namely, TNGs in which the players' objectives are given as a specification that is more general than simple reachability or TNGs in which the players reveal their choice of timed path in steps, bringing TNGs closer to the timed games of [11, 2].

References

1 S. Aland, D. Dumrauf, M. Gairing, B. Monien, and F. Schoppmann. Exact price of anarchy for polynomial congestion games. *SIAM J. Comput.*, 40(5):1211–1233, 2011.

2 L. Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *Proc. 14th Int. Conf. on Concurrency Theory*, pages 142–156, 2003.

3 S. Almagor, G. Avni, and O. Kupferman. Repairing multi-player games. In *Proc. 26th Int. Conf. on Concurrency Theory*, volume 42 of *LIPIcs*, pages 325–339, 2015.

4 R. Alur, M. Bernadsky, and P. Madhusudan. Optimal reachability for weighted timed games. In *Proc. 31st Int. Colloq. on Automata, Languages, and Programming*, pages 122–133, 2004.

5 R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

6 R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–236, 1994.

7 R. Alur and T. Henzinger. Real-time logics: complexity and expressiveness. In *Proc. 5th IEEE Symp. on Logic in Computer Science*, pages 390–401, 1990.

8 R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

9 R. Alur, S. La Torre, and G. J. Pappas. Optimal paths in weighted timed automata. *Theoretical Computer Science*, 318(3):297–322, 2004.

10 E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. *SIAM J. Comput.*, 38(4):1602–1623, 2008.

11 E. Asarin and O. Maler. As soon as possible: Time optimal control for timed automata. In *Proc 2nd International Workshop on Hybrid Systems: Computation and Control*, pages 19–30, London, UK, UK, 1999. Springer-Verlag.

12 G. Avni, S. Guha, and O. Kupferman. An abstraction-refinement methodology for reasoning about network games. In *Proc. 33rd Int. Joint Conf. on Artificial Intelligence*, pages 70–76, 2017.

13 G. Avni, S. Guha, and O. Kupferman. Timed network games. In *42nd Int. Symp. on Mathematical Foundations of Computer Science*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2017.

14 G. Avni, S. Guha, and O. Kupferman. Timed network games with clocks. *CoRR*, abs/1808.04882, 2018. `arXiv:1808.04882`.

15 G. Avni, T.A. Henzinger, and O. Kupferman. Dynamic resource allocation games. In *Proc. 9th International Symposium on Algorithmic Game Theory*, volume 9928 of *Lecture Notes in Computer Science*, pages 153–166, 2016.

16 G. Avni, O. Kupferman, and T. Tamir. Network-formation games with regular objectives. *Information and Computation*, 251:165–178, 2016.

17 G. Behrmann, A. A. Fehnker, T. Hune, K.G. Larsen, P. Pettersson, J. Romijn, and F. W. Vaandrager. Minimum-cost reachability for priced timed automata. In *Proc 4th International Workshop on Hybrid Systems: Computation and Control*, pages 147–161, London, UK, 2001. Springer-Verlag.

18 J. Bengtsson and W.Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, pages 87–124, 2003.

19 P. Bouyer, T. Brihaye, V. Bruyère, and J-F. Raskin. On the optimal reachability problem of weighted timed automata. *Formal Methods in System Design*, 31(2):135–175, oct 2007.

20 P. Bouyer, F. Cassez, E. Fleury, and K.G. Larsen. Optimal strategies in priced timed game automata. In *Proc. 24th Conf. on Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, pages 148–160, 2004.

21 R. Brenguier, F. Cassez, and J-F Raskin. Energy and mean-payoff timed games. In *Proc 17th International Workshop on Hybrid Systems: Computation and Control*, pages 283–292, 2014.

22 T. Brihaye, V. Bruyère, J. De Pril, and H. Gimbert. On subgame perfection in quantitative reachability games. *Logical Methods in Computer Science*, 9(1), 2012.

23 T. Brihaye, G. Geeraerts, S. N. Krishna, L. Manasa, B. Monmege, and A. Trivedi. Adding negative prices to priced timed games. In *Proc. 25th Int. Conf. on Concurrency Theory*, pages 560–575, 2014.

24 K. Chatterjee. Nash equilibrium for upward-closed objectives. In *Proc. 15th Annual Conf. of the European Association for Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*, pages 271–286. Springer, 2006.

25 K. Chatterjee, T. A. Henzinger, and M. Jurdzinski. Games with secure equilibria. *Theoretical Computer Science*, 365(1-2):67–82, 2006.

26 K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. In *Proc. 18th Int. Conf. on Concurrency Theory*, pages 59–73, 2007.

27 K. Chatterjee, T. A. Henzinger, and V. S. Prabhu. Timed parity games: Complexity and robustness. *Logical Methods in Computer Science*, 7(4), 2011.

28 K. Chatterjee, R. Majumdar, and M. Jurdzinski. On Nash equilibria in stochastic games. In *Proc. 13th Annual Conf. of the European Association for Computer Science Logic*, volume 3210 of *Lecture Notes in Computer Science*, pages 26–40. Springer, 2004.

29 G. Christodoulou and E. Koutsoupias. On the price of anarchy and stability of correlated equilibria of linear congestion games. In *ESA*, pages 59–70, 2005.

30 G. Christodoulou and E. Koutsoupias. The price of anarchy of finite congestion games. In *Proc. 37th ACM Symp. on Theory of Computing*, pages 67–73, 2005.

31 A. Fabrikant, C. Papadimitriou, and K. Talwar. The complexity of pure nash equilibria. In *Proc. 36th ACM Symp. on Theory of Computing*, pages 604–612, 2004.

32 J. Fearnley and M. Jurdziński. Reachability in two-clock timed automata is pspace-complete. In *Proc. 40th Int. Colloq. on Automata, Languages, and Programming*, pages 212–223, Berlin, Heidelberg, 2013. Springer-Verlag.

33 D. Fisman, O. Kupferman, and Y. Lustig. Rational synthesis. In *Proc. 16th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2010.

34 S. Guha, M. Jurdzinski, S. N. Krishna, and A. Trivedi. Mean-payoff games on timed automata. In *Proc. 36th Conf. on Foundations of Software Technology and Theoretical Computer Science*, pages 44:1–44:14, 2016.

35 T. Dueholm Hansen, R. Ibsen-Jensen, and P. Bro Miltersen. A faster algorithm for solving one-clock priced timed games. In *Proc. 24th Int. Conf. on Concurrency Theory*, pages 531–545, 2013.

36 M. Hoefer, V. S. Mirrokni, H. Röglin, and S. Teng. Competitive routing over time. *Theor. Comput. Sci.*, 412(39):5420–5432, 2011. doi:10.1016/j.tcs.2011.05.055.

37 M. Jurdzinski and A. Trivedi. Average-time games. In *Proc. 28th Conf. on Foundations of Software Technology and Theoretical Computer Science*, pages 340–351, 2008.

38 E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. *Computer Science Review*, 3(2):65–69, 2009.

39 E. Koutsoupias and K. Papakonstantinopoulou. Contention issues in congestion games. In *Proceedings of the 39th International Colloquium Conference on Automata, Languages, and Programming - Volume Part II*, ICALP'12, pages 623–635. Springer-Verlag, 2012.

40 O. Kupferman and T. Tamir. Hierarchical network formation games. In *Proc. 23rd Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 10205 of *Lecture Notes in Computer Science*, pages 229–246. Springer, 2017.

41 F. Laroussinie, N. Markey, and P. Schnoebelen. Model checking timed automata with one or two clocks. In *Proc. 15th Int. Conf. on Concurrency Theory*, pages 387–401, 2004.

42 B.C. Moszkowski and Z. Manna. Reasoning in interval temporal logic. In *Logics of Programs*, volume 164 of *Lecture Notes in Computer Science*, pages 371–382. Springer, 1983.

43 J.F. Nash. Equilibrium points in n-person games. In *Proceedings of the National Academy of Sciences of the United States of America*, 1950.

44 C. H. Papadimitriou. Algorithms, games, and the internet. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 749–753, 2001.

45 M. Penn, M. Polukarov, and M. Tennenholtz. Random order congestion games. *Mathematics of Operations Research*, 34(3):706–725, 2009.

46 A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

47 K. Ronald and S. Martin. Nash equilibria and the price of anarchy for flows over time. *Theoretical Computer Science*, 49(1):71–97, 2011.

48 R. W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

49 T. Roughgarden and E. Tardos. How bad is selfish routing? *Journal of the ACM*, 49(2):236–259, 2002.

Hardness Results for Consensus-Halving

Aris Filos-Ratsikas[1]
École Polytechnique Fédérale de Lausanne, Switzerland
aris.filosratsikas@epfl.ch

Søren Kristoffer Stiil Frederiksen
Aarhus University, Denmark
sorensf@gmail.com

Paul W. Goldberg
University of Oxford, United Kingdom
paul.goldberg@cs.ox.ac.uk

Jie Zhang[2]
University of Southampton, United Kingdom
jie.zhang@soton.ac.uk

Abstract

The *Consensus-halving* problem is the problem of dividing an object into two portions, such that each of n agents has equal valuation for the two portions. We study the ϵ-approximate version, which allows each agent to have an ϵ discrepancy on the values of the portions. It was recently proven in [13] that the problem of computing an ϵ-approximate Consensus-halving solution (for n agents and n cuts) is PPA-complete when ϵ is *inverse-exponential*. In this paper, we prove that when ϵ is *constant*, the problem is PPAD-hard and the problem remains PPAD-hard when we allow a constant number of additional cuts. Additionally, we prove that deciding whether a solution with $n-1$ cuts exists for the problem is NP-hard.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases PPAD, PPA, consensus halving, generalized-circuit, reduction

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.24

Related Version A full version of the paper is available at `https://arxiv.org/abs/1609.05136`.

1 Introduction

Suppose that two families wish to split a piece of land into two regions such that every member of each family believes that the land is equally divided, or suppose that a conference organizer wants to assign the conference presentations to the morning and the afternoon sessions, so that every participant thinks that the two sessions are equally interesting. Is it possible to achieve these objectives? If yes, how can it be done and how efficiently? What if we aim for "almost equal" instead of "equal"?

These real-life problems can be modeled as the *Consensus-halving problem* [27]. More formally, there are n agents and an object to be divided; each agent may have a different opinion as to which part of the object is more valuable. The problem is to divide the object

[1] The author was supported by the ERC Advanced Grant 321171 (ALGAME).
[2] The author was supported by the ERC Advanced Grant 321171 (ALGAME).

into two portions such that each of the n agents believes that the two portions have equal value, according to her personal opinion. The division may need to cut the object into pieces and then label each piece appropriately to include it in one of the two portions.

The importance of the Consensus-halving problem - or to be precise, of its approximate version, where there is an associated precision parameter ϵ - other than the fact that it models real-life problems like the ones described above, lies in in the following fact: It is the first "natural" problem that is *complete* for the complexity class PPA, where "natural" here means that its does not contain a circuit explicitly in its definition; this was proven quite recently by Filos-Ratsikas and Goldberg [13]. PPA is a class of *total search problems* [19] defined in [22], and is a superclass of the class PPAD, which precisely captures the complexity of computing a Nash equilibrium [11, 9]. Therefore, generally speaking, a PPA-hardness result is stronger than a PPAD-hardness result.

Crucially however, the hardness result in [13] requires the precision parameter to be inverse-exponential in the number of agents and does not even provably preclude the possibility of efficient algorithms, if we allow larger discrepancies in the values for the two portions. In this paper, we prove that this is actually not possible[3], by showing that even when the allowed discrepancy is independent of the number of agents, the problem is PPAD-hard. Understanding the problem for increasing values of the discrepancy parameter is quite important in terms of capturing precisely its complexity and resembles closely the series of results establishing hardness of computing a mixed ϵ-Nash equilibrium, from ϵ being inverse-polynomial in [11, 9] to being constant in [26], as well as several other problems (see [26]). Additionally, one could imagine that solutions where constant discrepancies are acceptable are the ones arising in several real-life scenarios, such as splitting land.

1.1 Our results

We are interested in the computational complexity of computing an ϵ-approximate solution to the Consensus-halving problem where ϵ is a constant function of the number of agents, as well as the complexity of deciding whether given an input instance, $n-1$ cuts are sufficient to achieve an ϵ-approximate solution. We discuss our main results below.

- We prove that the problem of finding an ϵ-approximate solution to the Consensus-halving problem for n agents using n cuts is PPAD-hard. Moreover, the problem remains PPAD-hard even if we allow a constant number of additional cuts. The result is established via a reduction from the approximate Generalized Circuit problem [9, 11, 26].
- We prove that it is NP-hard to decide whether or not an ϵ-approximate solution to the Consensus-halving problem for n agents using $n-1$ cuts exists. Using the gadgetry already developed for the PPAD-hardness proof, we establish the result via a reduction from 3-SAT.
- We prove that the problem of finding an ϵ-approximate solution to the Consensus-halving problem for n agents using n cuts is in the computational class PPA; we obtain the result via a reduction to the computational version of Tucker's Lemma [22, 1].

We remark here that an earlier version of this paper actually predated [13], and some of the results in [13] are established by referencing the results in the present paper. Specifically:

[3] Under usual computational complexity assumptions, here that PPAD-hard problems do not admit polynomial-time algorithms.

- While the authors of [13] provide a rather elaborate reduction to establish PPA-hardness of the problem, the inclusion in the class PPA is established with reference to the present paper. In turn, the inclusion result follows from a formalization of the ideas of the algorithms by [27] and [24] and Fan's version of Tucker's Lemma [12, 31].
- In [13], the authors obtain a *computational equivalence* between the *Necklace Splitting* problem [2] and ϵ-approximate Consensus-halving, for ϵ being at least inverse-polynomial. The inverse-polynomial dependence on ϵ implies that PPA-hardness of the former problem does not follow from their hardness result, but PPAD-hardness does follow from their reduction and our main result here.

Due to lack of space, some of the proofs and details are left for the the full version of the paper. Most emphasis is put on the main PPAD-hardness proof of Section 3, which is presented in sufficient detail, and the NP-hardness proof of Section 4. The exposition of the results in Section 5 is limited to higher-level intuition, with full proofs in the full version.

1.2 Related work

The Consensus-halving problem was explicitly formalized and studied firstly by Simmons and Su [27], who proved that a solution with n cuts always exists and constructed a protocol that finds an approximate solution, which allows for a small discrepancy on the values of the two portions. Their proofs are based on one of the most applied theorems in topology, the Borsuk-Ulam Theorem [6] and its combinatorial analogue, known as Tucker's Lemma [31]. The existence of solutions to the problem was already known since [16, 3, 4] but the algorithm in [27] is constructive, in the sense that it actually finds such a solution and furthermore, it does not require the valuations of the players to be additively separable over subintervals, like some of the previous papers do. Actually, for the case of valuations which are probability measures, the existence of a solution with n cuts was known since as early as the 1940s [20] and can also be obtained as an application of the Hobby-Rice Theorem [18] (also see [2]). Despite proposing an explicit protocol however, the authors in [27] do not answer the question of "efficiency", i.e. how fast can a protocol find an (approximate) solution and the running time of their protocol is worst-case exponential-time.[4]

To this end, Filos-Ratsikas and Goldberg [13] recently proved that the problem is PPA-complete, but as we explained in the introduction, the hardness holds only when the precision parameter is inversely exponential. Even more recently, the authors strengthened their result to PPA-completeness of the problem for inversely polynomial precision [14]. However, since our hardness result holds for constant precision, it is not subsumed by neither [13] or [14].

The computational classes PPA (Polynomial Parity Arguments) and PPAD (Polynomial Parity Arguments on Directed graphs) were introduced by Papadimitriou [22] in an attempt to capture the precise complexity of several interesting problems of a topological nature such as computational analogues of Sperner's Lemma [28] and Brouwer's and Kakutani's fixed point theorems [5], which are all known to be in PPAD [22]. Interestingly, Aisenberg et al. [1] recently proved that the search problems associated with the Borsuk-Ulam Theorem and Tucker's Lemma are PPA-complete; this is the starting point for the reduction in [13], but it will also be used for our "in-PPA" result, which complements the hardness result of [13].

Our PPAD-hardness reduction goes via the *Generalized Circuit* problem. Generalized circuits differ from usual circuits in the sense that they can contain cycles, a fact which basically induces a continuous function on the gates, and the solution is guaranteed to exist

[4] The protocol exhaustively iterates through all the vertices of triangulated geometric object, which, to achieve a small discrepancy, has to be exponentially large.

by Brouwer's fixed point theorem. The ϵ-approximate Generalized Circuit problem was implicitly proven to be PPAD-complete for exponentially small ϵ in [11] and explicitly for polynomial small ϵ [9], en route to proving that perhaps the most interesting problem in PPAD, that of computing a mixed-Nash equilibrium, is also complete for the class. The same problem was also used in [10] to prove that finding an approximate competitive equilibrium for the Arrow-Debreu market with linear and non-monotone utilities is PPAD-complete and in [21] to prove that finding an approximate solution of the Competitive Equilibrium with Equal Incomes (CEEI) for indivisible items is PPAD-complete. More recently, Rubinstein [26] showed that computing an ϵ-approximate solution for the Generalized Circuit problem is PPAD-complete for a constant ϵ, which implies that finding an ϵ-approximate Nash equilibrium is PPAD-complete for constant ϵ, in the context of graphical games; we reduce from that version of the problem. This improvement should also lead to stronger hardness results in [10] and [21], as well as other problems that rely on the Generalized Circuit problem.

The Consensus-halving problem is a typical fair division problem that studies how to divide a set of resources between a set of agents who have valuations on the resources, such that some fairness properties are fulfilled. The fair division literature, which dates back to the late 1940s [29], has studied a plethora of such problems, with *chore-division* [23, 15], *rent-partitioning* [17, 7, 30] and the perhaps the most well-known one, *cake-cutting* [8, 25] being notable examples. Note that Consensus-halving is inherently different from cake-cutting, since the objective is that all participants are (approximately) equally satisfied with the solution, and they do not have "ownership" over the resulting pieces.

2 Preliminaries

We represent the object O as a line segment $[0, 1]$. Each agent in the set of agents $N = \{1, \ldots, n\}$ has its own valuation over any subset of interval $[0, 1]$. These valuations are:

- *non-negative and bounded*, i.e. there exists $M > 0$, such that for any subinterval $X \subseteq [0, 1]$, it holds that $0 \leq u_i(X) \leq M$.
- *non-atomic*, i.e. agents have no value on any single point on the interval.

For simplicity, the reader may assume that the valuations are represented as step functions (where agents have constant values over distinct intervals), although this is not necessary for the results to hold.[5]

A set of k cuts $\{t_1, \ldots, t_k\}$, where $0 \leq t_1 \leq \ldots \leq t_k \leq 1$, means that we cut along the points $t_1, \ldots, t_k$, such that the object is cut into $k+1$ pieces $X_i = [t_{i-1}, t_i]$ for $i = 1, \ldots, k+1$, where $t_0 = 0$ and $t_{k+1} = 1$. A labelling of an interval X_i means that we assign a label $\ell \in \{+, -\}$ to X_i, which corresponds to including X_i in a set of intervals, either O_+ or O_-. In case some cuts happen to be on the same point, say $t_{j-1} = t_j$, then the corresponding subinterval X_j is a single point on which agents have no value. We will consider cuts on the same points to be the same cut, e.g. if there is only one such occurrence, we will consider the number of cuts to be $k - 1$.

The Consensus-halving problem is to divide the object O into two portions O_+ and O_-, such that every agent derives equal valuation from the two portions, i.e., $u_i(O_+) =$

[5] The inclusion result actually holds for more general functions, while our hardness results (PPAD-hardness and NP-hardness) hold even for well-behaved functions, such as step functions. We note here that while an exact solution to Consensus-halving generally requires the valuations to be continuous, this is not necessary for the existence of an approximate solution; the algorithm of [27] can find such a solution assuming that valuations are bounded and non-atomic.

$u_i(O_-), \forall i \in N$. The ϵ-approximate Consensus-halving problem allows that each agent has a small discrepancy on the values of the two partitions, i.e., $|u_i(O_+) - u_i(O_-)| < \epsilon$. As discussed in the Introduction, such a solution always exists [27].

We define the following search problem, called (n, k, ϵ)-ConHalving.

▶ **Problem 1.** (n, k, ϵ)-ConHalving.
Input: *The value density functions (valuation functions)* $v_i : O \to R_+, i = 1, \cdots, n$.
Output: *A partition* (O_+, O_-) *with* k *cuts such that* $|u_i(O_+) - u_i(O_-)| \leq \epsilon$.

We will also consider the following decision problem, called $(n, n-1, \epsilon)$-ConHalving. Note that for n agents and $n-1$ cuts, a solution to the ϵ-approximate Consensus-halving problem is not guaranteed to exist.

▶ **Problem 2.** $(n, n-1, \epsilon)$-ConHalving.
Input: *The valensity functions* $v_i : O \to R_+, i = 1, \cdots, n$.
Output: Yes*, if a partition* (O_+, O_-) *with* $n-1$ *cuts such that* $|u_i(O_+) - u_i(O_-)| \leq \epsilon$ *for all agents* $i \in N$ *exists, and* No *otherwise.*

TFNP, PPA and PPAD: Most of the problems that we will consider in this paper belong to the class of *total search problems*, i.e. search problems for which a solution is guaranteed to exist, regardless of the input. In particular, we will be interested in problems in the class TFNP, i.e. total search problems for which a solution is verifiable in polynomial time [19].

An important subclass of TFNP is the class PPAD, defined by Papadimitriou in [22]. PPAD stands for "Polynomial Parity Argument on a Directed graph" and is defined formally in terms of the problem End-Of-Line [22]. The class PPAD is defined in terms of an exponentially large digraph $G = (V, E)$ consisting of 2^n vertices with indegree and outdegree at most 1. An edge between vertices v_1 and v_2 is present in E if and only if the successor $S(v_1)$ of node v_1 is v_2 and the predecessor $P(v_2)$ of node v_2 is v_1. By construction, the point 0^n has indegree 0 and we are looking for a point with outdegree 0, which is guaranteed to exist. Note that the graph is given *implicitly* to the input, through a function that given any vertex v, returns its set of neighbours (predecessor and successor) in polynomial time. PPAD is a subclass of the class PPA ("Polynomial Parity Argument") which is defined similarly, but in terms of an undirected graph in which every vertex has degree at most 2, and given a vertex of degree 1, we are asked to find another vertex of degree 1; the computational problem associated with the class is called Leaf [22] and a problem is the class PPA if it is polynomial-time reducible to Leaf.

The formal definitions of End-Of-Line and Leaf are not required for the results presented in this version and therefore are left for the full version.

2.1 Generalized Circuits

A *generalized circuit* $S = (V, \mathcal{T})$ consists of a set of nodes V and a set of gates $\mathcal{T}$ and let $N = |V|$ and $M = |\mathcal{T}|$. Every gate $T \in \mathcal{T}$ is a 5-tuple $T = (G, v_{in_1}, v_{in_2}, v_{out}, \alpha)$ where

- $G \in \{G_\zeta, G_{\times\zeta}, G_=, G_+, G_-, G_<, G_\vee, G_\wedge, G_\neg\}$ is the type of the gate,
- $v_{in_1}, v_{in_2} \in V \cup \{nil\}$ are the first and second input nodes of the gate or *nil* if not applicable,
- $v_{out} \in V$ is the output node, and $\alpha \in [0, 1] \cup \{nil\}$ is a parameter if applicable,
- for any two gates $T = (G, v_{in_1}, v_{in_2}, v_{out}, \alpha)$ and $T' = (G', v'_{in_1}, v'_{in_2}, v'_{out}, \alpha')$ in $\mathcal{T}$ where $T \neq T'$, they must satisfy $v_{out} \neq v'_{out}$.

Table 1 Gate constraint $T = (G, v_{in_1}, v_{in_2}, v_{out}, \alpha)$.

Gate	Constraint
$(G_\zeta, nil, nil, v_{out}, \alpha)$	$\alpha - \epsilon \leq \mathbf{x}[v_{out}] \leq \alpha + \epsilon$
$(G_{\times\zeta}, v_{in_1}, nil, v_{out}, \alpha)$	$\alpha \cdot \mathbf{x}[v_{in_1}] - \epsilon \leq \mathbf{x}[v_{out}] \leq \alpha \cdot \mathbf{x}[v_{in_1}] + \epsilon$
$(G_=, v_{in_1}, nil, v_{out}, nil)$	$\mathbf{x}[v_{in_1}] - \epsilon \leq \mathbf{x}[v_{out}] \leq \mathbf{x}[v_{in_1}] + \epsilon$
$(G_+, v_{in_1}, v_{in_2}, v_{out}, nil)$	$\mathbf{x}[v_{out}] \in [\min(\mathbf{x}[v_{in_1}] + \mathbf{x}[v_{in_2}], 1) - \epsilon, \min(\mathbf{x}[v_{in_1}] + \mathbf{x}[v_{in_2}], 1) + \epsilon]$
$(G_-, v_{in_1}, v_{in_2}, v_{out}, nil)$	$\mathbf{x}[v_{out}] \in [\max(\mathbf{x}[v_{in_1}] - \mathbf{x}[v_{in_2}], 0) - \epsilon, \max(\mathbf{x}[v_{in_1}] - \mathbf{x}[v_{in_2}], 0) + \epsilon]$
$(G_<, v_{in_1}, v_{in_2}, v_{out}, nil)$	$\mathbf{x}[v_{out}] = \begin{cases} 1 \pm \epsilon, & \text{if } \mathbf{x}[v_{in_1}] < \mathbf{x}[v_{in_2}] - \epsilon; \\ 0 \pm \epsilon, & \text{if } \mathbf{x}[v_{in_1}] > \mathbf{x}[v_{in_2}] + \epsilon. \end{cases}$
$(G_\vee, v_{in_1}, v_{in_2}, v_{out}, nil)$	$\mathbf{x}[v_{out}] = \begin{cases} 1 \pm \epsilon, & \text{if } \mathbf{x}[v_{in_1}] = 1 \pm \epsilon \text{ or } \mathbf{x}[v_{in_2}] = 1 \pm \epsilon; \\ 0 \pm \epsilon, & \text{if } \mathbf{x}[v_{in_1}] = 0 \pm \epsilon \text{ and } \mathbf{x}[v_{in_2}] = 0 \pm \epsilon. \end{cases}$
$(G_\wedge, v_{in_1}, v_{in_2}, v_{out}, nil)$	$\mathbf{x}[v_{out}] = \begin{cases} 1 \pm \epsilon, & \text{if } \mathbf{x}[v_{in_1}] = 1 \pm \epsilon \text{ and } \mathbf{x}[v_{in_2}] = 1 \pm \epsilon; \\ 0 \pm \epsilon, & \text{if } \mathbf{x}[v_{in_1}] = 0 \pm \epsilon \text{ or } \mathbf{x}[v_{in_2}] = 0 \pm \epsilon. \end{cases}$
$(G_\neg, v_{in_1}, nil, v_{out}, nil)$	$\mathbf{x}[v_{out}] = \begin{cases} 1 \pm \epsilon, & \text{if } \mathbf{x}[v_{in_1}] = 0 \pm \epsilon; \\ 0 \pm \epsilon, & \text{if } \mathbf{x}[v_{in_1}] = 1 \pm \epsilon. \end{cases}$

Note that generalized circuits extend the standard boolean or arithmetic circuits in the sense that generalized circuits allow cycles in the directed graph defined by the nodes and gates. We define the search problem ϵ-Gcircuit [9, 26].

▶ **Problem 3.** ϵ-Gcircuit

Input: *A generalized circuit* $S = (V, \mathcal{T})$.

Output: *A vector* $\mathbf{x} \in [0,1]^N$ *of values for the nodes, satisfying the conditions from Table 1.*

Note that a solution to ϵ-Gcircuit always exists [9] and hence the problem is well-defined. Also, notice that for the logic gates $G_\vee, G_\wedge$ and $G_\neg$, if the input conditions are not fulfilled, the output is unconstrained, and for the multiplication gate, it holds that $\alpha \in (0, 1]$. ϵ-Gcircuit was proven to be PPAD-complete implicitly or explicitly in [11, 9] for inversely polynomial error ϵ and in [26] for constant ϵ. We state the latter theorem here as a lemma:

▶ **Lemma 1** ([26]). *There exists a constant* $\epsilon > 0$ *such that* ϵ-Gcircuit *is PPAD-complete.*

3 Consensus-Halving with $n + k$ cuts is is PPAD-hard

In this section, we will first prove that finding an approximate partition for Consensus-halving using n cuts is PPAD-hard, even if the allowed discrepancy between the two portions is a constant. We describe the reduction from ϵ-Gcircuit that we will be using for the PPAD-hardness proof. Given an instance $S = (V, \mathcal{T})$ of ϵ-Gcircuit, we will construct an instance of (n, n, ϵ')-ConHalving with $n = 2N$ agents, in which each node $v_i \in V$ of the circuit will correspond to two agents var_i and $copy_i$ and where ϵ' will be defined later. As a matter of convenience in the reduction, we will assume that for every gate $(G, v_{in_1}, v_{in_2}, v_{out}, \alpha)$ in $\mathcal{T}$, v_{in_1}, v_{in_2} and v_{out} are distinct. This does not affect the hardness of the problem as any ϵ-generalized circuit can be converted to this form by use of at most $2N$ additional equality-gates and nodes, and also since an $(\epsilon/2)$-approximate solution to the converted problem can clearly be converted to a solution in the original circuit.

For ease of notation, we consider a Consensus-halving instance on the interval $[0, 6N]$. Let $d_i := 6(i-1)$.

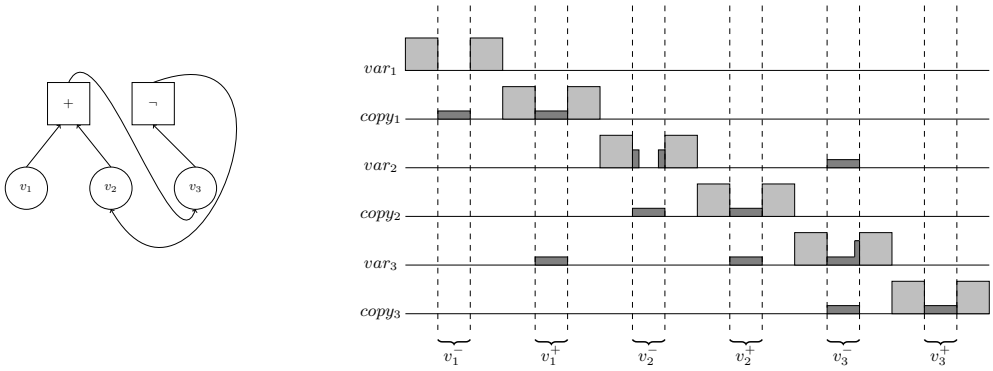


Figure 1 An instance of ϵ-GCIRCUITwith the corresponding construction for (n, ϵ')-CONHALVING.

The two agents var_i and $copy_i$ that we construct for every node v_i have valuations

$$var_i = \begin{cases} border_i(t) + G^\tau(t), & \text{if } v_i \text{ is the output of } \tau \\ border_i(t), & \text{otherwise} \end{cases}$$

$$copy_i = \begin{cases} 4, & t \in [d_i + 3, d_i + 4] \cup [d_i + 5, d_i + 6] \\ 1, & t \in [d_i + 1, d_i + 2] \cup [d_i + 4, d_i + 5] \\ 0, & \text{otherwise} \end{cases}$$

$$\text{where} \quad border_i = \begin{cases} 4, & \text{if } t \in [d_i, d_i + 1] \cup [d_i + 2, d_i + 3] \\ 0, & \text{otherwise} \end{cases}$$

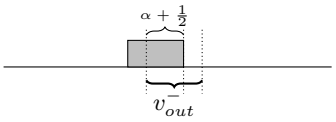
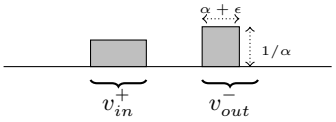
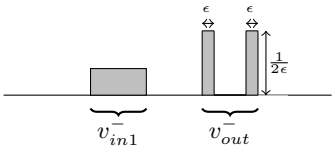
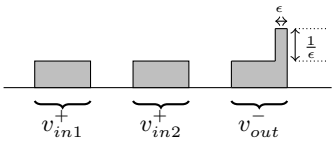
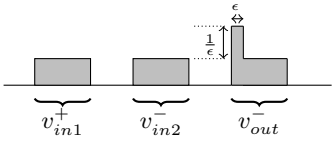
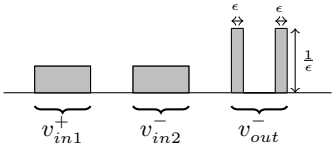
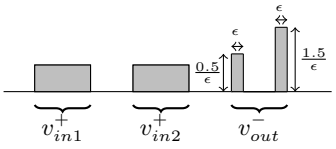
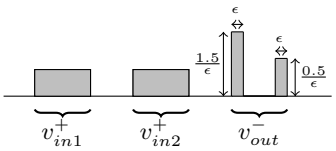
Since each node is the output of at most one gate, var_i is well-defined. Note that apart from the valuation defined by the function G^τ, agents var_i and $copy_i$ only have valuations on the sub-interval $[d_i, d_{i+1}]$, i.e., the agents associated with node v_1 only have valuations on $[0, 6]$, the agents associated with v_2 only on have valuations on $[6, 12]$ and so on. Let $v_i^- := [d_i + 1, d_i + 2]$ and the right and left endpoints respectively be $v_{i,\ell}^-$ and $v_{i,r}^-$, (and analogously for $v_i^+ := [d_i + 3, d_i + 4]$, $v_{i,\ell}^+$ and $v_{i,r}^+$). Now, we are ready to define the functions G^τ according to Table 2. Notice that because of the assumption that the two input nodes and the output node are distinct, the graphs of the functions are as in Table 2. Figure 1 demonstrates an example of a Consensus-halving instance corresponding to a small circuit.

▶ **Lemma 2.** *Given the construction of a* (n, n, ϵ')-CONHALVING *instance above, for* $\epsilon' < \min\{\epsilon/11, 1/40\}$*, a partition with* n *cuts corresponds to a solution to* ϵ-GCIRCUIT*.*

Proof. First observe that since all of the agents var_i and $copy_i$ are constructed to have at least $3/4$ of their valuation on $[d_i, d_i + 3]$ and $[d_i + 3, d_i + 6]$ respectively, there must be at least one cut in each one of those intervals in any ϵ'-approximate solution to Consensus-halving (with $\epsilon' < 1/4$) and therefore any ϵ'-approximate solution to Consensus-halving with $2N$ cuts must have exactly one cut in each interval. Furthermore, since the constructed instance consists of $2N$ agents, by [27], such a partition with $2N$ cuts is guaranteed to exist.

Now consider such a solution $\mathcal{C}$ to (n, n, ϵ')-CONHALVING with $2N$ cuts. For each agent var_i (and associated gate G^τ, if any), since her valuation in v_i^- is at least the same as her valuation outside the interval $[d_i, d_i + 3]$, the cut from $\mathcal{C}$ in $[d_i, d_i + 3]$ must be in $[d_i + 1 - \epsilon', d_i + 2 + \epsilon']$, since $\mathcal{C}$ is a solution to (n, n, ϵ')-CONHALVING. We will assume without loss of generality that the leftmost piece of the partition $\mathcal{C}$ is in O_-. Notice then

Table 2 Agent preferences from gate $\tau = (G, v_{in_1}, v_{in_2}, v_{out}, \alpha)$. For the gate $G_{\times\zeta}$, the figure depicts the case when $\alpha + \epsilon < 1$.

	$G^\tau(t)$	Picture
G_ζ	$\begin{cases} 1 & \text{if } t \in [v^-_{out,\ell} + \alpha - \frac{1}{2}, v^-_{out,\ell} + \alpha + \frac{1}{2}] \\ 0 & \text{otherwise} \end{cases}$	
$G_{\times\zeta}$	$\begin{cases} 1 & \text{if } t \in v^+_{in} \\ 1/\alpha & \text{if } t \in [v^-_{out,\ell}, v^-_{out,\ell} + \min(\alpha + \epsilon, 1)] \\ 0 & \text{otherwise} \end{cases}$	
$G_\neg$	$\begin{cases} 1 & \text{if } t \in v^-_{in} \\ 1/2\epsilon & \text{if } t \in [v^-_{out,\ell}, v^-_{out,\ell} + \epsilon] \\ 1/2\epsilon & \text{if } t \in [v^-_{out,r} - \epsilon, v^-_{out,r}] \\ 0 & \text{otherwise} \end{cases}$	
G_+	$\begin{cases} 1 & \text{if } t \in v^+_{in_1} \cup v^+_{in_2} \\ 1 & \text{if } t \in [v^-_{out,\ell}, v^-_{out,r} - \epsilon] \\ 1/\epsilon + 1 & \text{if } t \in [v^-_{out,r} - \epsilon, v^-_{out,r}] \\ 0 & \text{otherwise} \end{cases}$	
G_-	$\begin{cases} 1 & \text{if } t \in v^+_{in_1} \cup v^-_{in_2} \\ 1 & \text{if } t \in [v^-_{out,\ell} + \epsilon, v^-_{out,r}] \\ 1/\epsilon + 1 & \text{if } t \in [v^-_{out,\ell}, v^-_{out,\ell} + \epsilon] \\ 0 & \text{otherwise} \end{cases}$	
$G_<$	$\begin{cases} 1 & \text{if } t \in v^+_{in_1} \cup v^-_{in_2} \\ 1/\epsilon & \text{if } t \in [v^-_{out,\ell}, v^-_{out,\ell} + \epsilon] \\ 1/\epsilon & \text{if } t \in [v^-_{out,r} - \epsilon, v^-_{out,r}] \\ 0 & \text{otherwise} \end{cases}$	
$G_\vee$	$\begin{cases} 1 & \text{if } t \in v^+_{in_1} \cup v^+_{in_2} \\ 0.5/\epsilon & \text{if } t \in [v^-_{out,\ell}, v^-_{out,\ell} + \epsilon] \\ 1.5/\epsilon & \text{if } t \in [v^-_{out,r} - \epsilon, v^-_{out,r}] \\ 0 & \text{otherwise} \end{cases}$	
$G_\wedge$	$\begin{cases} 1 & \text{if } t \in v^+_{in_1} \cup v^+_{in_2} \\ 1.5/\epsilon & \text{if } t \in [v^-_{out,\ell}, v^-_{out,\ell} + \epsilon] \\ 0.5/\epsilon & \text{if } t \in [v^-_{out,r} - \epsilon, v^-_{out,r}] \\ 0 & \text{otherwise} \end{cases}$	

that for each node v_i, the piece on the left-hand side of the cut in v_i^- is always in O_- and the piece on the left-hand side of the cut in v_i^+ is always in O_+. Let the location of the cut be $d_i + 1 + t_i^-$ where $t_i^- \in [-\epsilon', 1 + \epsilon']$. Analogously, the same argument holds for agent $copy_i$ and the interval $[d_i + 3 - \epsilon', d_i + 4 + \epsilon']$, and define $t_i^+ \in [-\epsilon', 1 + \epsilon']$ similarly.

Now consider the agent $copy_i$ and the cut at location $d_i + 1 + t_i^-$. If $t_i^- \in [0, 1]$, then since agent $copy_i$ has valuation 1 on interval v_i^-, t_i^- of her valuation will be on a piece in O_- and $1 - t_i^-$ of her valuation will be on a piece in O_+. Then, since $\mathcal{C}$ is a solution to

(n, n, ϵ')-ConHalving, the cut in $d_i+3+t_i^+$ must be placed so that $|t_i^- - t_i^+| \leq \epsilon'/2$; similarly for the cases where $t_i^- \notin [0,1]$. In other words, $copy_i$ ensures that the cut at $d_i + 1 + t_i^-$ is "copied" ϵ'-approximately.

We will interpret the solution $\mathcal{C}$ as a solution to ϵ-Gcircuit in the following way. For each node v_i and each associated cut at $d_i + 1 + t_i^-$ let

$$x_i := \begin{cases} 0 & , \quad t_i^- < 0 \\ t_i^- & , \quad t_i^- \in [0,1] \\ 1 & , \quad t_i^- > 1 \end{cases} \tag{1}$$

and notice

$$|t_i^+ - x_i| \leq 2\epsilon' \quad , \quad |t_i^- - x_i| \leq 2\epsilon' \tag{2}$$

To complete the proof, we just need to argue that these variables satisfy the constraints of the gates of the circuit. Due to lack of space, we will only argue the correctness of some of the gates here; the arguments for the remaining gates follow a similar spirit and are presented in detail in the full version.

Constant gate $\tau = (G_\zeta, nil, nil, v_{out}, \alpha)$. The valuation of agent var_{out} for the intervals $[d_i, d_i + 1 + \alpha]$ and $[d_i + 1 + \alpha, d_i + 3]$ is the same and since the height of the agent's value density function is at least 1 in $[d_i, d_i + 3]$,[6] it holds that $t_{out}^- \in [\alpha - \epsilon', \alpha + \epsilon']$. Then, by Equation 2, it holds that $x_{out} \in [\alpha - 3\epsilon', \alpha + 3\epsilon']$, so for $\epsilon' < \epsilon/3$ the gate constraint is satisfied.

Multiplication-by-scalar gate $\tau = (G_{\times\zeta}, v_{in}, nil, v_{out}, \alpha)$. Notice that for any given cut t_{in}^+ and $1 - \alpha \geq \epsilon$, it holds that $t_{out}^- \in [\alpha t_{in}^+ + \epsilon/2 - \epsilon', \alpha t_{in}^+ + \epsilon/2 + \epsilon']$ as the height of G^τ in v_{out}^- is at least 1. Similarly, for the case $1 - \alpha < \epsilon$ and any given cut t_{in}^+, it holds that $t_{out}^- \in [\alpha t_{in}^+ + (1-\alpha)/2 - \epsilon', \alpha t_{in}^+ + (1-\alpha)/2 + \epsilon']$ as the height of G^τ in v_{out}^- is at least 1. In particular, since $1 - \alpha < \epsilon$, it also holds that $t_{out}^- \in [\alpha t_{in}^+ + \epsilon/2 - \epsilon', \alpha t_{in}^+ + \epsilon/2 + \epsilon']$ for this case as well. Then, by Equation 2, it holds that $x_{out} \in [\alpha t_{in}^+ + \epsilon/2 - 3\epsilon', \alpha t_{in}^+ + \epsilon/2 + 3\epsilon']$ and since $\alpha \leq 1$ it also holds that $x_{out} \in [\alpha x_{in} + \epsilon/2 - 5\epsilon', \alpha x_{in} + \epsilon/2 + 5\epsilon']$, again by Equation 2. Then the gate constraint is satisfied whenever $\epsilon' < \epsilon/10$.

Addition gate$\tau = (G_+, v_{in_1}, v_{in_2}, v_{out}, nil)$. If for the cuts $t_{in_1}^+$ and $t_{in_2}^+$ it holds that $t_{in_1}^+ + t_{in_2}^+ < 1 - \epsilon + 4\epsilon'$ then $t_{out}^- \in [t_{in_1}^+ + t_{in_2}^+ - 5\epsilon', t_{in_1}^+ + t_{in_2}^+ + 5\epsilon']$ as the height of G^τ in v_{out}^- is at least 1. This then implies that $x_{out} \in [x_{in_1}^+ + x_{in_2}^+ - 11\epsilon', x_{in_1}^+ + x_{in_2}^+ + 11\epsilon']$, by Inequality 2. On the other hand, when $t_{in_1}^+ + t_{in_2}^+ \geq 1 - \epsilon + 4\epsilon'$, then by Definition 1, it holds that $x_{in_1} + x_{in_2} \in [1 - \epsilon, 1]$ and clearly $t_{out}^- \in [1 - \epsilon, 1 + \epsilon']$ which by Definition 1 implies that $x_{out} \in [1 - \epsilon, 1]$. The gate constraints are satisfied for $\epsilon' < \epsilon/11$ for each of the cases.

Less-than-equal gate $\tau = (G_<, v_{in_1}, v_{in_2}, v_{out}, nil)$. We will consider three cases, depending on the positions of the cuts $t_{in_1}^+$ and $t_{in_2}^-$. First, when $|t_{in_1}^+ - t_{in_2}^-| < \epsilon - 4\epsilon'$, by Inequality 2 it holds that $|x_{in_1} - x_{in_2}| < \epsilon$ and the output of the gate is unconstrained. When $t_{in_1}^+ - t_{in_2}^- \geq \epsilon - 4\epsilon'$ then by Inequality 2 it holds that $x_{in_1} \geq x_{in_2} + \epsilon$. Additionally, since the height of G^τ in $[v_{out,r}^- - \epsilon, v_{out,r}^-]$ is at least 1, it holds that $t_{out}^- \in [1 - \epsilon, 1 + \epsilon']$, which by Definition 1 implies that $x_{out}^- \in [1 - \epsilon, 1]$ and the gate constraint is satisfied. The argument for the case when $t_{in_2}^- > t_{in_1}^+ - 2\epsilon'$ is completely symmetrical.

Logic OR gate $\tau = (G_\vee, v_{in_1}, v_{in_2}, v_{out}, nil)$. We will consider three cases depending on the position of the cuts $t_{in_1}^+$ and $t_{in_2}^+$. First, when $t_{in_1}^+ + t_{in_2}^+ < 0.4$ it holds that

[6] Notice that the constant gate is the only gate where $border_i$ and G^τ overlap.

$t^-_{out} \in [-\epsilon', \epsilon]$ and hence by Definition 1, it holds that $x_{out} \in [0, \epsilon]$. Furthermore, by Inequality 2 it holds that $x_{in_1} + x_{in_2} < 0.4 + 4\epsilon'$ and for $\epsilon' < 1/40$, it also holds that $x_{in_1}, x_{in_2} < 0.5$ and the gate constraint is satisfied. Next, when $t^+_{in_1} + t^+_{in_2} \in [0.4, 0.8]$ then by Inequality 2, it holds that $x_{in_1}, x_{in_2} \in [0.4 - \epsilon', 0.8 + 4\epsilon']$ and in particular, when $\epsilon' < 1/40$ then it also holds that $x_{in_1} + x_{in_2} \in [0.3, 0.9]$ and the output of the gate in unconstrained. Finally when $t^+_{in_1} + t^+_{in_2} > 0.8$, it holds that $t^-_{out} \in [1 - \epsilon, 1 + \epsilon']$ and hence by Definition 1, we have that $x_{out} \in [1 - \epsilon, 1]$. Furthermore, by Inequality 2 we have that $x_{in_1} + x_{in_2} > 0.8 + 4\epsilon'$ which is greater than 0.9 when $\epsilon' < 1/40$ which implies that at least one of the two inputs is greater than ϵ. In particular, the gate's output lies in $[1 - \epsilon, \epsilon]$ when the inputs are smaller than ϵ or greater than $1 - \epsilon$ and at least one of them is greater than $1 - \epsilon$. This shows that the gate constraint is satisfied.

Given the discussion above, by setting $\epsilon' < \min\{\epsilon/11, 1/40\}$[7], the gate constraints are satisfied, and the vector (x_i) obtained from $\mathcal{C}$ is a solution to ϵ-GCIRCUIT. ◀

Now from Lemma 2, we obtain the following result.

▶ **Theorem 3.** *There exists a constant $\epsilon' > 0$ such that (n, n, ϵ')-CONHALVING is PPAD-hard.*

Proof. Recall that in the proof of Lemma 2, ϵ' was constrained to be at most $\min\{1/40, \epsilon/11\}$ and in particular by Lemma 1, there exists a constant ϵ' that would make the reduction work. Recall however that we "expanded" the instance of (n, ϵ')-CONHALVING from the interval $[a, b]$ to $[0, 6N]$ for convenience, which implies that after rescaling the instance to a constant interval $[a, b]$, the allowed error ϵ' goes down to $O(1/n)$. To get a constant error ϵ', we simply multiply all valuations by N. ◀

Theorem 3 implies that although a solution with n cuts is generally desirable, it might be hard to compute, even for a relatively simple class of valuations like those used in the reduction. In fact, we can extend our results to the more general case of finding a partition with $n + k$ cuts where k is a constant.

▶ **Theorem 4.** *Let k be any constant. Then there exists a constant ϵ' such that $(n, n + k, \epsilon')$-CONHALVING is PPAD-hard.*

Proof. Let $S = (V, \mathcal{T})$ be an instance of ϵ-GCIRCUIT with N nodes, consisting of smaller identical sub-circuits $S_i = (V_i, \mathcal{T}_i)$, for $i = 1, 2, \ldots, k + 1$, with with $N/(k + 1)$ nodes each such that for all $i, j \in [k + 1]$ such that $i \neq j$, it holds that $V_i \cap V_j = \emptyset$. and $\mathcal{T}_i \cap \mathcal{T}_j = \emptyset$. In other words, the circuit S consists of $k + 1$ copies of a smaller circuit S_i that do not have any common nodes or gates. Furthermore, for convenience, assume without loss of generality that for two nodes l and m such that $u_l \in V_i$ and $u_m \in V_j$, with $i < j$, it holds that $l < m$. In other words, the labeling of the nodes is such that nodes in circuits with smaller indices have smaller indices.

Let H be the instance of (n, n, ϵ')-CONHALVING corresponding to the circuit S following the reduction described in the beginning of the section and recall that $n = 2N$ in the construction. Note that according to the convention adopted above for the labeling of the nodes, for $i < j$, the agents corresponding to V_i lie in the interval $[\ell_i, r_i]$, whereas the agents corresponding to V_j lie in the interval $[\ell_j, r_j]$ and $r_i \leq \ell_j$. In other words, agents corresponding to sub-circuits with smaller indices are placed before agents with higher indices, and there is no overlap between agents corresponding to different sub-circuits.

[7] We can in fact assume some $\epsilon \leq 11/40$, as the smaller the ϵ, the harder the problem is, since we are interested in establishing hardness for some constant ϵ.

Now suppose that we have a solution to $(n, n+k, \epsilon')$-ConHalving. Since there is no overlap between valuations corresponding to different sub-circuits, an approximate solution with $n+k$ cuts for the instance H implies that there exists some interval $[\ell_i, r_i]$ corresponding to the set of nodes V_i of sub-circuit S_i, such that at least $n/(k+1)$ cuts lie in $[\ell_i, r_i]$, otherwise the total number of cuts on H would be at least $n+k+1$. Since there are exactly $n/(k+1)$ agents with valuations on $[\ell_i, r_i]$, this would imply an approximate solution for n' agents with n' cuts and the problem reduces to (n, n, ϵ')-ConHalving. ◀

4 Consensus-Halving with $n-1$ cuts is NP-hard

We have proved that the problem of finding an approximate solution with n players and n cuts is PPAD-complete. For n players and $n-1$ cuts however, we no longer have a guarantee that a solution exists. We prove that deciding whether this is the case or not is NP-hard.

▶ **Theorem 5.** *There exists a constant $\epsilon' > 0$ such that $(n, n-1, \epsilon')$-ConHalving is NP-hard.*

Proof. We will first describe the construction that we will use in the reduction. For consistency with the previous section, we will denote the error of the Consensus-halving problem by ϵ' and the error of the (implict) generalized circuits by functions of ϵ. Let $R_\epsilon(S)$ be the construction for the reduction of Section 3, that encodes an ϵ-generalized circuit S into an $(n, n-1, \epsilon')$-ConHalving instance when $\epsilon' < \epsilon/11$. We will reduce from 3-SAT, which is known to be NP-complete.

Let ϕ be any 3-SAT formula with m clauses, $k \leq 3m$ variables $x_1, \ldots, x_k$, and let $\epsilon > 0$ be given. For convenience of notation, let $\delta = \epsilon/11$. We will (implicitly) create a generalized circuit S with the following building blocks:

- k input nodes $x_1, \ldots, x_k$ corresponding to the variables $x_1, \ldots, x_k$.
- k sub-circuits $\text{Bool}(x_i)$ for $i = 1, 2, \ldots, k$ that input the real value $x_i \in [0, 1]$ and output a boolean value $x_i^{bool} \in [0, 4\delta] \cup [1-4\delta, 1]$ (see the lower stage of Figure 2). The allowed error for these circuits will be δ. The implementation of the circuit in terms of the gates of the generalized circuit can be seen in Algorithm 1. Note that the sub-circuit containing all the $\text{Bool}(x_i)$ sub-circuits has at most $4k$ nodes as each $\text{Bool}(x_i)$ sub-circuit could be implemented with one constant gate, one subtraction gate, one addition gate and one equality gate; the latter is to maintain the convention that all inputs to each gate are distinct.
- A sub-circuit $\Phi(x_1^{bool}, \ldots, x_k^{bool})$ that implements the formula ϕ, inputing the boolean variables x_i^{bool} and outputting a value x_{out} corresponding to the value of the assignment plus the error introduced by the approximate gates. The allowed error for this circuit will be 4δ. A pictorial representation of such a circuit can be seen in Figure 2; note that the circuits $\text{Bool}(x_i)$ are also shown in the picture. This circuit has at most $k+3m$ nodes. First, there might be k possible negation gates to negate the input variables. Secondly, for each clause, in order to implement an OR gate of fan-in 3, we need 2 OR gates of fan-in 2, for a total of $2m$ gates for all clauses. Finally, in order to simulate the AND gate with fan-in m, we need m AND gates of fan-in 2. Overall, since $k \leq 3m$, we need at most $6m$ nodes to implement this sub-circuit, using elements of the generalized circuit.
- A sub-circuit $\text{Rebool}(x_1, \ldots, x_k, x_{out})$ that inputs the variables x_i, for $i = 1, 2, \ldots, k$ and the variable x_{out} and computes the function

$$\min(x_{out}, \max(x_1, 1-x_1), \ldots, \max(x_k, 1-x_k)).$$

Algorithm 1 Computing $bool(x)$.

$a \leftarrow x - 1/4$
$bool \leftarrow a + a$

Algorithm 2 Computing $\min(x, y)$ and $\max(x, y)$.

$a \leftarrow x - y$; $b \leftarrow y - x$; $c \leftarrow a + b$
$d \leftarrow c/2$; $\ell \leftarrow (x/2) + (y/2)$
$min \leftarrow \ell - d$; $max \leftarrow \ell + d$

The function can be computed using the gates of the generalized circuit as shown in Algorithm 2. Let x_{out}^{bool} be the output of that sub-circuit with allowed error 4δ. Note that this circuit has at most $16k$ nodes. Each min and max operation requires 8 nodes and we need to do $2k$ such computations overall; k for the k max operations and k to implement the min operation of fan-in k with min operations of fan-in 2. Again, since $k \leq 3m$, this sub-circuit requires at most $48m$ nodes in total.

Following the notation introduced above, let $R_\delta(\text{Bool})$, $R_{4\delta}(\Phi)$ and $R_{4\delta}(\text{Rebool})$ denote the valuations of the agents in the instance of Consensus-halving corresponding to those sub-circuits, according to the reduction described in Section 3. In other words, based on the circuit described above, we create an instance H of Consensus-halving where we have:

- $2k$ agents (as each node corresponds to two agents, var_i and $copy_i$) that correspond to the input variables $x_1, \ldots, x_k$, who are not the output of any gate
- at most $2(4k + k + 3m + 16k)$ nodes corresponding to the internal nodes and the output node of the circuit.
- an additional agent with valuation

$$u_n = \begin{cases} 1, \text{if } t \in [b - 18m\epsilon' - 1, b - 18m\epsilon'] \\ 1, \text{if } t \in [b, b + 1] \\ 0, \text{otherwise} \end{cases}$$

where $[a, b]$ is the interval where the value of x_{out}^{bool} is "read" in the instance of Consensus-halving, i.e. the interval where the cut $t_{out}^{bool}-$ will be placed in the Consensus-halving solution.

Recall Definition 1 from Section 3 and note that as far as agent n is concerned, any cut $t_{out}^{bool}-$ such that $1 - 18m\epsilon \leq x_{out}^{bool} \leq 1$ is a Consensus-halving solution.

We will now argue about the correctness of the reduction. Let n be the number of agents and notice that there are $n - 1$ agents that correspond to the nodes of the circuit and a single agent constraining the value of x_{out}^{bool}. Notice that since the allowed error for the sub-circuit $\text{Rebool}(x_1, \ldots, x_k, x_{out})$ is 4δ, the total additive error of the agents of $R_{4\delta}(\text{Rebool})$ will be at most $4\delta \cdot 48m \leq 18m\epsilon'$.

First, assume that there exists a a solution to ϵ'-approximate Consensus-halving with $n-1$ cuts. By the correctness of the construction of Section 3 and the fact that $\epsilon' < \epsilon/11 = \delta$, the solution encodes a valid assignment to the variables of the generalized circuit S. Due to the valuation of agent n, the output of C must satisfy $x_{out}^{bool} \geq 1 - 18m\epsilon' - \epsilon'$, otherwise the corresponding cut $t_{out}^{bool}-$ could not be a part of a valid solution. Since the total additive error for the circuit $\text{Rebool}(x_1, \ldots, x_k, x_{out})$ is at most $18m\epsilon'$, if we choose $\epsilon' < 1/90m$, it

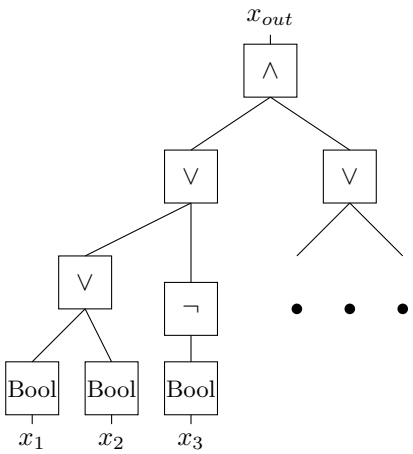


Figure 2 A generalized circuit corresponding to a 3-SAT formula ϕ, where the first clause is $(x_1 \vee x_2 \vee \overline{x_3})$. The nodes of the circuit between different layers are omitted. The layer at the output layer that "restores" the boolean values is also not shown, therefore x_{out} is the outcome of the emulated formula ϕ.

holds that $x_{out}^{bool} \geq 4/5 - \epsilon'$, which implies that $x_{out} \geq 3/4$, by the function implemented by the circuit Rebool$(x_1, \ldots, x_k, x_{out})$. For the same reason, for each $i = 1, \ldots, k$ it holds that $x_i \in [0, 1/4] \cup [3/4, 1]$, and hence the output of Bool(x_i) will lie in $[0, 4\delta] \cup [1 - 4\delta, 1]$, which means that the inputs $x_1^{bool}, \ldots, x_k^{bool}$ to the gates of the sub-circuit $\Phi(x_1^{bool}, \ldots, x_k^{bool})$ will be treated correctly as boolean values by the gates of the circuit (since the allowed error of the sub-circuit is 4δ). Since the circuit $\Phi(x_1^{bool}, \ldots, x_k^{bool})$ computes the boolean operations correctly and $x_{out} \geq 3/4$, the formula ϕ is satisfiable.

For the other direction, assume that ϕ is satisfiable and let $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_k)$ be a satisfying assignment. First we set the values of the variables $x_1, \ldots, x_k$ to 0 or 1 according to $\tilde{x}$ and then we propagate the values up the circuit S using the exact operation of the gates, which by our construction can be encoded to an instance of exact Consensus-halving for the $(n-1)$ agents corresponding to the nodes of S, i.e. the first $n-1$ will be exactly satisfied with the partition resulting from the encoded satisfying assignment. For the n-th agent, again, since the total additive error is bounded by $18m\epsilon'$, the agent will be satisfied with the solution. ◀

5 Consensus-Halving with n cuts is in PPA

In this section, we prove that (n, n, ϵ)-ConHalving is in PPA. As we discussed in the introduction, this result of ours was referenced in [13] to complement the PPA-hardness reduction of the inverse-exponential precision version and obtain PPA-completeness.

▶ **Theorem 6.** (n, n, ϵ)-ConHalving *is in PPA.*

For establishing this result, we construct a reduction from (n, n, ϵ)-ConHalving to the PPA-complete problem Leaf which goes via (n, T)-Tucker the computational version of Tucker's Lemma.

More precisely, to prove that (n, n, ϵ)-ConHalving is in PPA, we follow the main idea of the algorithm provided in [27] for obtaining a Consensus-halving solution: the coordinates of any vertex $\mathbf{x}$ in the unit cross polytope C^n naturally correspond to a partition that uses n cuts on the $[0, 1]$ interval. This is because the coordinates of any vertex $\mathbf{x} \in C^n$ satisfy

$\sum_{i=1}^{n+1} |x_i| = 1$, and a partition with n cuts on $[0,1]$ can be interpreted as partitioning the interval into $n+1$ pieces such that the length of each piece is equal to $|x_i|, i = 1, \ldots, n+1$. Furthermore, if the sign of the i-th coordinate x_i is "+", piece $|x_i|$ is assigned to portion O_+; otherwise it is assigned to portion O_-. We note that the use of the $[0,1]$ interval is for convenience and without loss of generality; for any choice of the interval we could use a a sphere of a different radius.

Given a sub-division of this sphere into small simplices (i.e. a *triangulation*) T of mesh size τ, we label each point of the triangulation by the label of the agent that is most dissatisfied by the corresponding set of cuts (and the sign indicates the direction of the discrepancy). This labelling satisfies the boundary conditions of Tucker's lemma and solutions to (n,T)-Tucker correspond to solutions of (n,n,ϵ)-ConHalving. In simple words, we show that the algorithm of [27] solves the computational version of Consensus-halving, using an algorithm for the computational version of Tucker as a subroutine.

The "in PPA" result is then established by the fact that Tucker is in PPA, i.e. it reduces to Leaf in polynomial time; this was already known from [22], where the problem is defined with respect to a subdivision of a hypercube. Technically, the algorithm of [27] that we use in our reduction requires the problem to be defined on the triangulation of a cross polytope, so one would have to prove that this version of the problem is in PPA as well. While this was already sketched in [22], we also prove it here via explaining how a constructive proof of Fan's combinatorial lemma [12] proposed by Prescott and Su [24] can be converted into a reduction to Leaf. The details along with all the necessary definitions are included in the full version.

6 Conclusion and Future Work

Our work takes an extra step in the direction of capturing the exact complexity of the Consensus-halving problem for all precision parameters. While, as we mentioned in the introduction, the techniques developed in [13] were successfully extended to obtain PPA-hardness of the problem for an inverse-polynomial precision parameter [14], it seems unlikely that they could be applicable when the precision is constant. In that sense, our main result is not implied by [13, 14], neither can it be subsumed by modifications to those reductions, even those involving highly non-trivial alterations. In other words, it seems that a PPA-completeness result for constant precision would require techniques fundamentally different from those used in [13, 14], and one can not even exclude the possibility of the problem being complete for PPAD instead.

References

1 James Aisenberg, Maria Luisa Bonet, and Sam Buss. 2-D Tucker is PPA complete. *ECCC TR15*, 163, 2015.

2 Noga Alon. Splitting necklaces. *Advances in Mathematics*, 63(3):247–253, 1987.

3 Noga Alon and Douglas B. West. The Borsuk-Ulam theorem and bisection of necklaces. *Proceedings of the American Mathematical Society*, 98(4):623–628, 1986.

4 Julius B Barbanel. Super envy-free cake division and independence of measures. *Journal of Mathematical Analysis and Applications*, 197(1):54–60, 1996.

5 Kim C Border. *Fixed point theorems with applications to economics and game theory*. Cambridge University Press, 1989.

6 Karol Borsuk. Drei Sätze über die n-dimensionale euklidische Sphäre. *Fundamenta Mathematicae*, 1(20):177–190, 1933.

7 Steven J. Brams and D. Marc Kilgour. Competitive fair division. *Journal of Political Economy*, 109(2):418–443, 2001.

8 Steven J. Brams and Alan D. Taylor. *Fair Division: From cake-cutting to dispute resolution*. Cambridge University Press, 1996.

9 Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM (JACM)*, 56(3):14, 2009.

10 Xi Chen, Dimitris Paparas, and Mihalis Yannakakis. The complexity of non-monotone markets. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC)*, pages 181–190. ACM, 2013.

11 Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

12 Ky Fan. Simplicial maps from an orientable n-pseudomanifold into Sm with the octahedral triangulation. *Journal of Combinatorial Theory*, 2(4):588–602, 1967.

13 Aris Filos-Ratsikas and Paul W. Goldberg. Consensus Halving is PPA-Complete. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC)*, pages 51–64. ACM, 2018.

14 Aris Filos-Ratsikas and Paul W. Goldberg. The Complexity of Splitting Necklaces and Bisecting Ham Sandwiches. *arXiv preprint arXiv:1805.12559*, 2018.

15 Martin Gardner. *Aha! Aha! insight*, volume 1. Scientific American, 1978.

16 Charles H. Goldberg and Douglas B. West. Bisection of circle colorings. *SIAM Journal on Algebraic Discrete Methods*, 6(1):93–106, 1985.

17 Claus-Jochen Haake, Matthias G. Raith, and Francis Edward Su. Bidding for envy-freeness: A procedural approach to n-player fair-division problems. *Social Choice and Welfare*, 19(4):723–749, 2002.

18 Charles R. Hobby and John R. Rice. A moment problem in L1 approximation. *Proceedings of the American Mathematical Society*, 16(4):665–670, 1965.

19 Nimrod Megiddo and Christos H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991.

20 Jerzy Neyman. Un theoreme d'existence. *C. R. Acad. Sci. Paris Ser. A-B 222*, pages 843–845, 1946.

21 Abraham Othman, Christos H. Papadimitriou, and Aviad Rubinstein. The complexity of fairness through equilibrium. In *Proceedings of the 15th ACM conference on Economics and Computation (EC)*, pages 209–226. ACM, 2014.

22 Christos 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.

23 Elisha Peterson and Francis Edward Su. Four-person envy-free chore division. *Mathematics Magazine*, 75(2):117–122, 2002.

24 Timothy Prescott and Francis Edward Su. A constructive proof of Ky Fan's generalization of Tucker's lemma. *Journal of Combinatorial Theory, Series A*, 111(2):257–265, 2005.

25 Jack Robertson and William Webb. *Cake-cutting algorithms: Be fair if you can*. Natick: AK Peters, 1998.

26 Aviad Rubinstein. Inapproximability of Nash equilibrium. In *Proceedings of the 47th Annual ACM on Symposium on Theory of Computing (STOC)*, pages 409–418. ACM, 2015.

27 Forest W. Simmons and Francis Edward Su. Consensus-halving via theorems of Borsuk-Ulam and Tucker. *Mathematical Social Sciences*, 45(1):15–25, 2003.

28 Emanuel Sperner. Neuer Beweis für die Invarianz der Dimensionszahl und des Gebietes. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 6 (1), pages 265–272. Springer, 1928.

29 Hugo Steinhaus. The problem of fair division. *Econometrica*, 16(1), 1948.

30 Francis Edward Su. Rental harmony: Sperner's lemma in fair division. *The American mathematical monthly*, 106(10):930–942, 1999.

31 Albert William Tucker. Some Topological Properties of Disk and Sphere. *Proc. First Canadian Math. Congress, Montreal*, pages 285—-309, 1945.

Maximum Rooted Connected Expansion

Ioannis Lamprou
Department of Computer Science, University of Liverpool, Liverpool, UK
lamprou@liverpool.ac.uk

Russell Martin
Department of Computer Science, University of Liverpool, Liverpool, UK
ramartin@liverpool.ac.uk

Sven Schewe
Department of Computer Science, University of Liverpool, Liverpool, UK
svens@liverpool.ac.uk

Ioannis Sigalas
Department of Informatics & Telecommunications, University of Athens, Athens, Greece
sigalasi@di.uoa.gr

Vassilis Zissimopoulos
Department of Informatics & Telecommunications, University of Athens, Athens, Greece
vassilis@di.uoa.gr

Abstract

Prefetching constitutes a valuable tool toward the goal of efficient Web surfing. As a result, estimating the amount of resources that need to be preloaded during a surfer's browsing becomes an important task. In this regard, prefetching can be modeled as a two-player combinatorial game [Fomin et al., *Theoretical Computer Science 2014*], where a surfer and a marker alternately play on a given graph (representing the Web graph). During its turn, the marker chooses a set of k nodes to mark (prefetch), whereas the surfer, represented as a token resting on graph nodes, moves to a neighboring node (Web resource). The surfer's objective is to reach an unmarked node before all nodes become marked and the marker wins. Intuitively, since the surfer is step-by-step traversing a subset of nodes in the Web graph, a satisfactory prefetching procedure would load in cache (without any delay) all resources lying in the neighborhood of this growing subset.

Motivated by the above, we consider the following maximization problem to which we refer to as the *Maximum Rooted Connected Expansion* (MRCE) problem. Given a graph G and a root node v_0, we wish to find a subset of vertices S such that S is connected, S contains v_0 and the ratio $\frac{|N[S]|}{|S|}$ is maximized, where $N[S]$ denotes the *closed neighborhood* of S, that is, $N[S]$ contains all nodes in S and all nodes with at least one neighbor in S.

We prove that the problem is NP-hard even when the input graph G is restricted to be a split graph. On the positive side, we demonstrate a polynomial time approximation scheme for split graphs. Furthermore, we present a $\frac{1}{6}(1-\frac{1}{e})$-approximation algorithm for general graphs based on techniques for the *Budgeted Connected Domination* problem [Khuller et al., *SODA 2014*]. Finally, we provide a polynomial-time algorithm for the special case of interval graphs. Our algorithm returns an optimal solution for MRCE in $\mathcal{O}(n^3)$ time, where n is the number of nodes in G.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases prefetching, domination, expansion, ratio

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.25

43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018).
Editors: Igor Potapov, Paul Spirakis, and James Worrell; Article No. 25; pp. 25:1–25:14

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

1 Introduction

In the evergrowing World Wide Web landscape, browsers compete against each other to offer the best quality of surfing to their users. A key characteristic in terms of quality is the speed attained when retrieving a new page or, in general, resource. Thus, a browser's objective is to minimize latency when moving from one resource to another. One way to achieve this goal is via *prefetching*: when the user lies at a certain Web node, predict what links she is more likely to visit next and preload them in cache so that, when the user selects to visit one of them, the transition appears to be instantaneous. Indeed, the World Wide Web Consortium (W3C) provides standards for prefetching in HTML [16]. Also, besides being nowadays a common practice for popular browsers, prefetching constitutes an intriguing research theme, e.g., see the surveys in [17, 1] for further references.

However, prefetching may come with a high network load cost if employed at a large scale. In other words, there is a trade-off that needs to be highlighted: more prefetching may mean less speed and even delays. For this reason, it becomes essential to acquire knowledge about the maximum number of resources to be prefetched over any potential Web nodes a surfer may visit. In this respect, Fomin et al. [4] define the *Surveillance Game* as a model for worst-case prefetching. The game is played by two players, namely the *surfer* and the *marker*, on a (directed) graph G representing (some view of) the Web graph. The surfer controls a token initially lying at a designated pre-marked start node v_0. In each round, the marker marks, i.e., prefetches, up to k so-far unmarked nodes during her turn and then the surfer chooses to move her token at a neighboring node of its current position. Notice that, once marked, a node always remains marked thereafter. The surfer wins if she arrives at an unmarked node, otherwise the marker wins if she manages to mark the whole graph before such an event occurs. In optimization terms, the quantity under consideration is the *surveillance number*, denoted $sn(G, v_0)$ for a graph G and a start (root) node v_0, which is the minimum number of marks the marker needs to use per round in order to ensure that a surfer walking on G (starting from v_0) never reaches an unmarked node.

A main observation regarding the above game is that the surfer follows some connected trajectory on the graph G. Let S stand for the set of nodes included in this trajectory. The marker's objective is to ensure that all nodes in S or in the neighborhood of S get marked promptly. Let $N[S]$ stand for the *closed neighborhood* of S, i.e., $N[S]$ includes all nodes in S and all nodes with at least one neighbor in S. Fomin et al. prove (Theorem 20 [4]) that, for any graph G and root v_0, it holds $sn(G, v_0) \geq \max\lceil \frac{|N[S]|-1}{|S|} \rceil$, where the maximum is taken over all subsets S that induce a connected subgraph of G containing v_0. Moreover, equality holds in case G is a tree. That is, a ratio of the form $|N[S]|/|S|$ (minus one and ceiling operator removed for clarity) provides a good lower bound and possibly in many occasions a good prediction on the prefetching load necessary to satisfy an impatient Web surfer. Hence, in this paper, we believe it is worth to independently study the problem of determining $\max \frac{|N[S]|}{|S|}$ where the maximum is taken over all subsets S inducing a connected subgraph of G containing v_0. We refer to this problem as the *Maximum Rooted Connected Expansion* problem (shortly MRCE) since we seek to find a connected set S (containing the root v_0) maximizing its *expansion ratio* in the form of $|N[S]|/|S|$.

Except for the prefetching motivation, such a problem can stand alone as an extension to the well-studied family of domination problems. Indeed, we later use connections between our problem and a domination variant in [14] to prove certain results. Finally, notice that removing the root node requirement makes the problem trivial. Let Δ stand for the maximum degree of a given graph G. Then, a solution consisting of a single max-degree node gives

a ratio of $\Delta + 1$. In addition, the ratio is at most $\Delta + 1$, since given any connected set S consisting of k nodes, $|N[S]| \leq (\Delta + 1)k$ due to the fact that each node can contribute at most $\Delta + 1$ new neighbors (including itself).

Related Work. The Surveillance Game was introduced in [4], where it was shown that computing $sn(G, v_0)$ is NP-hard in split graphs, nonetheless, it can be computed in polynomial time in trees and interval graphs. Furthermore, in the case of trees, the MRCE ratio is proved [4] to be equal to $sn(G, v_0)$ and therefore can be computed in polynomial time. In [7], the connected variant of the problem is considered, i.e., when the set of marked nodes is required to be connected after each round. For the corresponding optimization objective, namely the *connected surveillance number* denoted $csn(G, v_0)$, it holds $csn(G, v_0) \leq \sqrt{sn(G, v_0)n}$ for any n-node graph G. The more natural online version of the problem is also considered and (unfortunately) a competitive ratio of $\Omega(\Delta)$ is shown to be the best possible.

A problem closely related to ours (as demonstrated later in Section 4) is the *Budgeted Connected Dominating Set* problem (shortly BCDS), where, given a budget of k, one must choose a connected subset of k nodes with a maximum size of closed neighborhood. This problem is shown to have a $(1 - 1/e)/13$-approximation algorithm (in general graphs) in [14].

Regarding problems dealing with some ratio of quantities, we are familiar with the *isoperimetric number* problem [10], where the objective is to *minimize* $|\partial X|/|X|$ over all node-subsets X, where ∂X denotes the set of edges with exactly one endpoint in X. *Vertex-isoperimetric* variants also exist; see for example [12, 2]. Up to our knowledge, a ratio similar to the MRCE ratio we currently examine has not been considered.

Our Results. We initiate the study for MRCE. We prove that the decision version of MRCE is NP-complete, even when the given graph G is restricted to be a split graph. For the same case, we demonstrate a polynomial-time approximation scheme running in $\mathcal{O}(n^{k+1})$ time with a constant-factor $\frac{k}{k+2}$ guarantee, for any fixed integer $k > 0$. Our algorithm exploits a growth property for MRCE and the special topology of split graphs. Moving on, we provide another algorithm for general graphs, i.e., when no assumption is made on the topology of the given graph besides it being connected. The algorithm is inspired by an approximation algorithm for BCDS [14] and achieves an approximation guarantee of $(1 - 1/e)/6$. Finally, we show that in the case of interval graphs, the MRCE ratio can be computed optimally in $\mathcal{O}(n^3)$ time for any given n-node graph.

Outline. In Section 2, we first define some necessary preliminary graph-theoretic notions and then formally define the MRCE problem. In Section 3, we present our results for split graphs. Later, in Section 4, we give the approximation algorithm for general graphs. Next, in Section 5, we demonstrate the polynomial-time algorithm for interval graphs. Finally, in Section 6 we cite some concluding remarks and further work directions.

2 Preliminaries

A graph G is denoted as a pair $(V(G), E(G))$ of the nodes and edges of G. The graphs considered are simple (neither loops nor multi-edges are allowed), connected and undirected.

Two nodes connected by an edge are called *adjacent* or *neighboring*. The *open neighborhood* of a node $v \in V(G)$ is defined as $N(v) = \{u \in V(G) : \{v, u\} \in E(G)\}$, while the *closed neighborhood* is defined as $N[v] = \{v\} \cup N(v)$. For a subset of nodes $S \subseteq V(G)$, we expand the definitions of open and closed neighborhood as $N(S) = \bigcup_{v \in S}(N(v) \setminus S)$ and $N[S] = N(S) \cup S$.

The degree of a node $v \in V(G)$ is defined as $d(v) = |N(v)|$. The minimum (resp. maximum) degree of G is denoted by $\delta(G) = \min_{v \in V(G)} d(v)$ (resp. $\Delta(G) = \max_{v \in V(G)} d(v)$).

A *clique* is a set of nodes, where there exists an edge between each pair of them. The maximum size of a clique in G, i.e., the *clique number* of G, is denoted by $\omega(G)$.

An *independent set* is a set of nodes, where there exists no edge between any pair of them. The max. size of such a set in G, i.e., the *independence number* of G, is denoted by $\alpha(G)$.

In the results to follow, we consider two specific families of graphs, namely *split* and *interval* graphs. Any necessary preliminary knowledge for these two graph families is given more formally in their corresponding sections.

Finally, let us provide a formal definition of the quantity under consideration and the decision version of the corresponding optimization problem.

▶ **Definition 1.** We define the Maximum Rooted Connected Expansion number for a graph G and a node v_0 as follows, where $Con(G, v_0) := \{S \subseteq V(G) \mid v_0 \in S \text{ and } S \text{ is connected}\}$:

$$MRCE(G, v_0) = \max_{S \in Con(G, v_0)} \frac{|N[S]|}{|S|}$$

▶ **Definition 2** ($\mathcal{MRCE}$)**.** Given a graph G, a node $v_0 \in V(G)$ and two natural numbers a, b, decide whether $MRCE(G, v_0) \geq a/b$.

When the input graph is known to be split, respectively interval, we refer to the corresponding optimization problem as *Split* $\mathcal{MRCE}$, respectively *Interval* $\mathcal{MRCE}$.

3 Split Graphs

In this section, we define split graphs and cite a useful preliminary result regarding their structure. We proceed with our results and prove that *Split* $\mathcal{MRCE}$ is NP-hard, but it can be approximated within a constant factor of $\frac{k}{k+2}$ for any fixed integer $k > 0$.

▶ **Definition 3.** A graph is split if it can be partitioned into a clique and an independent set.

Given the above definition, we denote by (I, C) a partition for a split graph G where I stands for the independent set and C for the clique. However, there may be many different ways to partition a split graph into an independent set and a clique [11].

▶ **Theorem 4** (Follows from Theorem 3.1 [3])**.** *A split graph has at most a polynomial number of partitions into a clique and an independent set. Furthermore, all these partitions can be found in polynomial time.*

3.1 Hardness

We now move onward to investigate the complexity of *Split* $\mathcal{MRCE}$. Initially, let us a define a pair of satisfiability problems we rely on in order to prove NP-hardness.

▶ **Definition 5** (3-$\mathcal{SAT}$)**.** Given a CNF formula ϕ with n variables and m clauses, where each clause is a disjunction of exactly 3 literals, decide whether ϕ is satisfiable.

▶ **Definition 6** (3-$\mathcal{SAT}_{equal}$)**.** Given a CNF formula ϕ with n variables and n clauses, where each clause is a disjunction of exactly 3 literals, decide whether ϕ is satisfiable.

To demonstrate the hardness result in a more presentable way, we employ an auxiliary reduction from 3-$\mathcal{SAT}$ to 3-$\mathcal{SAT}_{equal}$ and then a reduction from 3-$\mathcal{SAT}_{equal}$ to *Split* $\mathcal{MRCE}$.

We recall that 3-$\mathcal{SAT}$ is well-known to be NP-hard, e.g. see [5].

▶ **Lemma 7.** 3-$\mathcal{SAT}_{equal}$ *is NP-hard.*

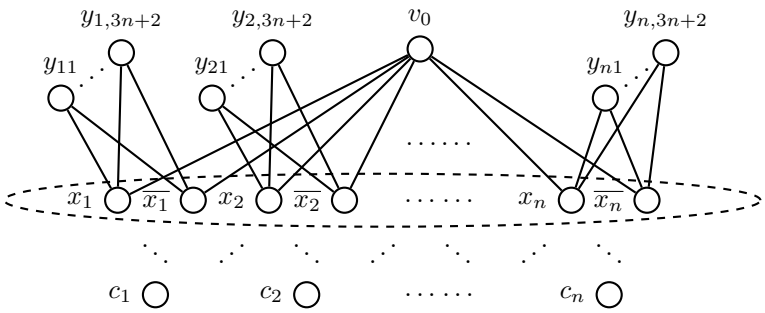


Figure 1 The graph G constructed for the reduction.

The Reduction. Given a 3-$\mathcal{SAT}_{equal}$ formula ϕ, we create a graph G with a node $v_0 \in V(G)$. Let $x_1, x_2, \ldots, x_n$ stand for the variables of ϕ and $c_1, c_2, \ldots, c_n$ for the clauses of ϕ. We construct the graph G in the following way: we place a node v_0, one node per literal $x_i, \overline{x_i}$ ($2n$ nodes in total), one node per clause c_i (n nodes in total) and a set of $3n+2$ "leaf" nodes for each variable (namely y_{ij} for $j = 1, \ldots, 3n+2$) summing up to $(3n+2) \cdot n = 3n^2 + 2n$ "leaf" nodes in total. We call the two nodes $x_i, \overline{x_i}$ a *literal-pair* and each node c_i a *clause-node*. Then, we connect v_0 to each literal node and each literal node to *all* the other literal nodes. Moreover, each literal-node is connected to all the corresponding clause-nodes where it appears in ϕ. Finally, x_i and $\overline{x_i}$ are connected to y_{ij} for all j. It is clear that the construction can be done in polynomial time. Formally, $V(G) = \{v_0\} \cup \{x_i, \overline{x_i} : 1 \le i \le n\} \cup \{c_i : 1 \le i \le n\} \cup \{y_{ij} : 1 \le i \le n, 1 \le j \le 3n+2\}$ and

$$\begin{aligned} E(G) = {} & \{[v_0, x_i] : 1 \le i \le n\} \cup \{[v_0, \overline{x_i}] : 1 \le i \le n\} \cup \\ & \cup \{[x_i, x_j] : 1 \le i, j \le n, i \ne j\} \cup \{[\overline{x_i}, x_j] : 1 \le i, j \le n, i \ne j\} \cup \{[\overline{x_i}, \overline{x_j}] : 1 \le i, j \le n, i \ne j\} \cup \\ & \cup \{[x_i, y_{ij}] : 1 \le i \le n, 1 \le j \le 3n+2\} \cup \{[\overline{x_i}, y_{ij}] : 1 \le i \le n, 1 \le j \le 3n+2\} \cup \\ & \cup \{[x_i, c_j] : x_i \text{ in clause } c_j\} \end{aligned}$$

That is, we get $|V(G)| = 1 + 5n + 3n^2$ and $|E(G)| = 2n + \binom{2n}{2} + 2n(3n+2) + 3n = 8n^2 + 8n$. Figure 1 demonstrates an example of such a construction; the literal-nodes within the dashed ellipsis form a clique.

▶ **Proposition 1.** *G is a split graph.*

Proof. $x_1, \overline{x_1}, x_2, \overline{x_2}, \ldots, x_n, \overline{x_n}$ form a clique; all other nodes form an independent set. ◀

▶ **Claim 1.** *If ϕ is satisfiable, then* $\mathrm{MRCE}(G, v_0) \ge \frac{1+5n+3n^2}{1+n}$.

Proof. Let A stand for a truth assignment under which ϕ is satisfiable. Then, to form a feasible solution for MRCE, we choose a set S including v_0 and these literal-nodes (either x_i or $\overline{x_i}$) whose corresponding literals are set true under A. Therefore, we get $|S| = 1 + n$. Since, in ϕ, each clause is satisfied by at least one literal set true under A, each clause-node c_i is connected to at least one literal-node in S. Moreover, any node y_{ij} is connected to S, since exactly one out of x_i and $\overline{x_i}$ is in S (due to A being a truth assignment). Overall, we see that $|N[S]| = |V(G)| = 1 + 5n + 3n^2$. ◀

▶ **Claim 2.** *If there exists no satisfiable assignment for ϕ, then $MRCE(G, v_0) < \frac{1+5n+3n^2}{1+n}$.*

Proof. Let us first show a proposition to restrict the shape of a feasible MRCE solution. Intuitively, adding any y_{ij} or c_i node does not contribute any new neighbors to the ratio.

Algorithm 1: Approximate Split MRCE.

Input : A split graph $G = (V(G), E(G))$, a node $v_0 \in V(G)$ and a fixed integer $k > 0$

Output : An MRCE solution and its corresponding ratio as a pair

1 $S_{apx} \leftarrow \arg\max_{S \in Con(G,v_0), 1 \leq |S| \leq k+2} |N[S]|/|S|$

2 **return** $(S_{apx}, |N[S_{apx}]|/|S_{apx}|)$

▶ **Proposition 2.** *Adding any y_{ij}, c_i node can only decrease the ratio of a feasible solution.*

The above proposition suggests it suffices to upper-bound potential solutions S containing v_0 and only literal nodes. Below, let $R = \frac{1+5n+3n^2}{1+n}$. To conclude the proof, we show that, if ϕ is unsatisfiable, then the ratio we can obtain is strictly less than R.

If $S = \{v_0\}$, then the ratio we get is $\frac{|N[\{v_0\}]|}{|\{v_0\}|} = \frac{1+2n}{1} < R$ for any $n > 0$.

If S contains v_0 and k literal nodes (any k of them), we distinguish three cases.

- *Case $k \leq n-1$:* For a fixed k, the ratio becomes at most $\frac{1+3n+k(3n+2)}{1+k}$, since at most k families of y nodes are in the neighborhood. We observe $\partial\left(\frac{1+3n+k(3n+2)}{1+k}\right)/\partial k = \frac{1}{(k+1)^2} > 0$ for any $k > 0$. Hence, the worst case is $k = n-1$, which yields a ratio $\frac{1+3n+(n-1)\cdot(3n+2)}{n} = \frac{3n^2+2n-1}{n} < R$ for any $n > 0$.
- *Case $k = n$:* If exactly one node from each literal pair is in S (i.e. S corresponds to a truth assignment), then the ratio becomes at most $\frac{1+3n-1+n(3n+2)}{1+n} < R$, since ϕ is unsatisfiable and therefore any truth assignment leaves at least one uncovered clause node. On the other hand, if there exists at least one literal-pair where both x_i and $\overline{x_i}$ are not in S, then the ratio is at most $\frac{1+3n+(n-1)(3n+2)}{1+n} < R$, since at least one set of $3n+2$ "leaf" nodes are not in $N[S]$.
- *Case $k > n$:* The ratio becomes at most $\frac{|(V(G)|}{1+k} = \frac{1+5n+3n^2}{1+k} < \frac{1+5n+3n^2}{1+n} = R$. ◀

▶ **Theorem 8.** *Split $\mathcal{MRCE}$ is NP-complete.*

3.2 Approximation

We now turn our attention to a polynomial time approximation scheme for *Split $\mathcal{MRCE}$*. Our algorithm is parameterized by any fixed integer $k > 0$ and provides an approximation guarantee of $\frac{k}{k+2}$. Intuitively, the idea is that, given the best MRCE ratio when the set size is restricted to be at most $k+2$, the overall optimal ratio cannot be much better due to a ratio growth property. Additionally, connectivity is ensured due to the special topology of split graphs. Below, the approach is described formally in Algorithm 1. Lemma 9 restricts the structure of a feasible MRCE solution on split graphs and the analysis follows in Theorem 10.

▶ **Lemma 9.** *Let G be a split graph, $v_0 \in V(G)$ the requested root node and (I, C) a partition of G into an independent set I and a clique C where $|C| = \omega(G)$. Any feasible solution for Split $\mathcal{MRCE}$ containing nodes in I can be transformed into another feasible solution with no nodes in I (except maybe for v_0) which achieves a non-decreased MRCE ratio.*

▶ **Theorem 10.** *For any fixed integer $k > 0$, Algorithm 1 runs in $\mathcal{O}(n^{k+1})$ time and returns a $\frac{k}{k+2}$-approximation for Split $\mathcal{MRCE}$.*

Proof. The algorithm computes a maximum value out of all connected subsets of size at most $k+2$, including v_0, and so it runs in $\mathcal{O}(n^{k+1})$ time.

Let S_{opt} stand for an optimal solution for *Split* $\mathcal{MRCE}$. In other words, it holds $S_{opt} \in \arg\max_{S \in Con(G,v_0)} |N[S]|/|S|$. We distinguish two cases based on the size of S_{opt}.

If $|S_{opt}| \leq k+2$, then Algorithm 1 considers S_{opt} and either returns it or another solution achieving the same ratio.

If $|S_{opt}| > k+2$, then consider the following procedure: repeatedly remove from S_{opt} the node with the least contribution in the numerator until k nodes are left. More formally, let us denote $|S_{opt}| = l$ and then $S_{opt} = S_l$. For $i = l-1, \ldots, k$, let $S_i = S_{i+1} \setminus \{u_{i+1}\}$ for some node u_{i+1} that maximizes $|N[S_{i+1} \setminus \{v\}]|$ over all $v \in S_{i+1}$. Equivalently, let $p(v) = |N[S_{i+1}]| - |N[S_{i+1} \setminus \{v\}]|$ denote the number of exclusive neighbors of v in $N[S_{i+1}]$. Then, $u_{i+1} \in \arg\min_{v \in S_{i+1}} p(v)$. Notice that, for any $i = l-1, \ldots, k$, it may be the case that S_i is *not* a feasible MRCE solution, since v_0 may be removed during this process.

Now, let us show that the ratio does not decrease while performing the above process. For any $i \in \{l-1, \ldots, k\}$, let $|N[S_i]| = N_i$ and $|S_i| = n_i$. Assume $\frac{N_{i+1}}{n_{i+1}} > \frac{N_i}{n_i}$. We rewrite the inequality as $\frac{N_{i+1}}{n_{i+1}} > \frac{N_{i+1} - p(u_{i+1})}{n_{i+1}-1}$ which implies $p(u_{i+1}) > \frac{N_{i+1}}{n_{i+1}}$. Since u_{i+1} minimizes the value of $p(\cdot)$, it follows that, for every $v \in S_{i+1}$, $p(v) \geq p(u_{i+1})$. Furthermore, $N_{i+1} \geq \sum_{v \in S_{i+1}} p(v)$ because $N[S_{i+1}]$ includes all exclusive neighbors of each node. Putting everything together, we get $N_{i+1} \geq \sum_{v \in S_{i+1}} p(v) > \sum_{v \in S_{i+1}} \frac{N_{i+1}}{n_{i+1}} = n_{i+1} \frac{N_{i+1}}{n_{i+1}} = N_{i+1}$, a contradiction. Based on this observation, we get $\frac{N_k}{n_k} \geq \frac{N_{k+1}}{n_{k+1}} \geq \ldots \geq \frac{N_l}{n_l} = OPT$, where OPT stands for the optimal MRCE number.

From Lemma 9, we may assume without loss of generality that $S_{opt} \setminus \{v_0\} \subseteq C$. Moreover, due to the removal procedure followed, $S_k \setminus \{v_0\} \subseteq S_{opt} \setminus \{v_0\} \subseteq C$. In the worst case, when $v_0 \in I$ and v_0 has no neighbor in S_k, we form $S' = S_k \cup \{v_0, r\}$ where $r \in N(v_0)$ is a representative of v_0 in the clique C such that $S_k \subseteq N(r)$. Notice that, since $S' \supseteq S_k$, then $N[S'] \supseteq N[S_k]$. Since $|S'| = k+2$, S' is considered by Algorithm 1 and therefore it holds $\frac{|N[S_{apx}]|}{|S_{apx}|} \geq \frac{|N[S']|}{|S'|}$ where S_{apx} is the solution returned by Algorithm 1. Overall, we get the approximation guarantee $\frac{|N[S_{apx}]|}{|S_{apx}|} \geq \frac{|N[S']|}{|S'|} \geq \frac{|N[S_k]|}{k+2} = \frac{k}{k+2}\frac{|N[S_k]|}{k} \geq \frac{k}{k+2}\frac{N_l}{n_l} = \frac{k}{k+2} OPT$. ◀

4 General Graphs

We hereby state a constant-factor approximation algorithm for the general case when the input graph G has no specified structure. Our algorithm and analysis closely follow the work in [14] for the *Budgeted Connected Dominating Set* (shortly BCDS) problem.

In BCDS, the input is a graph G with n vertices and a natural number k and we are asked to return a *connected* subgraph, say S, of at most k vertices of G which maximizes the number of dominated vertices $|N[S]|$. Khuller et al. [14] prove that there is a $(1-1/e)/13$ approximation algorithm for BCDS. In broad lines, their algorithmic idea is to compute a greedy dominating set and its corresponding *profit function* and then obtain a connected subgraph via an approximation algorithm for the *Quota Steiner Tree* (shortly QST) problem.

▶ **Definition 11** ($\mathcal{QST}$). Given a graph G, a node profit function $p : V(G) \to \mathbb{N} \cup \{0\}$, an edge cost function $c : E(G) \to \mathbb{N} \cup \{0\}$ and a quota $q \in \mathbb{N}$, find a subtree T that minimizes $\sum_{e \in E(T)} c(e)$ subject to the condition $\sum_{v \in V(T)} p(v) \geq q$.

Evidently, both MRCE and BCDS require finding a connected subset $S \subseteq V(G)$ with many neighbors. Nonetheless, while in BCDS we only care about maximizing $|N[S]|$, in MRCE we care about maximizing $|N[S]|/|S|$ with the additional demand that $v_0 \in S$. In order to deal with this extra requirement, in this paper, we are going to employ the rooted version of QST, namely the *Rooted Quota Steiner Tree* (shortly RQST) problem.

▶ **Definition 12** ($\mathcal{RQST}$)**.** Given a graph G, a root $v_0 \in V(G)$, a profit function $p : V(G) \to \mathbb{N} \cup \{0\}$, an edge cost function $c : E(G) \to \mathbb{N} \cup \{0\}$ and a quota $q \in \mathbb{N}$, find a subtree T that minimizes $\sum_{e \in E(T)} c(e)$ subject to the conditions $\sum_{v \in V(T)} p(v) \geq q$ and $v_0 \in T$.

Garg [6] gave a 2-approximation algorithm for the (rooted) *k-Minimum Spanning Tree* (shortly k-MST) problem based on the Goemans-Williamson *Prize-Collecting Steiner Tree* approximation algorithm (shortly GW) [8, 9]. Johnson et al. [13] showed that any polynomial-time α-approximation algorithm for (rooted) k-MST, which applies GW, yields a polynomial-time α-approximation algorithm for (rooted) QST. Hence, Theorem 13 below follows.

▶ **Theorem 13** ([6, 13])**.** *There is a* 2*-approximation algorithm for RQST.*

The Algorithm. Algorithm 2, namely the *Greedy Dominating Set* (shortly GDS) algorithm, describes a greedy procedure to obtain a dominating set and a corresponding profit function for the input graph G. At each step, a node dominating the maximum number of the currently undominated vertices is chosen for addition into the dominating set.

Algorithm 3, namely the *Greedy MRCE* algorithm, makes use of GDS to obtain a dominating set for a slightly modified version of G, namely a graph G', which is the same as G with the addition of n^2 leaves to node v_0. Then, the algorithm outputs a connected subset T_i (containing v_0) for any possible size i. Finally, the subset yielding the best MRCE ratio is chosen as our approximate solution.

In terms of notation, we refer to the approximation algorithm implied by Theorem 13 as the 2-$RQST(G, v_0, p, q)$ algorithm with a graph G, a root node $v_0 \in V(G)$, a profit function $p : V(G) \to \mathbb{N} \cup \{0\}$ and a quota q as input. We omit including an edge cost function, since in our case all edges have the same cost, that is, cost 1. Furthermore, let $[n] := \{1, 2, 3, \ldots, n\}$.

Now, consider a connected set S_i of size i (which contains v_0) yielding the maximum number of dominated vertices, i.e. $S_i \in \arg\max_{S:\, S \in Con(G, v_0),\, |S|=i} |N[S]|$. We then denote $OPT_i := |N[S_i]|$ and use it in the quota parameter of 2-$RQST$ at line 4 of Greedy MRCE. Yet, in the general case, we do not know OPT_i and also such a quantity may be hard to compute. To overcome this obstacle, notice that $OPT_i \in [i, n]$ and therefore we could *guess* OPT_i, e.g., by running a sequential or binary search within the loop of Greedy MRCE and then keeping the best tree returned by 2-$RQST$. Notice that such an extra step requires at most a linear time overhead. Therefore, the running time of Greedy MRCE remains polynomial and is dominated by the running time of 2-$RQST$. For presentation purposes, we omit this extra step and assume OPT_i is known for each $i \in [n]$.

In the analysis to follow, we focus on why this specific $(1 - 1/e)OPT_i$ quota is selected and how it leads to a $(1 - 1/e)/6$ approximation factor.

Analysis. Let us consider some step i of the loop in the Greedy MRCE algorithm. Recall that $OPT_i = \max_{S:\, S \in Con(G, v_0),\, |S|=i} |N[S]|$. That is, OPT_i stands for the maximum number of dominated vertices by a connected subset of size i, which contains v_0. In the call to 2-$RQST$, notice that, although OPT_i refers to the graph G and by definition contains v_0, the profit function p (as well as the corresponding greedy dominating set D) stems from running GDS on G'. The reason for this choice is, due to the extra n^2 leaves attached to v_0 in G', to force v_0 into the greedy dominating set D and assign to it the highest profit amongst all nodes. Below, let $S_{i,G'} \in \arg\max_{S:\, S \subseteq V(G),\, |S|=i,\, S \text{ is connected}} |N[S]|$ and $OPT_i^{G'} := |N[S_{i,G'}]|$, i.e., $OPT_{i,G'}$ denotes the maximum number of nodes dominated by a size-i subset of nodes in G'.

▶ **Claim 3.** *For any $i \in [n]$, it holds $v_0 \in S_{i,G'}$.*

Algorithm 2: Greedy Dominating Set (GDS) [14].

Input : A graph $G = (V(G), E(G))$
Output : A dominating set $D \subseteq V(G)$ and a profit function $p : V(G) \to \mathbb{N} \cup \{0\}$

$D \leftarrow \emptyset$
$U \leftarrow V(G)$
foreach $v \in V(G)$ **do**
  $p(v) \leftarrow 0$
end
while $U \neq \emptyset$ **do**
  $w \leftarrow \arg\max_{v \in V(G) \setminus D} |N_U(v)|$ /* $N_U(v) = N[\{v\}] \cap U$ */
  $p(w) \leftarrow |N_U(w)|$
  $U \leftarrow U \setminus N_U(w)$
  $D \leftarrow D \cup \{w\}$
end
return (D, p)

Algorithm 3: Greedy MRCE.

Input : A graph plus node pair (G, v_0)
Output : An MRCE solution S and its corresponding ratio s

Construct G': same as G with extra n^2 leaves attached to v_0
$(D, p) \leftarrow GDS(G')$
foreach $i \in [n]$ **do**
  $T_i \leftarrow 2\text{-}RQST(G, v_0, p, (1 - \frac{1}{e})OPT_i)$
end
Let $i^* = \arg\max_{i \in [n]} |N[T_i]|/|T_i|$
return $(T_{i^*}, |N[T_{i^*}]|/|T_{i^*}|)$

Let us introduce some further notation for the proofs to follow. Let $L_1 = S_{i,G'}$ and $L_2 = N(L_1)$, that is, $OPT_{i,G'} = |L_1 \cup L_2|$. Also, let $L_3 = N(L_2) \setminus L_1$ and $R = V(G) \setminus (L_1 \cup L_2 \cup L_3)$, where R denotes the remaining vertices, i.e., those outside the three layers L_1, L_2, L_3. Let us now consider the intersection of these layers with the greedy dominating set D returned by GDS. Let $L'_j = D \cap L_j$ for $j = 1, 2, 3$ and $D'_i = \{v_1, v_2, \ldots, v_i\}$ denote the first i vertices from $L'_1 \cup L'_2 \cup L'_3$ in the order selected by the greedy algorithm. In order to bound the total profit in D'_i, we define $g_j = \sum_{k=1}^{j} p(v_k)$ as the profit we gain from the first j vertices of D'_i.

▶ **Claim 4** (Variation of Claim 1 in [14]). *It holds* $g_{j+1} - g_j \geq \frac{1}{i}(OPT_{i,G'} - g_j)$.

▶ **Lemma 14** (Variation of Lemma 5.1 in [14]). *There exists a subset* $D'_i \subseteq D$ *of size* i *with total profit at least* $(1 - \frac{1}{e})OPT_i$. *Further,* D'_i *can be connected using at most* $2i$ *Steiner nodes and contains* v_0.

▶ **Theorem 15.** *There exists a* $\frac{1}{6}(1 - \frac{1}{e})$*-approximation for MRCE in general graphs.*

Proof. For each $i \in [n]$, by Lemma 14, there exists a solution of at most $3i$ vertices with profit at least $(1 - \frac{1}{e})OPT_i$. In Algorithm 3, we run 2-*RQST*, therefore obtaining a, connected and including v_0, solution of at most $6i$ vertices with profit at least $(1 - \frac{1}{e})OPT_i$. Let APX_i

stand for the MRCE ratio of the approximate solution corresponding to T_i. Then

$$APX_i \geq \frac{(1-\frac{1}{e})OPT_i}{6i} = \frac{1}{6}\left(1-\frac{1}{e}\right)\frac{OPT_i}{i}$$

Now, let OPT stand for the optimal ratio for MRCE. Then, $OPT = \max_{i\in[n]}\left\{\frac{OPT_i}{i}\right\}$. Let i^* be the solution size returned by Algorithm 3 and $i_0 = \arg\max_{i\in[n]}\left\{\frac{OPT_i}{i}\right\}$. Then, $APX_{i^*} \geq APX_{i_0} \geq \frac{1}{6}\left(1-\frac{1}{e}\right)OPT$, which concludes the proof. ◀

5 Interval Graphs

In this section, we provide an optimal polynomial time algorithm for the special case of *interval graphs*. We commence with some useful preliminaries and then provide the algorithm and its correctness.

Preliminaries. All intervals considered in this section are defined on the real line, closed and non-trivial (i.e., not a single point). Their form is $[\alpha, \beta]$, where $\alpha < \beta$ and $\alpha, \beta \in \mathbb{R}$.

▶ **Definition 16.** A graph is called interval if it is the intersection graph of a set of intervals on the real line.

Following the above definition, each graph node corresponds to a specific interval and two nodes are connected with an edge if and only if their corresponding intervals overlap.

▶ **Definition 17.** Given an interval graph G, a realization of G (namely $I(G)$) is a set of intervals on the real line corresponding to G, where

- for each node $v \in V(G)$, the corresponding interval is given by $I(v) \in I(G)$, and
- for $v, u \in V(G)$, $I(v)$ intersects $I(u)$ if and only if $[v, u] \in E(G)$.

Notice that we can always derive a realization, where *all interval ends are distinct*. Suppose that two intervals share a common end. One need only extend one of them by $\epsilon > 0$ chosen small enough such that neighboring relationships are not altered.

Below, we provide a definition caring for the relative position of two intervals with regards to each other. Building on that, we define a partition of $V(G)$ with respect to the position of the vertices' corresponding intervals apropos of the v_0-interval.

▶ **Definition 18.** Given two intervals $x = [x_l, x_r]$ and $y = [y_l, y_r]$, we denote the following:

- $x \sqsubset y$, i.e. x is contained in y, when $x_l > y_l$ and $x_r < y_r$.
- $x \cap_L y$, i.e. x intersects y to the left, when $x_l < y_l$ and $y_l < x_r < y_r$.
- $x \cap_R y$, i.e. x intersects y to the right, when $x_r > y_r$ and $y_l < x_l < y_r$.
- $x \prec_L y$, i.e. x is strictly to the left of y, when $x_r < y_l$.
- $x \succ_R y$, i.e. x is strictly to the right of y, when $x_l > y_r$.

▶ **Definition 19.** We define the following sets:

- Let $C \coloneqq \{v \in V(G) : I(v_0) \sqsubset I(v)\}$. Notice that $v_0 \notin C$.
- Let $C' \coloneqq \{v \in V(G) : I(v) \sqsubset I(v_0)\}$. Notice that $v_0 \notin C'$.
- Let $C_L \coloneqq \{v \in V(G) : I(v) \cap_L I(v_0)\}$.
- Let $C_R \coloneqq \{v \in V(G) : I(v) \cap_R I(v_0)\}$.
- Let $L \coloneqq \{v \in V(G) : I(v) \prec_L I(v_0)\}$.
- Let $R \coloneqq \{v \in V(G) : I(v) \succ_R I(v_0)\}$.

▶ **Proposition 3.** *$(L, C_L, C', C, \{v_0\}, C_R, R)$ forms a partition of $V(G)$.*

Let us proceed with some useful propositions regarding the form of an optimal solution.

▶ **Proposition 4.** *The addition of any node $v \in C'$ to any feasible Interval $\mathcal{MRCE}$ set does not increase the solution ratio.*

Let us now show that we need only care about a specific subset of C, namely C^*, defined as $C^* := \{v \in C \mid \nexists\, v' \in C : v \neq v' \wedge I(v) \sqsubset I(v')\}$. That is, we restrict ourselves to those vertices whose corresponding intervals contain $I(v_0)$, but are not contained in any other interval. In other words, we are only interested in the intervals that *maximally* contain $I(v_0)$.

▶ **Proposition 5.** *Any feasible Interval $\mathcal{MRCE}$ solution $S \subseteq V(G)$ containing a node $v \in C \setminus C^*$ can be transformed into another feasible solution S', where $v \notin S'$, with at least the same ratio as S.*

The Algorithm. The general idea of the algorithm is to start from the feasible solution $\{v_0\}$ and then consider a family of the best out of all possible expansions, while maintaining feasibility, either moving toward the left or the right in terms of the real line. The key in this approach is that the left and right part of the graph are dealt with *independently* from each other. Of course, special care needs to be taken when other intervals contain $I(v_0)$. During this left/right subroutine, we save a series of possible expansion stop-nodes with maximal ratio. In the end, we conflate each left ratio with each right ratio and pick the combination providing the maximum one. The algorithm is given in Algorithm 4 and the other routines follow in Algorithms 5, 6. We hereby provide a short description for each function.

- *Interval*: This is the main routine. The input is an interval graph G and a starting node $v_0 \in V(G)$. The output is a solution set together with its corresponding ratio. Initially, the algorithm computes a realization $I(G)$, a partition of $V(G)$ and the *core* set C^* as defined in the preliminaries. Then, possible left and right expansions to $\{v_0\}$ are sought. These are combined to get a best solution for this case. Finally, these basic steps are repeated for each $c \in C^*$ and the best are kept in the *Sols* pool. It then suffices to calculate the max out of the best candidate solutions.
- *Expand*: This function is responsible for providing a set of possible expansions either left or right of a starting node. A direction, the starting node, the realization, the node partition and a counter are given as input. The counter serves to save different solutions in a vector, which is returned as output. Notice that the solution vector is *static*, i.e. it can be accessed by any recursive call. The main step of the function is to select a node whose interval intersects the starting interval to the requested direction. At the same time, this interval needs to be the farthest away in this direction, i.e., its left/right endpoint needs to be smaller/greater to any other candidate's. The potential expansion is saved and the function is called recursively with the new node as a start point. The process continues till no further expansion can be made, i.e., the farthest interval is reached. The returned vector does contain a no-expansion solution (case $count = 0$).
- *Combine*: This function takes as input the potential left and right expansions. It then computes a ratio for each possible combination of left and right expansions and outputs the solution and ratio pair attaining the maximum ratio for the given starting node-set.
- *MaxRatio*: This routine simply returns the maximum set-ratio pair out of a set of different such pairs.
- *Ratio*: Simply returns the MRCE ratio for a given set.

Algorithm 4: Interval.

Input : An interval graph plus node pair (G, v_0)
Output : A set-ratio pair (S, s)

$I \longleftarrow Realization(G)$
$P \longleftarrow Partition(G, I)$
$C^* \longleftarrow Core(C, I)$
$L_{sols} \longleftarrow Expand(L, v_0, I, P, 0)$
$R_{sols} \longleftarrow Expand(R, v_0, I, P, 0)$
$Sols \longleftarrow Combine(\{v_0\}, L_{sols}, R_{sols}, G)$
foreach $c \in C^*$ **do**
  $L_{sols} \longleftarrow Expand(L, c, I, P, 0)$
  $R_{sols} \longleftarrow Expand(R, c, I, P, 0)$
  $Sols \longleftarrow Sols \cup \{Combine(\{v_0, c\}, L_{sols}, R_{sols}, G)\}$
end
return $MaxRatio(Sols)$

Algorithm 5: Expand.

Input : A direction, node, realization, partition and counter $(D, v, I, P, count)$
Output : A vector of sets of nodes $Sols$

if $count == 0$ **then**
  $Sols(count) \longleftarrow \{v\}$
end
Pick v' such that $I(v')$ is the farthest interval on direction D with $I(v') \cap_D I(v)$
if $\nexists$ *such a* v' **then**
  return $Sols$
else
  $Sols(count + 1) \longleftarrow Sols(count) \cup \{v'\}$
  return $Expand(D, v', I, P, count + 1)$
end

Algorithm 6: Combine.

Input : A node-set, left/right possible solutions and graph $(S, Left, Right, G)$
Output : A set-ratio pair $(Argmax, Max)$

$(Argmax, Max) \longleftarrow (S, Ratio(S))$
foreach $l \in Left$ **do**
  foreach $r \in Right$ **do**
    if $Ratio(S \cup l \cup r) > Max$ **then**
      $(Argmax, Max) \longleftarrow (S \cup l \cup r, Ratio(S \cup l \cup r))$
    end
  end
end
return $(Argmax, Max)$

Correctness & Complexity. Lemma 20 argues about the fact that the solutions $Expand()$ ignores do not have any effect on optimality. We state the lemma for the *left* expansion case and the reader can similarly adapt it to the right expansion case. Then, we conclude with the optimality and running time of the overall procedure (Theorem 21).

▶ **Lemma 20.** *Let L_{sols} stand for the vector returned by the function call $Expand(L, v, I, P, 0)$ for some node $v \in V(G)$. For any node-set $S \subseteq C_L \cup L \cup \{v\}$ such that $v \in S$ and $S \notin L_{sols}$, there exists a set $S' \in L_{sols}$ such that $Ratio(S') \geq Ratio(S)$.*

▶ **Theorem 21.** *$Interval(G, v_0)$ optimally solves Interval $\mathcal{MRCE}$ in $\mathcal{O}(n^3)$ time.*

6 Conclusion & Further Work

We proved that MRCE is NP-complete for split graphs. We showed that, in this case, the problem admits a polynomial time approximation scheme, whereas for interval graphs we proposed a polynomial-time algorithm. For general graphs, we also gave a constant-factor approximation algorithm by exploring the relation of MRCE with BCDS [14].

The major open question is to improve the approximability of the problem on general graphs without applying BCDS techniques, but using rather MRCE properties. Another open problem is the design of an approximation algorithm for chordal graphs. Towards this direction, we notice that even for chordal graphs with a dominating clique (a superclass of split graphs), equivalently chordal graphs with diameter at most three (Theorem 2.1 [15]), the assumption that only clique nodes need to be included in a solution (Lemma 9) now fails.

References

1. Waleed Ali, Siti Mariyam Hj. Shamsuddin, and Abdul Samad Ismail. A survey of web caching and prefetching. *International Journal of Advances in Soft Computing and its Application*, 3(1), 2011.
2. Sergei L. Bezrukov, Miquel Rius, and Oriol Serra. The vertex isoperimetric problem for the powers of the diamond graph. *Discrete Mathematics*, 308:2067–2074, 2008.
3. T. Feder, P. Hell, S. Klein, and R. Motwani. List partitions. *SIAM Journal on Discrete Mathematics*, 16(3):449–478, 2003.
4. Fedor V. Fomin, Frédéric Giroire, Alain Jean-Marie, Dorian Mazauric, and Nicolas Nisse. To satisfy impatient web surfers is hard. *Theoretical Computer Science*, 526:1–17, 2014.
5. 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, 1990.
6. Naveen Garg. Saving an epsilon: A 2-approximation for the k-mst problem in graphs. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 396–402, New York, NY, USA, 2005. ACM.
7. F. Giroire, I. Lamprou, D. Mazauric, N. Nisse, S. Pérennes, and R. Soares. Connected surveillance game. *Theoretical Computer Science*, 584:131–143, 2015. Special Issue on Structural Information and Communication Complexity.
8. Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.
9. Michel X. Goemans and David P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, pages 144–191. PWS Publishing Co., Boston, MA, USA, 1997.
10. PA Golovach. Computing the isoperimetric number of a graph. *Cybernetics and Systems Analysis*, 30(3):453–457, 1994.

11 Martin Charles Golumbic. Chapter 6 - split graphs. In Martin Charles Golumbic, editor, *Algorithmic Graph Theory and Perfect Graphs*, pages 149–156. Academic Press, 1980.

12 L.H. Harper. On an isoperimetric problem for hamming graphs. *Discrete Applied Mathematics*, 95(1):285–309, 1999.

13 David S. Johnson, Maria Minkoff, and Steven Phillips. The prize collecting steiner tree problem: Theory and practice. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 760–769, Philadelphia, PA, USA, 2000.

14 Samir Khuller, Manish Purohit, and Kanthi K. Sarpatwar. Analyzing the optimal neighborhood: Algorithms for budgeted and partial connected dominating set problems. In *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 1702–1713, Philadelphia, PA, USA, 2014.

15 Dieter Kratsch, Peter Damaschke, and Anna Lubiw. Dominating cliques in chordal graphs. *Discrete Mathematics*, 128(1):269–275, 1994.

16 W3C Resource Hints, 2018. URL: `https://www.w3.org/TR/resource-hints/`.

17 Jia Wang. A survey of web caching schemes for the internet. *SIGCOMM Comput. Commun. Rev.*, 29(5):36–46, 1999.

Interactive Proofs with Polynomial-Time Quantum Prover for Computing the Order of Solvable Groups

François Le Gall[1]
Graduate School of Informatics, Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501, Japan

Tomoyuki Morimae[2]
Yukawa Institute for Theoretical Physics, Kyoto University
Kitashirakawa Oiwakecho, Sakyo-ku, Kyoto 606-8502, Japan

Harumichi Nishimura[3]
Graduate School of Informatics, Nagoya University
Chikusa-ku, Nagoya, Aichi 464-8601, Japan

Yuki Takeuchi[4]
NTT Communication Science Laboratories, NTT Corporation
3-1 Morinosato-Wakamiya, Atsugi, Kanagawa 243-0198, Japan

Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama-cho, Toyonaka, Osaka 560-8531, Japan

Abstract

In this paper we consider what can be computed by a user interacting with a potentially malicious server, when the server performs polynomial-time quantum computation but the user can only perform polynomial-time classical (i.e., non-quantum) computation. Understanding the computational power of this model, which corresponds to polynomial-time quantum computation that can be efficiently verified classically, is a well-known open problem in quantum computing. Our result shows that computing the order of a solvable group, which is one of the most general problems for which quantum computing exhibits an exponential speed-up with respect to classical computing, can be realized in this model.

2012 ACM Subject Classification Theory of computation → Quantum computation theory

Keywords and phrases Quantum computing, interactive proofs, group-theoretic problems

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.26

1 Introduction

First-generation quantum computers will be implemented in the "cloud" style, since only few groups, such as governments or huge companies, will be able to possess such expensive and high-maintenance machines. In fact, IBM has recently opened their 16-qubit machine for a cloud service [34]. In a future when many companies provide their own quantum cloud computing services, a malicious company might emerge who is trying to palm a user off with

[1] Partially supported by the JSPS KAKENHI grants No. 15H01677, No. 16H01705 and No. 16H05853.
[2] Supported by JST PRESTO No. JPMJPR176A, and the Grant-in-Aid for Young Scientists (B) No. JP17K12637 of JSPS.
[3] Partially supported by the JSPS KAKENHI grants No. 26247016, No. 16H01705 and No. 16K00015.
[4] Supported by the Program for Leading Graduate Schools: Interactive Materials Science Cadet Program.

a wrong result forged from their fake quantum computer. In addition, even if a fortunate user is interacting with a honest server, some noises in the server's gate operations might change the result. How can a user verify the correctness of the server's quantum computation? If the user has his/her own quantum computer, the user can of course check the server's result, but in this case the user may not need the cloud service in the first place. If the solution of the problem is easily verifiable (e.g., integer factoring), the user can naturally verify the correctness of the server's result, but many problems considered in quantum computing are not believed to have this property. Verifying classically and efficiently a server's quantum computation is indeed in general highly nontrivial.

It is known that if at least two servers, who are entangled but not communicating with each other, are allowed, then any problem solvable in quantum polynomial time can be verified by a classical polynomial-time user who exchanges classical messages with the servers [20, 24, 27]. However, the assumption that servers are not communicating with each other is somehow unrealistic: how can the user guarantee that remote servers are not communicating with each other?

Whether the number of the servers can be reduced to one is a well-known open problem [4]. For certain computational problems solvable in quantum polynomial time, it is known that this can be done. Simon's problem [31] and factoring [30] are trivial examples, since the answer can be directly checked in classical polynomial time. It is known that recursive Fourier sampling [10], which was the first problem that separates efficient quantum and classical computing, can be verified by a polynomial number of message exchanges with a single quantum server [23]. Moreover, it was shown that certain promise problems related to quantum circuits in the second level of the Fourier hierarchy [29] are verifiable by a classical polynomial-time user interacting with a single quantum server who sends only a single message to the user [12, 26].

Our results. In this paper we consider the problem of computing the order, i.e., the number of elements, of a finite group given as a black-box group (the concept of black-box groups is defined in Section 2). This problem is central in computational group theory, especially since the ability of computing the order makes possible to decide membership in subgroups. This problem has also been the subject of several investigations in computational complexity [1, 6, 7, 9, 32, 33]. The seminal result by Babai [6], especially, which put this problem in the complexity class AM, has been one of the fundamental motivations behind the concept of interactive proofs. Note that this is clearly a hard problem for classical computation: it is easy to show that no polynomial-time classical algorithm exists in the black-box setting, even if the input is an abelian group [9].

Most of the known quantum algorithms that achieve exponential speedups with respect to the best known classical algorithms are for group-theoretic problems, and especially problems over abelian groups. Shor's algorithm for factoring [30], for instance, actually computes the order of a cyclic black-box group. Watrous has shown that the group order problem can be solved in quantum polynomial time when the input group is solvable [33]. Since the class of solvable groups, defined in Section 2, is a large[5] class of finite groups that includes all

[5] It is known (see for instance [11]) that

$$\lim_{m\to\infty} \frac{\log \mathcal{G}_s(m)}{\log \mathcal{G}(m)} = 1,$$

where $\mathcal{G}(m)$ denotes the number of finite groups of order at most m and $\mathcal{G}_s(m)$ denotes the number of finite solvable groups of order at most m. It is even conjectured that the quotient $\mathcal{G}_s(m)/\mathcal{G}(m)$ goes to 1 when m goes to infinity, i.e., most finite groups are solvable.

abelian groups, this result significantly generalized Shor's algorithm. Watrous' algorithm can actually be seen as one of the most general results achieving an exponential speedup with respect to classical computation.

In this paper we show that the group order problem over solvable groups is also verifiable with a single server. More formally, in Section 2, where we introduce the relevant model of interactive protocols, we will introduce the notation $\mathrm{IP}[k, \mathrm{qpoly}]$ to denote the class of computational problems that are verifiable by a classical polynomial-time user interacting in k messages with a server who works in quantum polynomial time. Our main result is as follows.

▶ **Theorem 1.** *The solvable group order problem is in the complexity class* $\mathrm{IP}[3, \mathrm{qpoly}]$. *Moreover, if the set of prime factors of the order is also given as input, then the solvable group order problem is in* $\mathrm{IP}[2, \mathrm{qpoly}]$.

This result shows that for this important computational problem, the number of servers can be reduced to one as well, using a small number of messages. Note that assuming, in the second part of Theorem 1, that the set of prime factors of the order is known corresponds to several practical situations. An important example is computing the order of p-groups[6] with p known, which cannot be done in polynomial time in the classical setting [9]. The main open question is whether the number of messages can also be reduced to 2 without any assumption on the prime factors.

Other related works. In addition to the introduction of multiple servers mentioned above, there are other approaches considered in the literature for constructing verification systems for quantum computation.

First, if the user is allowed to be "slightly quantum", any problem solvable in quantum polynomial time can be efficiently verified with a single quantum server. For example, Refs. [2, 14] assume that the user can generate randomly-rotated single-qubit states, and Refs. [13, 16, 25] assume that the user can measure single-qubit states.

Second, since the class BQP (the class of decision problems that can be solved in quantum polynomial-time) is trivially in PSPACE and PSPACE = IP [21, 28], any problem in BQP can be classically verified using generic interactive proof protocols for PSPACE. In such protocols, however, the server has unbounded computational power. A tempting approach is to try to specialize these generic protocols to the class BQP, with the hope that the server's necessary computational power may be reduced. Ref. [3] made an significant first step in this direction.

Finally, it has been shown very recently that assuming that the learning with errors problem is intractable for polynomial-time quantum computation, any problem solvable in quantum polynomial time can be efficiently verified with a single quantum server and a single classical user [22].

2 Preliminaries

In this paper we assume that the reader is familiar with the standard notions of group theory (we refer to, e.g., [18] for a good introduction). All the groups considered will be finite. Given

[6] A (finite) p-group, where p is a prime, is a group of order p^r for some integer $r \geq 1$. A basic result from group theory shows that any p-group is solvable.

a group G, we use $|G|$ to denote its order (i.e., the number of elements in G), and use e to denote its identity element. Given elements $g_1, \ldots, g_r \in G$, we denote $\langle g_1, \ldots, g_r \rangle$ the subgroup of G generated by $g_1, \ldots, g_r$.

Black-box groups. We now describe the model of black-box groups. This concept, in which each group element is represented by a string and each group operation is implemented using an oracle, was first introduced by Babai and Szemerédi [9] to describe group-theoretic algorithms in the most general way, without having to concretely specify how the elements are represented and how groups operations are implemented. Indeed, any efficient algorithm in the black-box group model gives rise to an efficient concrete algorithm whenever the oracle operations can be replaced by efficient procedures. Especially, performing group operations can be done directly on the elements in polynomial time for many natural groups, including permutation groups and matrix groups where the group elements are represented by permutations and matrices, respectively. In the quantum setting, black-box groups have first been considered by Ivanyos et al. [19] and Watrous [32, 33].

A black-box group is a representation of a group G where each element of G is uniquely encoded by a binary string of a fixed length n, which is called the encoding length. The encoding length n is known. In order to be able to express the complexity of black-box group algorithms in terms of the group order $|G|$, and not in terms of the encoding length, we make the standard assumption that $n = O(\log |G|)$. Oracles are available to perform group operations. More precisely, two oracles are available. A first oracle performs the group product: given two strings representing two group elements g and h, the oracle outputs the string representing gh. The second oracle performs inversion: given a string representing an element $g \in G$, the oracle outputs the string representing the element g^{-1}. Note that the two oracles may behave arbitrarily on strings not corresponding to elements in G; this is not a problem since our protocols will never use the oracles on such strings. We say that a group G is input as a black-box if a set of strings representing generators $\{g_1, \ldots, g_s\}$ of G with $s = O(\log |G|)$ is given as input and queries to the oracles can be done at cost 1.[7] The input length is thus $sn = \mathrm{poly}(\log |G|)$.

To be able to take advantage of the power of quantum computation when dealing with black-box groups, the oracles performing the group operations have to be able to deal with quantum superpositions. Concretely, this is done as follows (see [19, 32, 33]). Let $s\colon G \to \{0,1\}^n$ denote the encoding of elements as binary strings. We assume that a quantum oracle V_G is available, such that $V_G(|s(g)\rangle|s(h)\rangle) = |s(g)\rangle|s(gh)\rangle$ for any two elements $g, h \in G$, and behaving in an arbitrary way on other inputs (i.e., strings not in $s(G)$). Another quantum oracle V'_G is also available, such that $V'_G(|s(g)\rangle|s(h)\rangle) = |s(g)\rangle|s(g^{-1}h)\rangle$ for any $g, h \in G$ and again behaving in an arbitrary way on other inputs.

Approximate sampling in black-box groups. Babai [5] proved the following result for general groups, which shows that elements of a black-box group can be efficiently sampled nearly uniformly.

▶ **Theorem 2.** *([5]) Let G be a black-box group. For any $\varepsilon > 0$, there exists a classical randomized algorithm running in time polynomial in $\log(|G|)$ and $\log(1/\varepsilon)$ that outputs an element of G such that each $g \in G$ is output with probability in range $(1/|G| - \varepsilon, 1/|G| + \varepsilon)$.*

[7] The assumption $s = O(\log |G|)$ is standard. Indeed, every group G has a generating set of size $O(\log |G|)$. Additionally, a set of generators of any size can be converted efficiently into a set of generators of size $O(\log |G|)$ by taking random products of elements [5].

Solvable groups. Before discussing solvable groups, let us introduce the following concept of polycyclic generating sequences (see [17] for details).

▶ **Definition 3.** Let G be a group. A polycyclic generating sequence of G is a sequence $(h_1, \ldots, h_t)$ of t elements from G, for some integer t, such that:
1. $\langle h_1, \ldots, h_t \rangle = G$;
2. for each $1 < j \leq t$, the subgroup $\langle h_1, \ldots, h_{j-1} \rangle$ is normal in $\langle h_1, \ldots, h_j \rangle$.

There are many equivalent definitions of solvable groups in the literature (see, e.g., [17] for a thorough discussion). In this paper we will use the following characterization: a finite group is solvable if and only if it has a polycyclic generating sequence. This characterization, which was already used by Watrous [33], is the most convenient for our purpose. As discussed in [33], for any finite solvable group G given as a black box, a polycyclic generating sequence $(h_1, \ldots, h_t)$ with $t = O(\log |G|)$ can be computed classically in polynomial time with high probability using for instance the randomized algorithm by Babai et al. [8].

Watrous showed that the order of a solvable black-box group can be computed in polynomial time in the quantum setting. We state this result in the following theorem.

▶ **Theorem 4.** *([33]) Let G be a solvable group given as a black-box group. There exists a quantum algorithm running in time* $\mathrm{poly}(\log |G|)$ *that outputs $|G|$ with probability at least* $1 - 1/\mathrm{poly}(|G|)$.

Let G be a solvable group and $(h_1, \ldots, h_t)$ be a polycyclic generating sequence of G. In the following we will write $H_j = \langle h_1, \ldots, h_j \rangle$ for each $j \in \{1, \ldots, t\}$, and for convenience write $H_0 = \{e\}$. Since H_j is obtained from H_{j-1} by adding one generator, the factor group H_j/H_{j-1} is cyclic. Let us write its order m_j. Note that the order of G is thus the product $m_1 m_2 \cdots m_t$. A fundamental (and easy to show) property of polycyclic generating sequences is the following: For any $j \in \{1, \ldots, t\}$, any element $h \in H_j$ can be written, in a unique way, as $h = h_1^{a_1} h_2^{a_2} \cdots h_j^{a_j}$ with integers $a_i \in \{0, 1, \ldots, m_i - 1\}$ for $i \in \{1, \ldots, j\}$. We call this sequence $(a_1, \ldots, a_j)$ the decomposition of h over H_j. Watrous [33] showed that the decomposition of any element can be computed efficiently in the quantum setting, which immediately leads to an efficient algorithm for membership testing in the subgroups H_j. We state these two results, separately, in the following theorem.

▶ **Theorem 5.** *([33]) Let G be a solvable group given as a black-box group and let $(h_1, \ldots, h_t)$ be a polycyclic generating sequence of G with $t = O(\log |G|)$. There exist two quantum algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ running in time polynomial in* $\log |G|$ *as follows.*
- *Algorithm $\mathcal{A}_1$ receives an integer $j \in \{1, \ldots t\}$ and an element $h \in H_j$, and outputs with probability at least* $1 - 1/\mathrm{poly}(|G|)$ *the decomposition of h over H_j.*
- *Algorithm $\mathcal{A}_2$ receives an integer $j \in \{1, \ldots t\}$ and an element $h \in G$, and decides whether $h \in H_j$ or not. The decision is correct with probability at least* $1 - 1/\mathrm{poly}(|G|)$.

Interactive proofs with efficient quantum prover. Interactive proof systems are typically described as protocols for decision problems. In this paper it will be more convenient to consider interactive proofs for computing functions, since we are interested in computing the order of the input group.[8] The definition we give below is inspired by [15].

[8] In order to be completely rigorous, we should actually define this concept for functional problems where the input is represented using oracles (since we are dealing with black-box groups where the group operation is represented by oracles). We nevertheless omit this purely technical point in the exposition.

Let $f : X \to \{0,1\}^*$ be a function, where X is a finite set. We consider protocols between a prover and a verifier, who both receives as input an element $x \in X$ and can exchange classical messages of polynomial length. At the end of the protocol, the verifier outputs either some $y \in \{0,1\}^*$ or one special element $\perp$. We say that the function f has a *k-message polynomial-time interactive proof* if there exists a k-message protocol in which the verifier works in classical polynomial time, such that the following properties hold:

1. (completeness) there is a prover P such that the verifier's output y satisfies $y = f(x)$ with probability at least $2/3$ when interacting with P;
2. (soundness) for any prover P', the verifier's output y satisfies $y \in \{f(x), \perp\}$ with probability at least $2/3$ when interacting with P'.

The prover P in the completeness condition is called the *honest prover*.

The above definition makes no assumption on the computational powers of the provers. Our main definition is obtained by restricting the computational power of the *honest* prover, i.e., the prover P in the completeness condition.

▶ **Definition 6.** A function f is in the class $\mathrm{IP}[k, \mathrm{qpoly}]$ if it has a k-message polynomial-time interactive proof where the honest prover P works in quantum polynomial time.

The notation $\mathrm{IP}[k, \mathrm{qpoly}]$ comes from its definition as a k-message interactive protocol with a prover working in quantum polynomial time (when honest). We stress that in Definition 6 there is no assumption on the computational power of P' for the soundness.

3 2-Message Protocol with Known Prime Factors

In this section we assume that the prime factors of the order of the black-box group G are known. We present a 2-message protocol in this case, which proves the second part of Theorem 1.

3.1 Preliminaries

We will need the following result in our protocol. This result essentially shows how to reduce the computation of the order of a solvable group G to the problem of deciding if its factor groups H_i/H_{i-1} have order 1 or not.

▶ **Theorem 7.** *Let G be a solvable group given as a black-box group. Let $p_1, \ldots, p_\ell$ denote the prime factors of $|G|$ and assume that the set $S = \{p_1, \ldots, p_\ell\}$ is also given as input. There exists a classical algorithm running in time polynomial in $\log|G|$ that outputs elements $h_1, \ldots, h_t \in G$, with $t = \mathrm{poly}(\log|G|)$, and t prime numbers $r_1, \ldots, r_t \in S$ such that, with probability at least $1 - 1/\mathrm{poly}(|G|)$, the following conditions hold:*

- *$(h_1, \ldots, h_t)$ is a polycyclic generating sequence of G;*
- *the order of H_i/H_{i-1} is either 1 or r_i for each $1 \le i \le t$, where we denote $H_i = \langle h_1, \ldots, h_i\rangle$ for $1 \le i \le t$ and $H_0 = \{e\}$.*

Before proving Theorem 7, let us discuss the main idea of the algorithm in this theorem. The approach is to start with an arbitrary polycyclic generating sequence and refine it by replacing each element by decreasing powers of it. Consider for instance the cyclic group of order 12, for which we have $\ell = 2$, $p_1 = 2$, $p_2 = 3$ and $|G| = 12$. Assume that we start with the polycyclic generating sequence (k_1) consisting of a unique element k_1 of order 12. We refine this sequence as (h_1, h_2, h_3) with $h_1 = k_1^{|G|/p_1} = k_1^6$, $h_2 = k_1^{|G|/p_1^2} = k_1^3$ and $h_3 = k_1^{|G|/(p_1^2 p_2)} = k_1$. This is a polycyclic generating sequence with $|H_1/H_0| = 2$, $|H_2/H_1| = 2$ and $|H_3/H_2| = 3$. The difficulty is that naturally we do not know the order $|G|$. Remember nevertheless that

we know the encoding length n of the black-box group, which is an upper bound on $\log_2 |G|$. This means that the quantity $m = p_1^n \times \ldots \times p_\ell^n$ is a multiple of the order $|G|$, and thus we can use the same approach, working with m instead of $|G|$ when refining the original polycyclic generating sequence.

Proof of Theorem 7. Let us consider the function $\lambda\colon \{1,\ldots,\ell\} \times \{1,\ldots,n\} \to \mathbb{Z}$ such that

$$\lambda(i,a) = p_i^{n-a} \times p_{i+1}^n \times \cdots \times p_\ell^n$$

for any $(i,a) \in \{1,\ldots,\ell\} \times \{1,\ldots,n\}$. Now consider the sequence

$$(\lambda(1,1),\ldots,\lambda(1,n),\lambda(2,1),\ldots,\lambda(2,n),\ldots,\lambda(\ell,1),\ldots,\lambda(\ell,n)) \tag{1}$$

consisting of ℓn integers (the integers in the sequence are strictly decreasing). Define the function $\mu\colon \{1,\ldots,\ell n\} \to \mathbb{Z}$ such that $\mu(j)$ is the j-th integer in Sequence (1). Note that $\mu(j-1)/\mu(j) \in S$ for any $j \in \{2,\ldots,\ell n\}$.

We now describe our algorithm that computes the claimed generating sequence.

We first compute a polycyclic generating sequence $(k_1,\ldots,k_{t'})$ of G with $t' = O(\log |G|)$ using the randomized polynomial-time algorithm from [8], already mentioned in Section 2, which succeeds with probability at least $1 - 1/\mathrm{poly}(|G|)$. Let us write $K_{i'} = \langle k_1,\ldots,k_{i'}\rangle$ for each $1 \le i' \le t'$, and $K_0 = \{e\}$.

We now show how to refine the polycyclic generating sequence. For each $i' \in \{1,\ldots,t'\}$, we replace $k_{i'}$ by the sequence of ℓn elements $(k_{i'}^{\mu(1)},\ldots k_{i'}^{\mu(\ell n)})$, which gives a new sequence

$$\left(k_1^{\mu(1)},\ldots,k_1^{\mu(\ell n)},k_2^{\mu(1)},\ldots,k_2^{\mu(\ell n)},\ldots,k_{t'}^{\mu(1)},\ldots,k_{t'}^{\mu(\ell n)}\right), \tag{2}$$

of $\ell n t'$ elements. Sequence (2) is a polycyclic generating sequence of G since $(k_1,\ldots,k_{t'})$ is a polycyclic generating sequence of G and $\mu(\ell n) = 1$. For any $i' \in \{1,\ldots,t'\}$, observe that

$$\left|\langle k_1^{\mu(1)},\ldots,k_{i'}^{\mu(j)}\rangle / \langle k_1^{\mu(1)},\ldots,k_{i'}^{\mu(j-1)}\rangle\right| \in \{1,\mu(j-1)/\mu(j)\} \tag{3}$$

for any $j \in \{2,\ldots \ell n\}$. Similarly for any $i' \in \{2,\ldots t'\}$ we have

$$\left|\langle k_1^{\mu(1)},\ldots,k_{i'}^{\mu(1)}\rangle / \langle k_1^{\mu(1)},\ldots,k_{i'-1}^{\mu(\ell n)}\rangle\right| \in \{1,p_1\}. \tag{4}$$

Let us rename the elements of Sequence (2) as $h_1,\ldots,h_t$, with $t = \ell n t'$. Note that $t = O(\ell(\log|G|)^2) = O((\log|G|)^3)$. Let us write $H_i = \langle h_1,\ldots,h_i\rangle$ for $1 \le i \le t$ and $K_0 = \{e\}$. For each $1 \le i \le t$, the order of H_i/H_{i-1} is either 1 or r_i, where r_i can be determined from Equations (3) and (4). More concretely, r_i is of the form $\mu(j-1)/\mu(j)$ for some j (which can be immediately computed from i) when H_i/H_{i-1} corresponds to the case of Equation (3), and $r_i = p_1$ when H_i/H_{i-1} corresponds to the case of Equation (4). Note that in both cases we have $r_i \in S$, from the property $\mu(j-1)/\mu(j) \in S$ mentioned before. ◀

3.2 The protocol

Let $S = \{p_1,\ldots,p_\ell\}$ denote the set of prime factors of $|G|$, which is given as an additional input. The protocol is given in Figure 1. The main idea is that the verifier can, using Theorem 7, compute by itself a polycyclic generating sequence $(h_1,\ldots,h_t)$ and prime numbers $r_1,\ldots,r_t$ such that $|H_i/H_{i-1}| \in \{1,r_i\}$ for each $1 \le i \le t$. This is done at Step 1 of the protocol. Note that $|G| = \prod_{i=1}^t |H_i/H_{i-1}|$. The purpose of Steps 2-5 is to decide

Input: • a black-box solvable group G with generators $\{g_1, \ldots, g_s\}$
• the set $S = \{p_1, \ldots, p_\ell\}$ of prime factors of $|G|$

1. The verifier uses the algorithm of Theorem 7 to compute elements $h_1, \ldots, h_t$ and prime numbers $r_1, \ldots, r_t \in S$. Let us write $H_i = \langle h_1, \ldots, h_i \rangle$ for each $1 \leq i \leq t$ and $H_0 = \{e\}$.
2. For each $i \in \{1, \ldots, t\}$, the verifier takes a bit $s_i \in \{0, 1\}$ uniformly at random and takes a random element $x_i \in H_{i-1}$ using the algorithm of Theorem 2 with $\varepsilon = 1/2^{2n}$ (where n represents the encoding length of the black-box group).
3. The verifier sends to the prover the elements $h_1, \ldots, h_t$ and, for each $i \in \{1, \ldots, t\}$, the element $h_i^{s_i} x_i$.
4. The prover sends to the verifier bits $b_1, \ldots, b_t$ and integers $a_{i,j}$ for $i \in \{1, \ldots t\}$ and $j \in \{1, \ldots, i-1\}$.
5. For each $i \in \{1, \ldots, t\}$ the verifier does the following:
   5.1 If $h_i = h_1^{a_{i,1}} \cdots h_{i-1}^{a_{i,i-1}}$ then set $\ell_i = 1$;
   5.2 If $h_i \neq h_1^{a_{i,1}} \cdots h_{i-1}^{a_{i,i-1}}$ and $b_i = s_i$ then set $\ell_i = r_i$;
   5.3 If neither of these two conditions holds, then abort the protocol and output $\perp$.
6. The verifier outputs the product of the ℓ_i's.

Figure 1 Our 2-message protocol computing the order of a solvable group when the prime factors of the order are known.

whether $|H_i/H_{i-1}| = 1$ or $|H_i/H_{i-1}| = r_i$, for each $i \in \{1, \ldots, t\}$, by interacting with the prover. More precisely, the verifier interacts with the prover to test, for each i, whether $h_i \in H_{i-1}$ or $h_i \notin H_{i-1}$. This requires testing non-membership in a solvable group with a polynomial-time quantum prover, which is achieved by sending (at Step 3) to the prover the element $h_i^{s_i} x_i$ for a random bit s_i and a random element x_i, and asking the prover to find the chosen bit s_i. These tests enable the verifier to decide which of the two cases holds (at Steps 5.1 and 5.2), and then to compute $|G|$ at Step 6, or to detect cheating (at Step 5.3).

3.3 Analysis of the protocol

We now analyze the protocol of Figure 1. Let $h_1, \ldots, h_t$ be the group elements and $r_1, \ldots, r_t \in S$ be the prime numbers computed at Step 1. The analysis below is done under the assumption that $(h_1, \ldots, h_t)$ is a polycyclic generating sequence of G and $|H_i/H_{i-1}| \in \{1, r_i\}$ for all $i \in \{1, \ldots, t\}$, which is true with probability at least $1 - 1/\text{poly}(|G|)$ from Theorem 7.

Let us first consider the correctness, i.e., showing that there exists a prover (working in quantum polynomial time) who enables the verifier to compute $|G|$ with high probability. This prover acts as follows. For each $i \in \{1, \ldots, t\}$, the prover checks if the element $h_i^{s_i} x_i$ received at Step 3 is in the subgroup H_{i-1}, using Algorithm $\mathcal{A}_2$ of Theorem 5. If the prover learns that this element is in H_{i-1} then the prover applies Algorithm $\mathcal{A}_1$ of Theorem 5 to obtain a decomposition $(a_{i,1}, \ldots, a_{i,i-1})$ of h_i over H_{i-1}, and sends to the verifier the bit $b_i = 0$ and these values $a_{i,1}, \ldots, a_{i,i-1}$. If the prover learns that this element is not in H_{i-1}, then the prover sends to the verifier the bit $b_i = 1$ and arbitrary values $a_{i,1}, \ldots, a_{i,i-1}$.

Let us analyze the verifier's output when interacting with the above prover. If $|H_i/H_{i-1}| = 1$ then we have $h_i \in H_{i-1}$ and thus $h_i^{s_i} x_i \in H_{i-1}$ whatever the value of s_i is. With probability at least $1 - 1/\text{poly}(|G|)$, the prover's message is thus $b_i = 0$ and $a_{i,1}, \ldots, a_{i,i-1}$ corresponding

to the decomposition of h_i over H_{i-1}, and then the verifier sets $\ell_i = 1$. If $|H_i/H_{i-1}| = r_i$ then we have $h_i \notin H_{i-1}$ and thus $h_i^{s_i} x_i \in H_{i-1}$ if and only if $s_i = 0$. With probability at least $1 - 1/\mathrm{poly}(|G|)$, the bit b_i sent by the prover satisfies $b_i = s_i$, and thus the verifier sets $\ell_i = r_i$ (since the second part of the message $a_{i,1}, \ldots, a_{i,i-1}$ cannot correspond to the decomposition of h_i over H_{i-1}). In conclusion, with probability at least $1 - 1/\mathrm{poly}(|G|)$ the output at Step 6 is

$$\prod_{i=1}^{t} \ell_i = \prod_{i=1}^{t} |H_i/H_{i-1}| = |G|.$$

Let us now consider the soundness, i.e., showing that for any prover the verifier outputs either $|G|$ or $\perp$ with high probability. It is clear that if $|H_i/H_{i-1}| = r_i$, then the prover cannot convince the verifier to set $\ell_i = 1$, since there is no set of integers $a_{i,1}, \ldots, a_{i,i-1}$ such that $h_i = h_1^{a_{i,1}} \cdots h_{i-1}^{a_{i,i-1}}$. On the other hand, if $|H_i/H_{i-1}| = 1$ then the prover cannot convince the verifier to set $\ell_i = r_i$ unless the prover is able to decide whether $s_i = 0$ or $s_i = 1$ from the element $h_i^{s_i} x_i$ received, which cannot be done with probability larger than $\frac{1}{2} + \frac{1}{2}\delta$, where

$$\delta = \frac{1}{2} \sum_{h \in H_{i-1}} \left| \Pr_{x_i}[x_i = h] - \Pr_{x_i}[h_i x_i = h] \right|$$

represents the variational distance between the two probability distributions x_i and $h_i x_i$ (seen as distributions over H_{i-1}). We have

$$\begin{aligned}
\delta &\leq \frac{1}{2} \sum_{h \in H_{i-1}} \left| \Pr[x_i = h] - \frac{1}{|H_{i-1}|} \right| + \frac{1}{2} \sum_{h \in H_{i-1}} \left| \Pr[h_i x_i = h] - \frac{1}{|H_{i-1}|} \right| \\
&= \frac{1}{2} \sum_{h \in H_{i-1}} \left| \Pr[x_i = h] - \frac{1}{|H_{i-1}|} \right| + \frac{1}{2} \sum_{h \in H_{i-1}} \left| \Pr[x_i = h_i^{-1} h] - \frac{1}{|H_{i-1}|} \right| \\
&= \sum_{h \in H_{i-1}} \left| \Pr[x_i = h] - \frac{1}{|H_{i-1}|} \right| \\
&\leq |H_{i-1}| \varepsilon \\
&\leq 1/2^n,
\end{aligned}$$

where the second inequality follows from Theorem 2 and the third inequality follows from our choice of ε and the upper bound $|G| \leq 2^n$. Thus, for any fixed i such that $|H_i/H_{i-1}| = 1$, the prover cannot convince the verifier to set $\ell_i = r_i$ with probability greater than $\frac{1}{2} + \frac{1}{2^{n+1}} = 1/2 + 1/\mathrm{poly}(|G|)$. Let us now bound the probability that the verifier's output is either $|G|$ or $\perp$. This corresponds to the probability that the verifier does not output an integer different from the order of G. Note that the verifier can output an integer not equal to the order only if the prover forces the verifier to set $\ell_i \neq |H_i/H_{i-1}|$ for at least one index i. From the above analysis, we know that this can happen with probability at most $1/2 + 1/\mathrm{poly}(|G|)$, i.e., such a cheating is detected by the verifier at Step 5.3 with probability at least $1/2 - 1/\mathrm{poly}(|G|)$, in which case the verifier immediately aborts the protocol and outputs $\perp$. Thus the overall probability that the verifier's output is either $|G|$ or $\perp$ is at least $1/2 - 1/\mathrm{poly}(|G|)$. Note finally that this probability can be amplified to reach the soundness threshold of $2/3$ used in Definition 6 by repeating the protocol of Figure 1 a constant number of times in parallel and deciding the output based on a standard threshold argument.

Input: a black-box solvable group G with generators $\{g_1, \ldots, g_s\}$

0. The prover sends to the verifier the following:
- **a.** a list of t elements $h_1, \ldots, h_t \in G$, for some $t = \text{poly}(\log |G|)$;[9]
- **b.** a list of t prime numbers $r_1, \ldots, r_t$;
- **c.** a list of integers $\alpha_{i,j}$, for $i \in \{1, \ldots, s\}$ and $j \in \{1, \ldots, t\}$;
- **d.** a list of integers $\beta_{i,j}$, for $i \in \{2, \ldots, t\}$ and $j \in \{1, \ldots, i-1\}$;
- **e.** a list of integers $\gamma_{i,j,\ell}$, for $i \in \{2, \ldots, t\}$ and $j, \ell \in \{1, \ldots, i-1\}$.

Let us write $H_i = \langle h_1, \ldots, h_i \rangle$ for each $1 \leq i \leq t$ and $H_0 = \{e\}$.

1. The verifier checks that the following equalities hold:
- **a.** $g_i = h_1^{\alpha_{i,1}} \cdots h_t^{\alpha_{i,t}}$ for all $i \in \{1, \ldots, s\}$;
- **b.** $h_i^{r_i} = h_1^{\beta_{i,1}} \cdots h_{i-1}^{\beta_{i,i-1}}$ for all $i \in \{2, \ldots, t\}$ and $h_1^{r_1} = e$;
- **c.** $h_i h_\ell h_i^{-1} = h_1^{\gamma_{i,1,\ell}} \cdots h_{i-1}^{\gamma_{i,i-1,\ell}}$ for all $i \in \{2, \ldots, t\}$ and all $\ell \in \{1, \ldots, i-1\}$.

If any of these equalities fails, then the verifier aborts the protocol and outputs $\perp$.

2. For each $i \in \{1, \ldots, t\}$, the verifier takes a bit $s_i \in \{0, 1\}$ uniformly at random and takes a random element $x_i \in H_{i-1}$ using the algorithm of Theorem 2 with $\varepsilon = 1/2^{2n}$.

3. The verifier sends, for each $i \in \{1, \ldots, t\}$, the element $h_i^{s_i} x_i$.

4. The prover sends to the verifier bits $b_1, \ldots, b_t$ and integers $a_{i,j}$ for $i \in \{1, \ldots t\}$ and $j \in \{1, \ldots, i-1\}$.

5. For each $i \in \{1, \ldots, t\}$ the verifier does the following:
- 5.1 If $h_i = h_1^{a_{i,1}} \cdots h_{i-1}^{a_{i,i-1}}$ then set $\ell_i = 1$;
- 5.2 If $h_i \neq h_1^{a_{i,1}} \cdots h_{i-1}^{a_{i,i-1}}$ and $b_i = s_i$ then set $\ell_i = r_i$;
- 5.3 If neither of these two conditions holds, then abort the protocol and output $\perp$.

6. The verifier outputs the product of the ℓ_i's.

Figure 2 Our 3-message protocol computing the order of a solvable group.

4 General 3-Message Protocol

In this section we show that when the prime factors of the order of G are not known, we can design a 3-message protocol, which proves the first part of Theorem 1.

4.1 The protocol

Our 3-message protocol, described in Figure 2, is obtained by modifying the protocol of the previous section. More precisely, Step 1 in the protocol of the previous section is replaced by two steps (Steps 0 and 1 in Figure 2): instead of having the verifier compute a polycyclic generating sequence $(h_1, \ldots, h_t)$ using Theorem 7, which requires the knowledge of the set of factors of $|G|$, in the new protocol the prover computes by itself this sequence and sends it at Step 0 to the verifier, who then checks that the sequence is really correct at Step 1. All the other steps 2-6 are exactly the same as for the protocol in Figure 1 (one small exception is Step 3, which is slightly rewritten since the polycyclic generating sequence does not need to be sent to the prover anymore).

4.2 Analysis of the protocol

Let us consider the correctness. In that case the prover first uses the algorithm of Theorem 4 to compute the order $|G|$, then factorizes it using Shor's algorithm [30] and collects the prime factors in a set S. The prover then uses the algorithm of Theorem 7 using the set S as input to obtain group elements $h_1, \ldots, h_t$ and a list of integers $r_1, \ldots, r_t \in S$ such that with probability at least $1 - 1/\text{poly}(|G|)$ the following two conditions hold:

(i) $(h_1, \ldots, h_t)$ is a polycyclic generating sequence of G, with $t = \text{poly}(\log |G|)$,

(ii) the order of H_i/H_{i-1} is either 1 or r_i for each $1 \leq i \leq t$,

where as usual we use the notation $H_i = \langle h_1, \ldots, h_i \rangle$ for any $i \in \{1, \ldots, t\}$ and the convention $H_0 = \{e\}$. These two conditions are equivalent to the following:

(a) $H_t = G$, i.e., $g_i \in H_t$ for each $i \in \{1, \ldots, s\}$;

(b) $h_i^{r_i} \in H_{i-1}$ for each $i \in \{1, \ldots, t\}$;

(c) H_{i-1} is normal in H_i for any $i \in \{2, \ldots, t\}$, i.e., $h_i h_\ell h_i^{-1} \in H_{i-1}$ for any $\ell \in \{1, \ldots, i-1\}$.

Thus, with probability at least $1 - 1/\text{poly}(|G|)$, the prover can compute the following decompositions in quantum polynomial time using Algorithm $\mathcal{A}_1$ of Theorem 5:

- a decomposition $(\alpha_{i,1}, \ldots, \alpha_{i,t})$ of g_i over H_t, for each $i \in \{1, \ldots, s\}$;
- a decomposition $(\beta_{i,1}, \ldots, \beta_{i,i-1})$ of $h_i^{r_i}$ over H_{i-1}, for each $i \in \{2, \ldots, t\}$;
- a decomposition $(\gamma_{i,1,\ell}, \ldots, \gamma_{i,i-1,\ell})$ of $h_i h_\ell h_i^{-1}$ over H_{i-1}, for each $i \in \{2, \ldots, t\}$ and each $\ell \in \{1, \ldots, i-1\}$.

At Step 0, the prover sends all these integers, along with the elements $h_1, \ldots, h_t$ and the primes $r_1, \ldots, r_t$. All the tests performed by the verifier at Step 1 then pass. The analysis of the second part of the protocol (Steps 2-6) is then exactly the same as the analysis of the protocol of Section 3.

The soundness follows by observing that passing the tests performed by the verifier at Step 1 guarantees that Conditions (a)-(c) of the previous paragraph hold. This guarantees that Conditions (i)-(ii) hold as well, and thus the soundness analysis for the second part of the protocol (Steps 2-6) is exactly the same as the analysis of the protocol of Section 3.

References

1 Scott Aaronson and Greg Kuperberg. Quantum versus classical proofs and advice. *Theory of Computing*, 3:129–157, 2007. doi:10.4086/toc.2007.v003a007.

2 Dorit Aharonov, Michael Ben-Or, Elad Eban, and Urmila Mahadev. Interactive proofs for quantum computations. *arXiv:1704.04487*, 2017.

3 Dorit Aharonov and Ayal Green. A quantum inspired proof of $P^{\#P} \subseteq IP$. *arXiv:1710.09078*, 2017.

4 Dorit Aharonov and Umesh Vazirani. Is quantum mechanics falsifiable? A computational perspective on the foundations of quantum mechanics. *arXiv:1206.3686*, 2012.

5 László Babai. Local expansion of vertex-transitive graphs and random generation in finite groups. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 164–174, 1991. doi:10.1145/103418.103440.

[9] Naturally, this is binary strings corresponding to the elements $h_1, \ldots, h_t$ (i.e., the oracle representations of these elements) that are actually sent, not the elements themselves. Note also that, to simplify the exposition, we are assuming that these strings do correspond to elements of G. To deal with a cheating prover that may send strings not corresponding to group elements, we can simply ask the prover to send a certificate of membership in G for each string (such a certificate can be computed in quantum polynomial time using the algorithms of Theorem 5).

6 László Babai. Bounded round interactive proofs in finite groups. *SIAM Journal on Discrete Mathematics*, 5(1):88–111, 1992. doi:10.1137/0405008.

7 László Babai, Robert Beals, and Ákos Seress. Polynomial-time theory of matrix groups. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 55–64, 2009. doi:10.1145/1536414.1536425.

8 László Babai, Gene Cooperman, Larry Finkelstein, Eugene M. Luks, and Ákos Seress. Fast monte carlo algorithms for permutation groups. *Journal of Computer and System Sciences*, 50(2):296–308, 1995. doi:10.1006/jcss.1995.1024.

9 László Babai and Endre Szemerédi. On the complexity of matrix group problems I. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, pages 229–240, 1984. doi:10.1109/SFCS.1984.715919.

10 Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26:1411–1473, 1997. doi:10.1137/S0097539796300921.

11 Simon R. Blackburn, Peter M. Neumann, and Geetha Venkataraman. *Enumeration of Finite Groups*. Cambridge University Press, 2017. doi:10.1017/CBO9780511542756.

12 Tommaso F. Demarie, Yungkai Ouyang, and Joseph F. Fitzsimons. Classical verification of quantum circuits containing few basis changes. *arXiv:1612.04914*, 2016.

13 Joseph F. Fitzsimons, Michael Hajdušek, and Tomoyuki Morimae. Post hoc verification of quantum computation. *Physical Review Letters*, 120:040501, 2018. doi:10.1103/PhysRevLett.120.040501.

14 Joseph F. Fitzsimons and Elham Kashefi. Unconditionally verifiable blind computation. *Physical Review A*, 96:012303, 2017. doi:10.1103/PhysRevA.96.012303.

15 Shafi Goldwasser, Ofer Grossman, and Dhiraj Holden. Pseudo-deterministic proofs. In *Proceedings of the 9th Innovations in Theoretical Computer Science Conference*, pages 17:1–17:18, 2018. doi:10.4230/LIPIcs.ITCS.2018.17.

16 Masahito Hayashi and Tomoyuki Morimae. Verifiable measurement-only blind quantum computing with stabilizer testing. *Physical Review Letters*, 115:220502, 2015. doi:10.1103/PhysRevLett.115.220502.

17 Derek F. Holt, Bettina Eick, and Eamonn A. O'Brien. *Handbook of computational group theory*. Chapman & Hall/CRC, 2005. doi:10.1201/9781420035216.

18 I. Martin Isaacs. *Finite group theory*. American Mathematical Society, 2008.

19 Gábor Ivanyos, Frédéric Magniez, and Miklos Santha. Efficient quantum algorithms for some instances of the non-abelian hidden subgroup problem. *International Journal of Foundations of Computer Science*, 14(5):723–740, 2003. doi:10.1142/S0129054103001996.

20 Zhengfeng Ji. Classical verification of quantum proofs. In *Proceedings of the 48th Annual ACM symposium on Theory of Computing*, pages 885–898, 2016. doi:10.1145/2897518.2897634.

21 Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992. doi:10.1145/146585.146605.

22 Urmila Mahadev. Classical verification of quantum computations. *arXiv:1804.01082*, 2018.

23 Mathew McKague. Interactive proofs with efficient quantum prover for recursive fourier sampling. *Chicago Journal of Theoretical Computer Science*, 2012(6), 2012. doi:10.4086/cjtcs.2012.006.

24 Mathew McKague. Interactive proofs for BQP via self-tested graph states. *Theory of Computing*, 12(3):1–42, 2016. doi:10.4086/toc.2016.v012a003.

25 Tomoyuki Morimae, Daniel Nagaj, and Norbert Schuch. Quantum proofs can be verified using only single-qubit measurements. *Physical Review A*, 93:022326, 2016. doi:10.1103/PhysRevA.93.022326.

26 Tomoyuki Morimae, Yuki Takeuchi, and Harumichi Nishimura. Merlin-Arthur with efficient quantum Merlin and quantum supremacy for the second level of the fourier hierarchy. *arXiv:1711.10605*, 2017.

27 Ben W. Reichardt, Falk Unger, and Umesh Vazirani. Classical command of quantum systems. *Nature*, 496:456–460, 2013. doi:10.1038/nature12035.

28 Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992. doi:10.1145/146585.146609.

29 Yaoyun Shi. Quantum and classical tradeoffs. *Theoretical Computer Science*, 344:335–343, 2005. doi:10.1016/j.tcs.2005.03.053.

30 Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. doi:10.1137/S0097539795293172.

31 Daniel R. Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997. doi:10.1137/S0097539796298637.

32 John Watrous. Succinct quantum proofs for properties of finite groups. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 537–546, 2000. doi:10.1109/SFCS.2000.892141.

33 John Watrous. Quantum algorithms for solvable groups. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 60–67, 2001. doi:10.1145/380752.380759.

34 https://www-03.ibm.com/press/us/en/pressrelease/52403.wss.

On the Complexity of Team Logic and Its Two-Variable Fragment

Martin Lück
Institut für Theoretische Informatik, Leibniz Universität Hannover
Appelstraße 4, 30167 Hannover, Germany
lueck@thi.uni-hannover.de

Abstract

We study the logic FO(∼), the extension of first-order logic with team semantics by unrestricted Boolean negation. It was recently shown to be axiomatizable, but otherwise has not yet received much attention in questions of computational complexity. In this paper, we consider its two-variable fragment $\mathrm{FO}^2(\sim)$ and prove that its satisfiability problem is decidable, and in fact complete for the recently introduced non-elementary class TOWER(poly). Moreover, we classify the complexity of model checking of FO(∼) with respect to the number of variables and the quantifier rank, and prove a dichotomy between PSPACE- and ATIME-ALT(exp, poly)-complete fragments. For the lower bounds, we propose a translation from modal team logic MTL to $\mathrm{FO}^2(\sim)$ that extends the well-known standard translation from modal logic ML to FO^2. For the upper bounds, we translate FO(∼) to fragments of second-order logic with PSPACE-complete and ATIME-ALT(exp, poly)-complete model checking, respectively.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic, Theory of computation → Logic

Keywords and phrases team logic, two-variable logic, complexity, satisfiability, model checking

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.27

Related Version A full version of the paper can be found at [30], `https://arxiv.org/abs/1804.04968`.

Acknowledgements I wish to thank Juha Kontinen, Heribert Vollmer and the anonymous referees for many helpful comments.

1 Introduction

In the past decades, the work of logicians has unearthed a plethora of decidable fragments of first-order logic FO. Many decidability results are rooted in a finite model property: if there exists a (computable) upper bound on the size of minimal models with respect to a class of formulas, and if the logic admits sufficiently feasible model checking, then the question of satisfiability can be settled by exhaustively searching all structures of suitable size. Prominent examples meeting the above criteria are logics with restricted quantifier prefixes, such as the BSR-fragment which contains only $\exists^*\forall^*$-sentences [34]. Others include the monadic class [27], the guarded fragment GF [2], the recently introduced separated fragment SF [36, 37], or the two-variable fragment FO^2 [31, 35, 19], which all are decidable. See also the excellent book by Börger et al. [6] for a comprehensive classification.

The above fragments all have been subject to intensive research with the purpose of further pushing the boundary of decidability. One example is the guarded fixpoint logic, μGF, which extends GF and is 2-EXPTIME-complete [18, 3]. Another is FOC^2, the extension FO^2

with counting quantifiers. Due to an exponential model property, satisfiability is NEXPTIME-complete for both FO^2 and FOC^2 [16, 33].

Another novel and very actively studied formalism is *team semantics*, introduced by Hodges [22]. At its core, it refers to the simultaneous evaluation of formulas on whole *sets* of assignments, called *teams*. This extension is conservative in the sense that the evaluation of singleton teams, which consist of a single assignment, coincides with classical Tarski semantics. Logics with team semantics offer many applications in areas such as statistics, database theory, physics, cryptography and social choice theory (see also Abramsky et al. [1]).

As a prototypical logic with team semantics, Väänänen [38] introduced *dependence logic* D. It extends FO by *dependence atoms* $=(x_1, \dots, x_n, y)$, which intuitively state that the value of y in the team functionally depends on the values of $x_1, \dots, x_n$. With respect to expressive power, D coincides with existential second-order logic. Nonetheless, its two-variable fragment D^2 was recently proved by Kontinen et al. [23] to have a NEXPTIME-complete satisfiability problem due to a satisfiability-preserving translation to FOC^2. However, D is not closed under Boolean negation, and the *validity* problem of D^2 is in fact undecidable [24], and non-arithmetical for full D [38]. By adding a negation operator $\sim$ to D, Väänänen [38] introduced *team logic* TL, which is equivalent to full second-order logic SO [25].

As a generalization of TL, we study the logic $\mathsf{FO}(\sim, \mathcal{D})$ introduced by Galliani [13, 12]. It extends FO under team semantics by a Boolean negation $\sim$ and a set $\mathcal{D}$ of so-called generalized dependence atoms (cf. [26]). We focus on FO-definable atoms, which covers the dependence atom and many other important atoms such as the independence $\perp$ [17] or inclusion atom $\subseteq$ [10]. We abbreviate $\mathsf{FO}(\sim, \emptyset)$ as $\mathsf{FO}(\sim)$. While $\mathsf{FO}(\sim)$ and D have incomparable expressive power, in terms of complexity, $\mathsf{FO}(\sim)$ is much weaker than D. In particular, unlike D it is axiomatizable [29] and its validity problem is complete for the class Σ^0_1 of recursively enumerable sets, as with ordinary FO.

As a new result, we prove in Section 4 that its two-variable fragment $\mathsf{FO}^2(\sim)$ is decidable. More precisely, we show that satisfiability and validity of $\mathsf{FO}^2(\sim)$ are complete for the recently introduced non-elementary complexity class TOWER(poly) [28]. This pushes the "decidability frontier" away from FO^2 into a new direction, and creates the curious situation that the satisfiability problem for $\mathsf{FO}^2(\sim)$ is strictly harder than for D^2, while for validity the exact opposite is the case (cf. Table 1).

On the path to decidability, we also investigate the model checking problem of $\mathsf{FO}(\sim, \mathcal{D})$. In the first-order setting, model checking in team semantics has received only little attention so far, unlike the well-understood propositional [21] and modal [9, 32, 39] variants of team logic and dependence logic. In Section 3 and 6, we fill this gap and show that model checking for $\mathsf{FO}(\sim, \mathcal{D})$ (for "well-behaved" $\mathcal{D}$) is complete for the class ATIME-ALT(exp, poly), i.e., for exponential runtime with polynomially many alternations. This complements the result of Grädel [14] that model checking for D is NEXPTIME-complete.

Finally, we also consider fragments $\mathsf{FO}^n_k(\sim, \mathcal{D})$ which have only n variables and quantifier rank k, and relate them to certain "sparse" fragments of SO which we call $\mathsf{SO}[p]$. We prove that model checking of $\mathsf{SO}[p]$ and $\mathsf{FO}^n_k(\sim, \mathcal{D})$ is only PSPACE-complete, as opposed to unrestricted SO and $\mathsf{FO}^\omega_\omega(\sim, \mathcal{D})$.

Due to space constraints, some proofs are moved to the appendix and marked with $(\star)$, and can also be found in the full version of this paper [30].

Table 1 Complexity of logics with team semantics. Completeness unless stated otherwise. $\mathcal{D}$ is a set of generalized dependency atoms, the superscript refers to the number of variables, and the subscript to the quantifier rank.

Logic	Satisfiability	Validity	References
FO^2	NEXPTIME	co-NEXPTIME	[19]
D^2	NEXPTIME	Σ^0_1-hard	[24]
$\mathsf{FO}^2(\sim)$	TOWER(poly)	TOWER(poly)	Theorem 6.6
TL^2_2	Π^0_1-hard	Σ^0_1-hard	Theorem 6.7
	Model Checking		
$\mathsf{FO}_k, \mathsf{FO}^n$	$\in$ PTIME		see, e.g., [15]
FO	PSPACE		see, e.g., [15]
$\mathsf{FO}_k(\sim, \mathcal{D}), \mathsf{FO}^n(\sim, \mathcal{D})$	PSPACE		Theorem 6.4
$\mathsf{FO}(\sim, \mathcal{D})$	ATIME-ALT(exp, poly)		Theorem 6.4

2 Preliminaries

The domain of a function f is $\mathsf{dom}\, f$. For $f\colon X \to Y$ and $Z \subseteq X$, $f{\restriction}Z$ is the restriction of f to the domain Z. The power set of X is $\mathfrak{P}(X)$. The cardinality of the natural numbers is ω. The class of recursively enumerable sets (resp. their complements) is Σ^0_1 (resp. Π^0_1).

Given a logic $\mathcal{L}$, the sets of all satisfiable and valid formulas of $\mathcal{L}$ are written $\mathsf{SAT}(\mathcal{L})$ and $\mathsf{VAL}(\mathcal{L})$, respectively. Likewise, the model checking problem $\mathsf{MC}(\mathcal{L})$ contains the tuples (A, φ) such that φ is an $\mathcal{L}$-formula and A is a model of φ.

We assume the reader to be familiar with basic complexity theory and alternating Turing machines [7]. When stating that a problem is hard or complete for a complexity class $\mathcal{C}$, we refer to logspace-computable reductions. In this paper, we require Turing machines that are restricted in both their runtime and their *alternation depth*, as introduced by Berman [4], where the alternation depth is the maximal number of alternations between existential and universal non-determinism that a given machine performs on any computation path.

In what follows, we use the tetration function $\exp_k$, defined by $\exp_0(n) := n$ and $\exp_{k+1}(n) := 2^{\exp_k(n)}$. We write $\exp(n)$ instead of $\exp_1(n)$.

▶ **Definition 2.1.** For $k \geq 0$, ATIME-ALT$(\exp_k, \mathrm{poly})$ is the class of problems decided by an alternating Turing machine with at most $p(n)$ alternations and runtime at most $\exp_k(p(n))$, for a polynomial p.

▶ **Definition 2.2.** TOWER(poly) is the class of problems that are decided by a deterministic Turing machine in time $\exp_{p(n)}(1)$ for some polynomial p.

The reader may verify that both ATIME-ALT$(\exp_k, \mathrm{poly})$ and TOWER(poly) are closed under all Boolean operations and under polynomial time resp. logspace computable reductions.

First-order Team Logic

A vocabulary τ is a set of function symbols f and predicate symbols P, with their respective arity denoted by $\mathsf{arity}(f)$ and $\mathsf{arity}(P)$. τ is called relational if it contains no function symbols. We explicitly state $= \in \tau$ if we permit equality as part of the syntax. For obvious reasons, we require that a vocabulary always contains at least one predicate or $=$.

We fix a set $\mathsf{Var} = \{x_1, x_2, \ldots\}$ of first-order variables. If $\vec{t}$ is a tuple of τ-terms, $\mathsf{Var}(\vec{t})$ is the set of variables appearing in $\vec{t}$. Formulas are interpreted in τ-structures, denoted as pairs $\mathcal{A} = (A, \tau^{\mathcal{A}})$, with the domain A of $\mathcal{A}$ also written $\mathsf{dom}\,\mathcal{A}$. We sometimes identify $\mathcal{A}$ and $\mathsf{dom}\,\mathcal{A}$ if the meaning is clear. If $s\colon X \to \mathcal{A}$, t is a τ-term, and $\mathsf{dom}\, s \supseteq \mathsf{Var}(t)$, then $t\langle s\rangle \in \mathcal{A}$ is the evaluation of t in $\mathcal{A}$ under s. Likewise, if $\vec{t} = (t_1, \ldots, t_n)$, then $\vec{t}\langle s\rangle := (t_1\langle s\rangle, \ldots, t_n\langle s\rangle)$.

A *team* T (in $\mathcal{A}$) is a set of assignments $s\colon X \to \mathcal{A}$, where X is called domain of T. If $X \supseteq \mathsf{Var}(\vec{t})$ and $\vec{t}$ is a tuple of terms, then $\vec{t}\langle T\rangle := \{\vec{t}\langle s\rangle \mid s \in T\}$. If T is a team with domain $X \supseteq Y$, then its restriction to Y is $T{\restriction}Y := \{s{\restriction}Y \mid s \in T\}$. In slight abuse of notation, we sometimes identify a tuple $\vec{x}$ with its underlying set, e.g., write $T{\restriction}\vec{x}$ for $T{\restriction}\{x_1, \ldots, x_n\}$.

If $s\colon X \to \mathcal{A}$ and $x \in \mathsf{Var}$, then $s^x_a\colon X \cup \{x\} \to \mathcal{A}$ is the assignment that maps x to a and $y \in X \setminus \{x\}$ to $s(y)$. If T is a team in $\mathcal{A}$ with domain X, then $f\colon T \to \mathfrak{P}(\mathcal{A}) \setminus \{\emptyset\}$ is called a *supplementing function* of T. It extends (or modifies) T to the *supplementing team* $T^x_f := \{\, s^x_a \mid s \in T, a \in f(s)\,\}$. If $f(s) = A$ is constant, we write T^x_A for T^x_f.

In this paper, we consider generalized dependencies in team semantics (cf. [26, 12]), but restrict ourselves to the special case of FO-definable dependencies. For this reason, in our setting, the definition boils down to the following.

▶ **Definition 2.3** (Dependencies)**.** If P is a predicate and $\tau_P = \{P, =\}$, then a τ_P-FO-formula δ is called *dependency*. Furthermore, if $\mathsf{arity}(P) = k$, then δ is also called *k-ary dependency*.

Let $\mathcal{D} = \{\delta_1, \delta_2, \ldots\}$ be a (possibly infinite) set of dependencies. Then we consider special atoms $A_i\vec{t}$, called *generalized dependency atoms*, to represent the dependencies δ_i in the syntax. The logic τ-$\mathsf{FO}(\sim, \mathcal{D})$ extends τ-FO as follows:

$$\varphi ::= \alpha \mid A_i\vec{t} \mid {\sim}\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists x\, \varphi \mid \forall x\, \varphi,$$

where α is any τ-FO-formula, $\delta_i \in \mathcal{D}$ is a k-ary dependency, $\vec{t}$ is a k-tuple of τ-terms, and $x \in \mathsf{Var}$. For easier distinction, we usually call classical FO-formulas $\alpha, \beta, \gamma, \ldots$ and reserve $\varphi, \psi, \vartheta, \ldots$ for $\mathsf{FO}(\sim, \mathcal{D})$-formulas. If $\vec{t} = (t_1, \ldots, t_n)$ and $\vec{u} = (u_1, \ldots, u_n)$ are tuples of τ-terms, then we use the shorthand $\vec{t} = \vec{u}$ for $\bigwedge_{i=1}^n t_i = u_i$.

From now on, we usually omit τ. The $\sim$-free fragment of $\mathsf{FO}(\sim, \mathcal{D})$ is $\mathsf{FO}(\mathcal{D})$, and we abbreviate $\mathsf{FO}(\sim, \emptyset)$ as $\mathsf{FO}(\sim)$.

▶ **Example 2.4.** Let $\mathsf{dep} := \{\mathsf{dep}_1, \mathsf{dep}_2, \ldots\}$ be defined by

$$\mathsf{dep}_n(R) := \forall x_1 \cdots \forall x_{n-1} \forall y \forall z (Rx_1 \cdots x_{n-1}y \wedge Rx_1 \cdots x_{n-1}z \to y = z).$$

Then dep is set of dependencies, and the corresponding atom $A_n\vec{t}$ is called n-ary *dependence atom* and is also written $=\!(t_1, \ldots, t_n)$. It holds $(\mathcal{A}, T) \vDash =\!(t_1, \ldots, t_n)$ if and only if for all $s, s' \in T$ we have that $t_1\langle s\rangle = t_1\langle s'\rangle, \ldots, t_{n-1}\langle s\rangle = t_{n-1}\langle s'\rangle$ implies $t_n\langle s\rangle = t_n\langle s'\rangle$. Likewise, for the case $n = 1$, the atom $=\!(t)$ means that t is constant, i.e., $t\langle s\rangle = t\langle s'\rangle$ for all $s, s' \in T$.

In this notation, Väänänen's dependence logic D is $\mathsf{FO}(\mathsf{dep})$, and team logic TL is $\mathsf{FO}(\sim, \mathsf{dep})$ [38]. Many other important atoms are FO-definable, such as independence [17], inclusion and exclusion [10] (see also Durand et al. [8]).

If φ is a formula, $\mathsf{Fr}(\varphi)$ and $\mathsf{Var}(\varphi)$ denote the set of free resp. of all variables in φ, with $\mathsf{Fr}(A_i\vec{t}) := \mathsf{Var}(A_i\vec{t}) := \mathsf{Var}(\vec{t})$. If $\mathsf{Fr}(\varphi) = \emptyset$, then φ is called sentence. We write $\varphi(x_1, \ldots, x_n)$ to indicate that $x_1, \ldots, x_n$ are free in φ. The *width* $\mathsf{w}(\varphi)$ of φ is $|\mathsf{Var}(\varphi)|$.

The *quantifier rank* $\mathsf{qr}(\varphi)$ of φ is 0 if φ is atomic, and otherwise defined recursively as $\mathsf{qr}({\sim}\varphi) := \mathsf{qr}(\neg\varphi) := \mathsf{qr}(\varphi)$, $\mathsf{qr}(\varphi \wedge \psi) := \mathsf{qr}(\varphi \vee \psi) := \max\{\mathsf{qr}(\varphi), \mathsf{qr}(\psi)\}$, and $\mathsf{qr}(\exists x\, \varphi) := \mathsf{qr}(\forall x\, \varphi) := \mathsf{qr}(\varphi) + 1$, respectively. The fragment of FO with formulas of width at most n

and quantifier rank at most k is FO^n_k. The corresponding fragments D^n_k, TL^n_k, $\mathsf{FO}^n_k(\sim, \mathcal{D})$, $\mathsf{FO}^n_k(\sim)$ and $\mathsf{FO}^n_k(\mathcal{D})$ are defined analogously.

We evaluate τ-$\mathsf{FO}(\sim, \mathcal{D})$-formulas φ on pairs $(\mathcal{A}, T)$ as follows, where $\mathcal{A}$ is a τ-structure and T a team in $\mathcal{A}$ with domain $X \supseteq \mathsf{Fr}(\varphi)$:

$$
\begin{aligned}
(\mathcal{A}, T) \vDash \varphi \quad &\Leftrightarrow\ \forall s \in T : (\mathcal{A}, s) \vDash \varphi \text{ (in Tarski semantics), for } \varphi \in \mathsf{FO},\\
(\mathcal{A}, T) \vDash A_i \vec{t} \quad &\Leftrightarrow\ \mathcal{A} \vDash \delta_i(\vec{t}\langle T \rangle), \text{ where } \delta_i \in \mathcal{D},\\
(\mathcal{A}, T) \vDash {\sim}\psi \quad &\Leftrightarrow\ (\mathcal{A}, T) \nvDash \psi,\\
(\mathcal{A}, T) \vDash \psi \wedge \theta \ &\Leftrightarrow\ (\mathcal{A}, T) \vDash \psi \text{ and } (\mathcal{A}, T) \vDash \theta,\\
(\mathcal{A}, T) \vDash \psi \vee \theta \ &\Leftrightarrow\ \exists S, U \subseteq T \text{ such that } T = S \cup U, (\mathcal{A}, S) \vDash \psi, \text{ and } (\mathcal{A}, U) \vDash \theta,\\
(\mathcal{A}, T) \vDash \exists x \varphi \ &\Leftrightarrow\ (\mathcal{A}, T^x_f) \vDash \varphi \text{ for some } f \colon T \to \mathfrak{P}(\mathsf{dom}\,\mathcal{A}) \setminus \{\emptyset\},\\
(\mathcal{A}, T) \vDash \forall x \varphi \ &\Leftrightarrow\ (\mathcal{A}, T^x_{\mathsf{dom}\,\mathcal{A}}) \vDash \varphi.
\end{aligned}
$$

A τ-formula φ is *satisfiable* if there exists a τ-structure $\mathcal{A}$ and team T with domain $X \supseteq \mathsf{Fr}(\varphi)$ in $\mathcal{A}$ such that $(\mathcal{A}, T) \vDash \varphi$. Likewise, φ is *valid* if $(\mathcal{A}, T) \vDash \varphi$ for all such τ-structures $\mathcal{A}$ and teams T.

The so-called *locality* property ensures that the truth of a formula, as in classical semantics, depends only on the assignments to variables that occur free in it.

▶ **Proposition 2.5** (Locality). *Let $\varphi \in \mathsf{FO}(\sim, \mathcal{D})$ and $X \supseteq \mathsf{Fr}(\varphi)$. If T is a team in $\mathcal{A}$ with domain $Y \supseteq X$, then $(\mathcal{A}, T) \vDash \varphi$ if and only if $(\mathcal{A}, T{\restriction}X) \vDash \varphi$.*

Proof. Proof by induction on φ. The base case of FO-formulas and the inductive step for $\wedge$, $\vee$, $\exists$ and $\forall$ work similarly to Galliani's proof for inclusion/exclusion logic [10, Theorem 4.22], to which the $\sim$-case can be added in the obvious manner. It remains to consider the dependence atoms $A_i\vec{t}$. As $X \supseteq \mathsf{Fr}(A_i\vec{t}) = \mathsf{Var}(\vec{t})$, clearly $\vec{t}\langle s \rangle = \vec{t}\langle s{\restriction}X \rangle$ for any $s \in T$, and consequently, $\vec{t}\langle T \rangle = \vec{t}\langle T{\restriction}X \rangle$. Hence, $\mathcal{A} \vDash \delta_i(\vec{t}\langle T \rangle)$ iff $\mathcal{A} \vDash \delta_i(\vec{t}\langle T{\restriction}X \rangle)$. ◀

Second-Order Logic

Second-order logic τ-SO (or simply SO) extends τ-FO by second-order quantifiers $\exists f$, $\forall f$, $\exists P$ and $\forall P$ for function and predicate variables. For an SO-formula α, the sets $\mathsf{Var}(\alpha)$ and $\mathsf{Fr}(\alpha)$ refer to all resp. all free variables in α (first-order or second-order). SO is evaluated on pairs $(\mathcal{A}, \mathcal{J})$, where $\mathcal{A}$ is a structure and $\mathcal{J}$ maps first-order variables x to elements $\mathcal{J}(x) \in \mathcal{A}$, function variables f to functions $\mathcal{J}(f) \colon \mathcal{A}^{\mathsf{arity}(f)} \to \mathcal{A}$, and predicate variables P to relations $\mathcal{J}(P) \subseteq \mathcal{A}^{\mathsf{arity}(P)}$. The notation $\mathcal{J}^X_Y$ for a (first-order or second-order) variable X and an element resp. function resp. relation Y is defined as in the first-order setting. Instead of $(\mathcal{A}, \mathcal{J}) \vDash \alpha(X_1, \ldots, X_n)$ and $\mathcal{J}(X_1) = \mathcal{X}_1, \ldots, \mathcal{J}(X_n) = \mathcal{X}_n$, we also write $\mathcal{A} \vDash \alpha(\mathcal{X}_1, \ldots, \mathcal{X}_n)$.

Second-order model checking, $\mathsf{MC}(\mathsf{SO})$, is decidable using a straightforward algorithm: Given a formula α and a finite input structure $\mathcal{A}$, evaluate α in recursive top-down manner, using non-deterministic guesses for the quantified elements, functions and relations, which are of exponential size with respect to $|\mathsf{dom}\,\mathcal{A}|$.

▶ **Proposition 2.6** (⋆). $\mathsf{MC}(\mathsf{SO})$ *is decidable on input $(\mathcal{A}, \mathcal{J}, \alpha)$ in time $2^{n^{\mathcal{O}(1)}}$ and with $|\alpha|$ alternations.*

If the arity of quantified functions and relations is bounded by c, then each quantified function and relation has at most $|\mathsf{dom}\,\mathcal{A}|^c$ elements and hence takes only polynomial space:

▶ **Corollary 2.7.** *Let c-SO be the fragment of SO where all quantified functions and relations have arity at most c. Then $\mathsf{MC}(c\text{-}\mathsf{SO})$ is* PSPACE*-complete.*

3 From FO(∼) to SO: Upper bounds for model checking

In this section, we present upper bounds for the model checking problem of $\mathsf{FO}(\sim, \mathcal{D})$. On that account, we assume all first-order structures $\mathcal{A}$ and teams T to be finite and to have a suitable encoding. Instead of deciding $\mathsf{MC}(\mathsf{FO}(\sim, \mathcal{D}))$ directly, we reduce it to the corresponding problem of second-order logic, $\mathsf{MC}(\mathsf{SO})$. For this purpose, we build on top of a result of Väänänen [38], which roughly speaking states that TL-formulas can efficiently be translated to SO.

However, in Väänänen's original translation [38, Theorem 8.12, p. 159] from TL to SO it is assumed that the truth in a team is preserved when taking subteams (which is not the case if $\sim$ is available), and that all variables in a formula are quantified at most once. However, in fragments $\mathsf{FO}^n(\sim, \mathcal{D})$ of finite width n, re-quantification of variables cannot be avoided in general. In what follows, we adapt the translation accordingly. Furthermore, we extend it to include generalized dependency atoms.

Suppose $\vec{x} = (x_1, \ldots, x_n)$ is a tuple of variables. In order to avoid repetitions of variables, we define the notation $\vec{x};y$ as follows: $\vec{x};y := \vec{x}$ if $y \in \vec{x}$, and $\vec{x};y := (x_1, \ldots, x_n, y)$ if $y \notin \vec{x}$. Let now $\varphi \in \mathsf{FO}(\sim, \mathcal{D})$ such that $\mathsf{Fr}(\varphi) \subseteq \vec{x}$, and R be a n-ary predicate. Then we inductively define the SO-formula $\eta^{\vec{x}}_{\varphi}(R)$ as shown below.

- If φ is a classical first-order formula, then $\eta^{\vec{x}}_{\varphi}(R) := \forall \vec{x}(R\vec{x} \to \varphi)$.
- If $\varphi = A_i(\vec{t})$ and $\delta_i \in \mathcal{D}$ is k-ary, then let $\vec{z} = (z_1, \ldots, z_k)$ be pairwise distinct variables disjoint from $\vec{x}$ and $\eta^{\vec{x}}_{\varphi}(R) := \exists S \forall \vec{z}\,(S\vec{z} \leftrightarrow (\exists \vec{x}\, R\vec{x} \wedge \vec{t} = \vec{z})) \wedge \delta_i(S)$.[1]
- If $\varphi = \sim\psi$, then $\eta^{\vec{x}}_{\varphi}(R) := \neg\eta^{\vec{x}}_{\psi}(R)$.
- If $\varphi = \psi \wedge \theta$, then $\eta^{\vec{x}}_{\varphi}(R) := \eta^{\vec{x}}_{\psi}(R) \wedge \eta^{\vec{x}}_{\theta}(R)$.
- If $\varphi = \psi \vee \theta$, then $\eta^{\vec{x}}_{\varphi}(R) := \exists S\, \exists U\ \forall \vec{x}(R\vec{x} \leftrightarrow (S\vec{x} \vee U\vec{x})) \wedge \eta^{\vec{x}}_{\psi}(S) \wedge \eta^{\vec{x}}_{\theta}(U)$.
- If $\varphi = \exists y\, \psi$, then $\eta^{\vec{x}}_{\varphi}(R) := \exists S \forall \vec{x}((\exists y R\vec{x}) \leftrightarrow (\exists y\, S\vec{x};y)) \wedge \eta^{\vec{x};y}_{\psi}(S)$.
- If $\varphi = \forall y\, \psi$, then $\eta^{\vec{x}}_{\varphi}(R) := \exists S \forall \vec{x}((\exists y R\vec{x}) \leftrightarrow (\exists y\, S\vec{x};y)) \wedge \eta^{\vec{x};y}_{\psi}(S) \wedge \forall \vec{x}(R\vec{x} \to \forall y\, S\vec{x};y)$.

By an inductive proof, the formulas φ and $\eta^{\vec{x}}_{\varphi}(R)$ can be shown equivalent, provided that the team T is represented as a relation $R := \vec{x}\langle T\rangle$:

▶ **Theorem 3.1** (⋆). *Let $\varphi \in \mathsf{FO}(\sim, \mathcal{D})$, let $\vec{x} \supseteq \mathsf{Fr}(\varphi)$ be a tuple of variables, and T be a team in $\mathcal{A}$ with domain $Y \supseteq \vec{x}$. Then $(\mathcal{A}, T) \vDash \varphi$ if and only if $\mathcal{A} \vDash \eta^{\vec{x}}_{\varphi}(\vec{x}\langle T\rangle)$.*

▶ **Definition 3.2.** We call a set $\mathcal{D} = \{\delta_1, \delta_2, \ldots\}$ of dependencies *p-uniform* if there is a polynomial time algorithm that for all i, when given $A_i\vec{t}$, computes δ_i.

▶ **Corollary 3.3.** *Let $\mathcal{D}$ be a p-uniform set of dependencies. Then $\mathsf{MC}(\mathsf{FO}(\sim, \mathcal{D}))$ is decidable on input $(\mathcal{A}, T, \varphi)$ in time $2^{n^{\mathcal{O}(1)}}$ and with $|\varphi|^{\mathcal{O}(1)}$ alternations.*

Proof. First, we compute $\vec{x} := \mathsf{Fr}(\varphi)$, the formula $\eta^{\vec{x}}_{\varphi}$ and the relation $\vec{x}\langle T\rangle$ from φ and T in polynomial time. When translating the atoms $A_i\vec{t}$, we apply the p-uniformity of $\mathcal{D}$. Afterwards, we accept if and only if $\mathcal{A} \vDash \eta^{\vec{x}}_{\varphi}(\vec{x}\langle T\rangle)$. By Proposition 2.6, the latter can be checked by an algorithm with $|\eta^{\vec{x}}_{\varphi}|$ alternations and time exponential in $(\mathcal{A}, \vec{x}\langle T\rangle, \eta^{\vec{x}}_{\varphi})$. In total, this leads to $|\varphi|^{\mathcal{O}(1)}$ alternations and runtime exponential in the size of $(\mathcal{A}, T, \varphi)$. ◀

[1] Note that the "obvious" translation $\eta^{\vec{x}}_{\varphi}(R) := \delta_i(R)$ does not work in general if $A_i(\vec{t})$ contains proper terms. For instance, any team T satisfies $=\!(c)$ for any constant term c, but R represents only $\vec{x}\langle T\rangle$, which might or might not satisfy δ_i. To properly reflect such atoms, we quantify $\vec{t}\langle T\rangle$ in the relation S; in our example, $S = \{(c)\}$ for $T \neq \emptyset$.

Sparse second-order logic

The complexity of the model checking problem of $\mathsf{FO}(\sim, \mathcal{D})$ significantly drops if either the number of variables or the quantifier rank is bounded by an arbitrary constant. To prove this, we introduce a fragment of SO that corresponds to these restricted fragments of $\mathsf{FO}(\sim, \mathcal{D})$. We call this fragment *sparse second-order logic*, based on *sparse quantifiers* $\exists^p$ and $\forall^p$:

$$(\mathcal{A}, \mathcal{J}) \vDash \exists^p P\, \psi \Leftrightarrow \text{there exists } R \subseteq \mathcal{A}^{\mathsf{arity}(P)} \text{ such that } |R| \leq p(|\mathcal{A}|) \text{ and } (\mathcal{A}, \mathcal{J}^P_R) \vDash \psi,$$
$$(\mathcal{A}, \mathcal{J}) \vDash \forall^p P\, \psi \Leftrightarrow \text{for all } R \subseteq \mathcal{A}^{\mathsf{arity}(P)} \text{ such that } |R| \leq p(|\mathcal{A}|) \text{ it holds } (\mathcal{A}, \mathcal{J}^P_R) \vDash \psi,$$

where $p\colon \mathbb{N} \to \mathbb{N}$ and $|\mathcal{A}| := |\mathsf{dom}\, \mathcal{A}| + \sum_{X \in \tau} |X^{\mathcal{A}}|$. In other words, all quantified relations have bounded cardinality relative to the underlying structure. For obvious reasons, there are no sparse function quantifiers.

The logic $\mathsf{SO}[p]$ is now defined as SO, but with only $\exists^p$ and $\forall^p$ as permitted second-order quantifiers. Consider the case where p is bounded by a polynomial. The interpretation of each quantified relation then contains at most $|\mathcal{A}|^{\mathcal{O}(1)}$ tuples. Consequently, on $\mathsf{SO}[p]$-formulas, the recursive model checking algorithm from Proposition 2.6 then runs in polynomial time:

▶ **Corollary 3.4.** *If p is bounded by a polynomial, then $\mathsf{MC}(\mathsf{SO}[p])$ is decidable on input $(\mathcal{A}, \mathcal{J}, \alpha)$ in polynomial time and with $|\alpha|$ alternations.*

It remains to show that the translation from team logic with bounded width or quantifier rank takes place in this fragment of SO. This can be seen as follows. Intuitively, every quantified relation in $\eta^{\vec{x}}_\varphi$ represents either a subteam of an existing team (for the $\vee$-case), or it is a supplementing team (for the $\exists$-case and $\forall$-case). For this reason, the cardinality of the quantified relations grows at most by a factor of $|\mathsf{dom}\, \mathcal{A}|$ for every occurrence of $\vee$, $\exists$ or $\forall$.

Now, for $p\colon \mathbb{N} \to \mathbb{N}$, define the $\mathsf{SO}[p]$-formula $\zeta^{\vec{x},p}_\varphi$ like $\eta^{\vec{x}}_\varphi$, but with all second-order quantifiers replaced by $\exists^p$. The next theorem states that $\zeta^{\vec{x},p}_\varphi$ is an appropriate translation of φ, similarly to $\eta^{\vec{x}}_\varphi$, if φ has sufficiently small width or quantifier rank:

▶ **Theorem 3.5** (⋆). *Let $\varphi \in \mathsf{FO}(\sim, \mathcal{D})$, let $\vec{x} \supseteq \mathsf{Fr}(\varphi)$ be a tuple of variables, and T be a team in $\mathcal{A}$ with domain $Y \supseteq \vec{x}$. If $p(n) \geq |T| \cdot n^{\mathsf{qr}(\varphi)}$ or $p(n) \geq n^{\mathsf{w}(\varphi)}$, then $(\mathcal{A}, T) \vDash \varphi$ if and only if $\mathcal{A} \vDash \zeta^{\vec{x},p}_\varphi(\vec{x}\langle T\rangle)$.*

Proof. By a careful analysis, it can be shown that all second-order quantifiers $\exists S$ in $\eta^{\vec{x}}_\varphi$ can be replaced by $\exists^p S$. See the appendix for details. As then $\eta^{\vec{x}}_\varphi$ and $\zeta^{\vec{x},p}_\varphi$ agree on $(\mathcal{A}, T)$, the claim follows by Theorem 3.1. ◀

▶ **Corollary 3.6.** *Let $\mathcal{D}$ be a p-uniform set of dependencies and $m < \omega$. $\mathsf{MC}(\mathsf{FO}^m_\omega(\sim, \mathcal{D}))$ and $\mathsf{MC}(\mathsf{FO}^\omega_m(\sim, \mathcal{D}))$ are then decidable on input $(\mathcal{A}, T, \varphi)$ in polynomial time with $|\varphi|^{\mathcal{O}(1)}$ alternations.*

Proof. Let $p(n) := n^{m+1}$. Analogously to Corollary 3.3, we reduce both $\mathsf{MC}(\mathsf{FO}^m_\omega(\sim, \mathcal{D}))$ and $\mathsf{MC}(\mathsf{FO}^\omega_m(\sim, \mathcal{D}))$ to $\mathsf{MC}(\mathsf{SO}[p])$. Assume that $(\mathcal{A}, T, \varphi)$ is the input, and that either $\mathsf{w}(\varphi) \leq m$ or $\mathsf{qr}(\varphi) \leq m$. Then the input is mapped to $(\mathcal{A}, \vec{x}\langle T\rangle, \zeta^{p,\vec{x}}_\varphi)$, where $\vec{x} = \mathsf{Fr}(\varphi)$.

- If $\mathsf{w}(\varphi) \leq m$, then $(\mathcal{A}, T) \vDash \varphi$ if and only if $\mathcal{A} \vDash \zeta^{p,\vec{x}}_\varphi(\vec{x}\langle T\rangle)$ by Theorem 3.5.
- If $\mathsf{qr}(\varphi) \leq m$, then w.l.o.g. $|T| \leq |\mathcal{A}|$ (if necessary, pad $\mathcal{A}$ with a dummy predicate in polynomial time). Then $|T| \cdot |\mathcal{A}|^m \leq p(|\mathcal{A}|)$, and we can again apply Theorem 3.5. ◀

4 From $\mathsf{FO}^2(\sim)$ to FO^2: Upper bounds for satisfiability

In this section, we turn to the satisfiability problem of $\mathsf{FO}^2(\sim)$ and prove that it is complete for TOWER(poly) (cf. Definition 2.2). Our approach is to establish a finite model property for $\mathsf{FO}^2(\sim)$. However, instead of constructing a finite model directly, we reduce $\mathsf{FO}^2(\sim)$-formulas to FO^2-formulas, and use the exponential model property of FO^2 [19]. As a first step, we expand $\mathsf{FO}^2(\sim)$-formulas into a specific "disjunctive normal form" over $\wedge$ and $\sim$. Recall that $\vee$ is not the Boolean disjunction, which we instead define via $\varphi \otimes \psi := \sim(\sim\varphi \wedge \sim\psi)$. We also use the abbreviation $\mathsf{E}\beta := \sim\neg\beta$ ("at least one assignment in the team satisfies β").

▶ **Lemma 4.1** ($\star$). *Every τ-$\mathsf{FO}^n_k(\sim)$-formula φ is equivalent to a formula of the form*

$$\psi := \bigotimes_{i=1}^{n} \left(\alpha_i \wedge \bigwedge_{j=1}^{m_i} \mathsf{E}\beta_{i,j} \right)$$

such that $\{\alpha_1, \dots, \alpha_n, \beta_{1,1}, \dots, \beta_{n,m_n}\} \subseteq \tau$-$\mathsf{FO}^n_k$ and $|\psi| \leq \exp_{\mathcal{O}(|\varphi|)}(1)$.

Proof. The proof is by induction on φ, and consists of repeatedly applying the following distributive laws. Here, $\varphi \equiv \psi$ means that φ and ψ are logically equivalent. See the appendix for details.

$$\alpha \wedge \bigwedge_{i=1}^{n} \mathsf{E}\beta_i \equiv \bigvee_{i=1}^{n} (\alpha \wedge \mathsf{E}\beta_i) \qquad \bigvee_{i=1}^{n} (\alpha_i \wedge \mathsf{E}\beta_i) \equiv \Big(\bigvee_{i=1}^{n} \alpha_i\Big) \wedge \bigwedge_{i=1}^{n} \mathsf{E}(\alpha_i \wedge \beta_i)$$

$$(\vartheta_1 \otimes \vartheta_2) \vee \vartheta_3 \equiv (\vartheta_1 \vee \vartheta_3) \otimes (\vartheta_2 \vee \vartheta_3) \qquad \vartheta_1 \vee (\vartheta_2 \otimes \vartheta_3) \equiv (\vartheta_1 \vee \vartheta_2) \otimes (\vartheta_1 \vee \vartheta_3)$$

$$(\vartheta_1 \otimes \vartheta_2) \wedge \vartheta_3 \equiv (\vartheta_1 \wedge \vartheta_3) \otimes (\vartheta_2 \wedge \vartheta_3) \qquad \vartheta_1 \wedge (\vartheta_2 \otimes \vartheta_3) \equiv (\vartheta_1 \wedge \vartheta_2) \otimes (\vartheta_1 \wedge \vartheta_3)$$

$$\exists v\,(\vartheta_1 \otimes \vartheta_2) \equiv (\exists v\,\vartheta_1) \otimes (\exists x\,\vartheta_2) \qquad \exists v\,(\vartheta_1 \vee \vartheta_2) \equiv (\exists v\,\vartheta_1) \vee (\exists v\,\vartheta_2)$$

$$\exists v\,(\alpha \wedge \mathsf{E}\beta) \equiv (\exists v\,\alpha) \wedge \mathsf{E}\,\exists v\,(\alpha \wedge \beta) \qquad \forall v\,(\vartheta_1 \wedge \vartheta_2) \equiv (\forall v\,\vartheta_1) \wedge (\forall v\,\vartheta_2)$$

$$\forall v \sim\vartheta \equiv \sim\forall v\,\vartheta$$

◀

▶ **Theorem 4.2.** *If τ is a relational vocabulary, then every satisfiable $\varphi \in \tau$-$\mathsf{FO}^2(\sim)$ has a model of size $\exp_{\mathcal{O}(|\varphi|)}(1)$.*

Proof. Let $\varphi \in \tau$-$\mathsf{FO}^2(\sim)$ be satisfiable. φ is equivalent to a disjunction of size $\exp_{\mathcal{O}(|\varphi|)}(1)$ as stated in Lemma 4.1. Clearly, this disjunction must have at least one satisfiable disjunct, which is of the form

$$\psi = \alpha \wedge \bigwedge_{i=1}^{m} \mathsf{E}\beta_i,$$

for $\{\alpha, \beta_1, \dots, \beta_m\} \subseteq \tau$-$\mathsf{FO}^2$ and w.l.o.g. $\mathsf{Var}(\psi) \subseteq \{x, y\}$. Let $(\mathcal{A}, T)$ be a model of ψ. For every i, as ψ implies $\mathsf{E}(\alpha \wedge \beta_i)$, there exists $s \in T$ such that $(\mathcal{A}, s) \vDash \alpha \wedge \beta_i$. But then $\mathcal{A}$ also satisfies – in Tarski semantics – the classical FO^2-sentence

$$\gamma := \bigwedge_{i=1}^{m} \exists x\,\exists y\,\alpha \wedge \beta_i,$$

as $s(x)$ and $s(y)$ are witnesses for $\exists x$ and $\exists y$. However, by the exponential model property of FO^2 [19], there exists a model $\mathcal{B}$ of size $2^{\mathcal{O}(|\gamma|)}$ for γ. As for every i there is $\hat{s}_i \colon \{x, y\} \to \mathcal{B}$ such that $(\mathcal{B}, \hat{s}_i) \vDash \alpha \wedge \beta_i$, we conclude $(\mathcal{B}, \{\hat{s}_1, \dots, \hat{s}_m\}) \vDash \psi$ by definition of team semantics. Clearly, this shows that ψ and hence φ has a model of size $\exp_{\mathcal{O}(|\varphi|)}(1)$. ◀

▶ **Corollary 4.3.** *If τ is a relational vocabulary, then* $\mathsf{SAT}(\tau\text{-}\mathsf{FO}^2(\sim))$ *and* $\mathsf{VAL}(\tau\text{-}\mathsf{FO}^2(\sim))$ *are in* TOWER(poly).

Proof. By Corollary 3.6, model checking for $\mathsf{FO}^2(\sim)$ is possible in alternating polynomial time, and hence in deterministic exponential time. The following is a TOWER(poly)-algorithm for $\mathsf{SAT}(\tau\text{-}\mathsf{FO}^2(\sim))$. Given a formula φ, iterate over all interpretations $(\mathcal{A}, T)$ of size $\exp_{\mathcal{O}(|\varphi|)}(1)$ and accept if $(\mathcal{A}, T) \models \varphi$. The algorithm for $\mathsf{VAL}(\tau\text{-}\mathsf{FO}^2(\sim))$ is similar. ◀

5 From MTL to FO²(∼): A team-semantical standard translation

In order to prove the lower bounds for $\mathsf{FO}^2(\sim)$ and $\mathsf{FO}^n_k(\sim)$, we reduce from the corresponding satisfiability, validity and model checking problems of so-called *modal team logic* MTL. This logic was introduced by Müller [32], and extends classical modal logic ML by $\sim$ in the same fashion as $\mathsf{FO}(\sim)$ extends FO.

We fix a countably infinite set Φ of propositions. MTL is defined as follows, where φ denotes an MTL-formula, α an ML-formula, and p a proposition.

$$\varphi ::= \sim\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid \Diamond\varphi \mid \alpha \qquad\qquad \alpha ::= \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \Box\alpha \mid \Diamond\alpha \mid p$$

The modal depth $\mathsf{md}(\varphi)$ of φ is defined recursively, i.e., $\mathsf{md}(p) := 0$, $\mathsf{md}(\sim\varphi) := \mathsf{md}(\neg\varphi) := \mathsf{md}(\varphi)$, $\mathsf{md}(\varphi \wedge \psi) := \mathsf{md}(\varphi \vee \psi) := \max\{\mathsf{md}(\varphi), \mathsf{md}(\psi)\}$, and $\mathsf{md}(\Box\varphi) := \mathsf{md}(\Diamond\varphi) := \mathsf{md}(\varphi)+1$. MTL_k is the fragment of MTL with modal depth at most k. The set of propositional variables occurring in $\varphi \in \mathsf{MTL}$ is written $\mathsf{Prop}(\varphi)$.

Let $X \subseteq \Phi$ be finite. Then, a Kripke structure (over X) is a tuple $\mathcal{K} = (W, R, V)$, where W is a set of *worlds* or *points*, (W, R) is a directed graph, and $V\colon X \to \mathfrak{P}(W)$. If $w \in W$, then $(\mathcal{K}, w)$ is called *pointed Kripke structure*.

ML is evaluated on pointed Kripke structures in the classical Kripke semantics, whereas MTL is evaluated on pairs $(\mathcal{K}, T)$, where $\mathcal{K}$ is a Kripke structure and – analogously to the first-order case – $T \subseteq W$ is called *team* (in $\mathcal{K}$). The team $RT := \{v \in W \mid \exists w \in T\colon Rwv\}$ is the *image* of T, and we write Rw instead of $R\{w\}$ for brevity. A successor team of T is a team S such that every $w \in T$ has at least one successor in S, and every $v \in S$ has at least one predecessor in T. The semantics of MTL is now defined as follows:

$$\begin{aligned}
(\mathcal{K}, T) \models \varphi &\;\Leftrightarrow\; \forall w \in T\colon (\mathcal{K}, w) \models \varphi \text{ (in Kripke semantics) for } \varphi \in \mathsf{ML},\\
(\mathcal{K}, T) \models \sim\psi &\;\Leftrightarrow\; (\mathcal{K}, T) \not\models \psi,\\
(\mathcal{K}, T) \models \psi \wedge \theta &\;\Leftrightarrow\; (\mathcal{K}, T) \models \psi \text{ and } (\mathcal{K}, T) \models \theta,\\
(\mathcal{K}, T) \models \psi \vee \theta &\;\Leftrightarrow\; \exists S, U \subseteq T \text{ such that } T = S \cup U,\ (\mathcal{K}, S) \models \psi, \text{ and } (\mathcal{K}, U) \models \theta,\\
(\mathcal{K}, T) \models \Diamond\psi &\;\Leftrightarrow\; \exists S \text{ such that } S \text{ is a successor team of } T \text{ and } (\mathcal{K}, S) \models \psi,\\
(\mathcal{K}, T) \models \Box\psi &\;\Leftrightarrow\; (\mathcal{K}, RT) \models \psi.
\end{aligned}$$

A formula $\varphi \in \mathsf{MTL}$ is satisfiable if $(\mathcal{K}, T) \models \varphi$ for some Kripke structure $\mathcal{K}$ over $X \supseteq \mathsf{Prop}(\varphi)$ and team T in $\mathcal{K}$. Likewise, φ is valid if it is true in every such pair.

The modality-free fragment MTL_0 syntactically coincides with *propositional team logic* PTL [20, 21, 40]. The usual interpretations of the latter, i.e., via sets of Boolean assignments, can easily be simulated by teams in Kripke structures. For this reason, we identify PTL and MTL_0 in this paper.

The following lower bounds due to Hannula et al. [21] are logspace reductions.

▶ **Theorem 5.1** ([21])**.** MC(PTL) *is* PSPACE*-complete.*

▶ **Theorem 5.2** ([21]). $\mathsf{SAT}(\mathsf{PTL})$ *and* $\mathsf{VAL}(\mathsf{PTL})$ *are* ATIME-ALT(exp, poly)*-complete.*

For each increment in modal depth, the complexity of the satisfiability problem increases by an exponential, reaching the non-elementary class TOWER(poly) in the unbounded case:

▶ **Theorem 5.3** ([28]). $\mathsf{SAT}(\mathsf{MTL})$ *and* $\mathsf{VAL}(\mathsf{MTL})$ *are* TOWER(poly)*-complete.* $\mathsf{SAT}(\mathsf{MTL}_k)$ *and* $\mathsf{VAL}(\mathsf{MTL}_k)$ *are* ATIME-ALT($\exp_{k+1}$, poly)*-complete for every* $k < \omega$.

Next, let us demonstrate how MTL can be embedded into $\mathsf{FO}^2(\sim)$. More precisely, we present an extension of the well-known standard translation that embeds modal logic ML into FO^2. The underlying relational vocabulary usually is $\tau_{\mathrm{st}} = (R, P_1, P_2, \ldots)$, where $\mathsf{arity}(R) = 2$ and $\mathsf{arity}(P_i) = 1$ for all i. The translation of an ML-formula α is denoted by $\mathsf{st}_x(\alpha)$ resp. $\mathsf{st}_y(\alpha)$, and is defined by mutual recursion:

$$\begin{aligned}
\mathsf{st}_x(p_i) &:= P_i x \ \text{ for } p_i \in \mathcal{PS} & \mathsf{st}_x(\neg\alpha) &:= \neg\mathsf{st}_x(\alpha)\\
\mathsf{st}_x(\Box\alpha) &:= \forall y\, Rxy \to \mathsf{st}_y(\alpha) & \mathsf{st}_x(\alpha \wedge \beta) &:= \mathsf{st}_x(\alpha) \wedge \mathsf{st}_x(\beta)\\
\mathsf{st}_x(\Diamond\alpha) &:= \exists y\, Rxy \wedge \mathsf{st}_y(\alpha) & \mathsf{st}_x(\alpha \vee \beta) &:= \mathsf{st}_x(\alpha) \vee \mathsf{st}_x(\beta),
\end{aligned}$$

with $\mathsf{st}_y(\alpha)$ defined symmetrically via $\mathsf{st}_x(\alpha)$. The corresponding *first-order interpretation* of a Kripke structure $\mathcal{K} = (W, R', V)$ is the τ_{st}-structure $\mathcal{A}(\mathcal{K})$ defined by $\mathsf{dom}\,\mathcal{A}(\mathcal{K}) = W$, $R^{\mathcal{A}(\mathcal{K})} = R'$ and $P_i^{\mathcal{A}(\mathcal{K})} = V(p_i)$. For a world w, let $w^x \colon \{x\} \to W$ be defined by $w^x(x) = w$.

▶ **Theorem 5.4** (see, e.g., Blackburn et al. [5]). *Let* $(\mathcal{K}, w)$ *be a pointed Kripke structure and* $\alpha \in \mathsf{ML}$. *Then* $(\mathcal{K}, w) \vDash \alpha$ *if and only if* $(\mathcal{A}(\mathcal{K}), w^x) \vDash \mathsf{st}_x(\alpha)$.

Let us now turn to team semantics. On the model side, the first-order interpretation of a team T in a Kripke structure is straightforwardly $T^x := \{\, w^x \mid w \in T \,\}$. For the syntax, we require the additional operator $\hookrightarrow$. It was introduced by Galliani [12] and Kontinen and Nurmi [25] in the first-order setting, but was also adapted to the modal setting [28]. For $\alpha \in \mathsf{ML}$ and $\varphi \in \mathsf{MTL}$, define $\alpha \hookrightarrow \varphi := \neg\alpha \vee (\alpha \wedge \varphi)$. If $(\mathcal{K}, T)$ is a Kripke structure with team, let $T_\alpha := \{\, w \in T \mid (\mathcal{K}, w) \vDash \alpha \,\}$.

▶ **Proposition 5.5.** $(\mathcal{A}, T) \vDash \alpha \hookrightarrow \varphi$ *if and only if* $(\mathcal{A}, T_\alpha) \vDash \varphi$.

Proof. Straightforward. See also Galliani [12, Lemma 16]. ◀

We extend the above translation by an $\sim$-case, and in the $\Box$-case replace $\to$ by $\hookrightarrow$.[2] The standard translation for MTL, denoted by st_x^* resp. st_y^*, then becomes:

$$\begin{aligned}
\mathsf{st}_x^*(\alpha) &:= \mathsf{st}_x(\alpha) \ \text{ for } \alpha \in \mathsf{ML} & \mathsf{st}_x^*(\sim\varphi) &:= \sim\mathsf{st}_x^*(\varphi)\\
\mathsf{st}_x^*(\Box\varphi) &:= \forall y\, Rxy \hookrightarrow \mathsf{st}_y^*(\varphi) & \mathsf{st}_x^*(\varphi \wedge \psi) &:= \mathsf{st}_x^*(\varphi) \wedge \mathsf{st}_x^*(\psi)\\
\mathsf{st}_x^*(\Diamond\varphi) &:= \exists y\, Rxy \wedge \mathsf{st}_y^*(\varphi) & \mathsf{st}_x^*(\varphi \vee \psi) &:= \mathsf{st}_x^*(\varphi) \vee \mathsf{st}_x^*(\psi),
\end{aligned}$$

with $\mathsf{st}_y^*(\varphi)$ again defined symmetrically.

▶ **Theorem 5.6.** *For every Kripke structure* $\mathcal{K}$*, team* T *in* $\mathcal{K}$ *and* $\varphi \in \mathsf{MTL}$ *it holds* $(\mathcal{K}, T) \vDash \varphi$ *if and only if* $(\mathcal{A}(\mathcal{K}), T^x) \vDash \mathsf{st}_x^*(\varphi)$.

Proof. Proof by induction on φ. We omit $\mathcal{K}$ and $\mathcal{A}(\mathcal{K})$ and simply write, e.g., $T \vDash \varphi$.

[2] It is not hard to show that the "classical" translation of $\Box\varphi$ to $\forall y\, Rxy \to \mathsf{st}_y^*(\varphi) = \forall y\, (\neg Rxy \vee \mathsf{st}_y^*(\varphi))$ is unsound under team semantics.

- $\varphi \in \mathsf{ML}$: We have $T \vDash \varphi$ iff $\forall w \in T\colon w \vDash \varphi$ by definition of the semantics of MTL, which by Theorem 5.4 is equivalent to $\forall w^x \in T^x\colon w^x \vDash \mathsf{st}_x(\varphi)$. However, as $\mathsf{st}_x(\varphi) \in \mathsf{FO}$, the latter is equivalent to $T^x \vDash \mathsf{st}_x(\varphi)$ by the semantics of $\mathsf{FO}(\sim)$, and hence $T^x \vDash \mathsf{st}^*_x(\varphi)$.
- $\varphi = \psi \wedge \vartheta$ and $\varphi = {\sim}\psi$ are clear.
- $\varphi = \psi \vee \theta$: Suppose $T \vDash \psi \vee \theta$. Then $T = S \cup U$ such that $S \vDash \psi$ and $U \vDash \theta$. By induction hypothesis, $S^x \vDash \mathsf{st}^*_x(\psi)$ and $U^x \vDash \mathsf{st}^*_x(\theta)$. As $S \cup U = T$, clearly $S^x \cup U^x = T^x$. As a consequence, $T^x \vDash \mathsf{st}^*_x(\psi) \vee \mathsf{st}^*_x(\theta) = \mathsf{st}^*_x(\psi \vee \theta)$.

  For the other direction, suppose $T^x \vDash \mathsf{st}^*_x(\psi \vee \theta) = \mathsf{st}^*_x(\psi) \vee \mathsf{st}^*_x(\theta)$ by the means of some subteams $S' \cup U' = T^x$ such that $S' \vDash \mathsf{st}^*_x(\psi)$ and $U' \vDash \mathsf{st}^*_x(\theta)$. As T^x has domain $\{x\}$, there are unique $S, U \subseteq T$ such that $S' = S^x$ and $U' = U^x$. By induction hypothesis, $S \vDash \psi$ and $U \vDash \theta$. In order to prove $T \vDash \psi \vee \theta$, it remains to show $T \subseteq S \cup U$. For this purpose, let $w \in T$. As then $w^x \in T^x$, as least one of $w^x \in S'$ or $w^x \in U'$ holds. But then $w \in S$ or $w \in U$.
- $\varphi = \Box\psi$: We define subteams S and U of the duplicating team $(T^x)^y_W$ as follows: S contains all "outgoing edges": $S := \{s \in (T^x)^y_W \mid s(y) \in Rs(x)\}$. On the other hand, U contains all "non-edges": $U := \{s \in (T^x)^y_W \mid s(y) \notin Rs(x)\}$. Then clearly $(T^x)^y_W = S \cup U$, $S \vDash Rxy$ and $U \vDash \neg Rxy$. Moreover, the above division of $(T^x)^y_W$ into S and U is the *only* possible splitting of $(T^x)^y_W$ such that $S \vDash Rxy$ and $U \vDash \neg Rxy$.
  By induction hypothesis, clearly $T \vDash \Box\psi \Leftrightarrow (RT)^y \vDash \mathsf{st}^*_y(\psi)$. Moreover, by the above argument, $T^x \vDash \mathsf{st}^*_x(\Box\psi) \Leftrightarrow S \vDash \mathsf{st}^*_y(\psi)$. Consequently, it suffices to show that $(RT)^y$ and S agree on $\mathsf{st}^*_y(\psi)$. This follows from Proposition 2.5, since

$$\begin{aligned}(RT)^y &= \{s\colon \{y\} \to W \mid \exists w \in T : s(y) \in Rw\}\\ &= \{s{\restriction}y \mid s \in (T^x)^y_W \text{ and } s(y) \in Rs(x)\} = S{\restriction}y.\end{aligned}$$

- $\varphi = \Diamond\psi$: Suppose $T \vDash \Diamond\psi$, i.e., $S \vDash \psi$ for some successor team S of T. By induction hypothesis, $S^y \vDash \mathsf{st}^*_y(\psi)$. In order to prove $T^x \vDash \exists y\, Rxy \wedge \mathsf{st}^*_y(\psi)$, we define a supplementing function $f : T^x \to \mathfrak{P}(W) \setminus \{\emptyset\}$ such that $(T^x)^y_f \vDash Rxy \wedge \mathsf{st}^*_y(\psi)$.
  Let $f(w^x) := Rw \cap S$. Then $f(w^x)$ is non-empty for each w, as S is a successor team. Moreover, $(T^x)^y_f \vDash Rxy$. It remains to show that $(T^x)^y_f \vDash \mathsf{st}^*_y(\psi)$ follows from $S^y \vDash \mathsf{st}^*_y(\psi)$. Here, we combine Proposition A.1 and 2.5, since

$$y\langle S^y\rangle = S = \bigcup_{w \in T} Rw \cap S = \bigcup_{w^x \in T^x} f(w^x) = \{s(y) \mid s \in (T^x)^y_f\} = y\langle (T^x)^y_f\rangle.$$

  For the other direction, suppose $T^x \vDash \exists y\, Rxy \wedge \mathsf{st}^*_y(\psi)$ by the means of a supplementing function $f : T^x \to \mathfrak{P}(W) \setminus \{\emptyset\}$ such that $(T^x)^y_f \vDash Rxy \wedge \mathsf{st}^*_y(\psi)$.
  We define $S := \bigcup_{w \in T} f(w^x)$, and first prove that it is a successor team of T, i.e., that every $v \in S$ has a predecessor in T and that every $w \in T$ has a successor in S.
  Let $v \in S$. Then there exists $w \in T$ such that $v \in f(w^x)$. As a consequence, the assignment s given by $s(x) = w$ and $s(y) = v$ is in $(T^x)^y_f$, and hence satisfies Rxy. In other words, v has a predecessor in T. Conversely, if $w \in T$, then $f(w^x)$ is non-empty, i.e., contains an element v. As before, v is a successor of w. Since $v \in f(w^x)$, $v \in S$, so w has a successor in S. By a similar argument as above, $y\langle (T^x)^y_f\rangle = S = y\langle S^y\rangle$, hence $S^y \vDash \mathsf{st}^*_y(\psi)$, and consequently $S \vDash \psi$ by induction hypothesis. ◀

6 Lower bounds

As a first application of the extended standard translation from the previous section, we prove several complexity theoretic lower bounds.

▶ **Lemma 6.1.** $\mathsf{MC}(\tau\text{-}\mathsf{FO}^1_0(\sim))$ *is* PSPACE*-hard if* τ *contains infinitely many predicates.*

Proof. We reduce from $\mathsf{MC}(\mathsf{PTL})$, which is PSPACE-hard by Theorem 5.1. The reduction maps $(\mathcal{K}, T, \varphi)$ to $(\mathcal{A}(\mathcal{K}), T^x, \mathsf{st}^*_x(\varphi))$. W.l.o.g. τ contains unary predicates $P_0, P_1, \ldots$; otherwise they are easily simulated by predicates of higher arity. It is now easy to see that $\mathsf{st}^*_x(\varphi)$ is quantifier-free and contains only the variable x. Moreover, by Theorem 5.6, $(\mathcal{K}, T) \models \varphi$ if and only if $(\mathcal{A}(\mathcal{K}), T^x) \models \mathsf{st}^*_x(\varphi)$. ◀

▶ **Lemma 6.2.** $\mathsf{MC}(\tau\text{-}\mathsf{FO}^\omega_\omega(\sim))$ *is* ATIME-ALT(exp, poly)*-hard for all vocabularies* τ*, even on sentences and for a fixed* τ*-structure* $\mathcal{A}$ *with domain* $\{0,1\}$ *and a fixed team* $\{\emptyset\}$*.*

Proof. Here, we reduce from $\mathsf{SAT}(\mathsf{PTL})$, which is ATIME-ALT(exp, poly)-hard by Theorem 5.2. Given $\varphi \in \mathsf{PTL}$, suppose $\mathsf{Prop}(\varphi) = \{p_1, \ldots, p_n\}$. The idea is that a team of worlds (and their Boolean assignments to $p_1, \ldots, p_n$), are simulated by a team of first-order assignments $s\colon X \to B$, where $X = \{z, x_1, \ldots, x_n\}$ and $B := \{0, 1\}$. Here, the variable z acts as the constant 1, while x_i simulates p_i. For each $b \in B$, define the team $V_b := (\{\emptyset\}^z_{\{b\}})^{x_1}_B \cdots ^{x_n}_B$. In other words, V_b is the n-fold supplemented team of $\{\emptyset\}^z_{\{b\}} = \{\{z \mapsto b\}\}$.

In the remaining proof, we distinguish two cases based on τ. By definition of a vocabulary, either $= \in \tau$, or τ contains a predicate. First, we consider the case $= \in \tau$. We reduce via the mapping $\varphi \mapsto (\mathcal{A}, \{\emptyset\}, \psi)$, where $\mathcal{A}$ is a fixed τ-structure with $\mathsf{dom}\,\mathcal{A} = B$, $\psi := \exists z \forall x_1 \cdots \forall x_n \top \vee \varphi^*$, and φ^* is obtained from φ by replacing each p_i by $x_i = z$. We prove that the reduction is correct, and begin with the following equivalence:

$$\exists U \subseteq V_1 : (\mathcal{A}, U) \models \varphi^* \quad \text{iff} \quad (\mathcal{A}, V_1) \models \top \vee \varphi^* \quad \text{iff} \quad (\mathcal{A}, \{\emptyset\}) \models \psi. \tag{1}$$

Here, "⇒" follows from the semantics of $\vee$ and the definition of ψ. For "⇐", suppose $(\mathcal{A}, \{\emptyset\}) \models \psi$. Then, again by definition of ψ, we have $(\mathcal{A}, U) \models \varphi^*$ for some $U \subseteq V_0 \cup V_1$. In particular, the variable z can take the values 0, 1 or both in U. However, for all $s \in U \cap V_0$, we can simply flip the ones and zeroes of s. This leaves the truth of any atomic formula $x_i = z$ unchanged, and by induction preserves the semantics of φ^*.

Next, we proceed with the correctness of the reduction. Assume that φ is satisfiable, i.e., has a model $(\mathcal{K}, T)$. For each world $w \in T$, define $s_w\colon X \to B$ by $s_w(z) = 1$ and $s_w(x_i) = 1 \Leftrightarrow (\mathcal{K}, w) \models p_i$. Then $(\mathcal{K}, w) \models p_i$ if and only if $(\mathcal{A}, s_w) \models x_i = z$. By induction on the syntax of φ, we obtain $(\mathcal{A}, U) \models \varphi^*$, where $U := \{s_w \mid w \in T\}$. As $U \subseteq V_1$, the equivalence (1) yields $(\mathcal{A}, \{\emptyset\}) \models \psi$. The other direction is similar.

Next, consider the case where $= \notin \tau$; then τ contains a predicate P. We define $\mathcal{A}$ as above, but let $P^{\mathcal{A}} := \{(1, \ldots, 1)\}$. Furthermore, $\psi := \forall x_1 \cdots \forall x_n \top \vee \varphi^*$, and φ^* is now as φ, with p_i replaced by $P(x_i, \ldots, x_i)$. The remaining proof is similar to the previous one. ◀

Clearly, the standard translation of satisfiable formulas is itself satisfiable. A converse result holds as well. Loosely speaking, from a first-order structure (and team) for $\mathsf{st}^*_x(\varphi)$ we can reconstruct a Kripke model (and team) for φ.

▶ **Lemma 6.3.** *If* $\varphi \in \mathsf{MTL}$*, then* φ *is satisfiable if and only if* $\mathsf{st}^*_x(\varphi)$ *is satisfiable.*

Proof. As Theorem 5.6 implies "⇒", we show "⇐". Suppose $(\mathcal{B}, S) \models \mathsf{st}^*_x(\varphi)$. Then $\mathcal{B}$ interprets the binary predicate R and unary predicates $P_1, P_2, \ldots$. By Proposition 2.5, w.l.o.g. S has domain $\{x\}$, i.e., $S = (x\langle S \rangle)^x$. Define now the Kripke structure $\mathcal{K} = (\mathsf{dom}\,\mathcal{B}, R^{\mathcal{B}}, V)$ such that $V(p_i) := P^{\mathcal{B}}_i$. Then clearly $\mathcal{A}(\mathcal{K}) = \mathcal{B}$. By Theorem 5.6, $(\mathcal{K}, x\langle S \rangle) \models \varphi$. ◀

Finally, with the above lower bounds, let us gather the completeness results for the satisfiability, validity and model checking problems.

▶ **Theorem 6.4.** *Let $\mathcal{D}$ be any p-uniform set of dependencies and τ any vocabulary.*

- $\mathsf{MC}(\tau\text{-}\mathsf{FO}^{\omega}_{\omega}(\sim, \mathcal{D}))$ *is* ATIME-ALT(exp, poly)*-complete, with hardness already on sentences and for a fixed τ-structure $\mathcal{A}$ with domain $\{0,1\}$ and a fixed team $\{\emptyset\}$.*
- *If τ contains infinitely many relations and at least one of $k \geq 0, n \geq 1$ is finite, then* $\mathsf{MC}(\tau\text{-}\mathsf{FO}^{n}_{k}(\sim, \mathcal{D}))$ *is* PSPACE*-complete.*

Proof. The upper bounds are due to Corollary 3.3 and 3.6, since alternating polynomial time coincides with PSPACE [7]. The lower bounds are due to Lemma 6.1 and 6.2. ◀

▶ **Corollary 6.5.** $\mathsf{MC}(\tau\text{-}\mathsf{SO})$ *is* ATIME-ALT(exp, poly)*-complete for all vocabularies τ, with hardness already on sentences and with a fixed τ-interpretation $\mathcal{A}$ with* $\mathsf{dom}\,\mathcal{A} = \{0,1\}$.

Proof. The upper bound is by Proposition 2.6. The lower bound is by the previous theorem and reduction from $\mathsf{MC}(\tau\text{-}\mathsf{FO}(\sim))$. Let R be a nullary predicate variable. In the spirit of Corollary 3.3, we map $(\mathcal{A}, \{\emptyset\}, \varphi)$ to $(\mathcal{A}, \emptyset, \exists R\, \eta^{\emptyset}_{\varphi}(R) \wedge R)$, where φ w.l.o.g. is a sentence. ◀

The next theorem settles the complexity of the satisfiability and validity problem of $\mathsf{FO}^2(\sim)$, and provides lower bounds for $\mathsf{FO}^1_0(\sim)$ and $\mathsf{FO}^2_k(\sim)$.

▶ **Theorem 6.6.** *Let τ contain at least one binary predicate, infinitely many unary predicates, and no functions. Then the problems* $\mathsf{SAT}(\tau\text{-}\mathsf{FO}^{n}_{k}(\sim))$ *and* $\mathsf{VAL}(\tau\text{-}\mathsf{FO}^{n}_{k}(\sim))$ *are*

- TOWER(poly)*-complete for $n = 2$ and $k = \omega$,*
- ATIME-ALT($\exp_{k+1}$, poly)*-hard for $n = 2$ and $0 \leq k < \omega$,*
- ATIME-ALT(exp, poly)*-hard for $n = 1$ and $0 \leq k < \omega$.*

Proof. The upper bound for $\tau\text{-}\mathsf{FO}^2(\sim)$ is by Corollary 4.3. For the lower bounds, the mapping $\varphi \mapsto \mathsf{st}^*_x(\varphi)$ is a reduction from $\mathsf{SAT}(\mathsf{MTL})$ resp. $\mathsf{SAT}(\mathsf{MTL}_k)$ (see Theorem 5.3 and Lemma 6.3). Finally, the validity cases follow since the logic is closed under negation. ◀

Let us contrast the above decidable cases with the following negative result, where a single unary dependence atom is added to the logic (cf. p. 4).

▶ **Theorem 6.7.** *There is a vocabulary τ such that* $\mathsf{SAT}(\mathcal{L})$ *is Π^0_1-hard and* $\mathsf{VAL}(\mathcal{L})$ *is Σ^0_1-hard, where $\mathcal{L} = \tau\text{-}\mathsf{FO}^2_2(\sim, \{\mathsf{dep}_1\})$.*

Proof. Kontinen et al. [24] showed that $\mathsf{VAL}(\mathsf{D}^2_2)$ is Σ^0_1-hard, and their reduction in fact uses only unary and binary dependence atoms. Moreover, the binary dependence atom $=\!(x, y)$ can equivalently be rewritten as $\sim(\top \vee (=\!(x) \wedge \sim=\!(y)))$, where $\top$ is an arbitrary tautology. Intuitively, this formula stipulates that every subteam constant in x is also constant in y. This concludes the reduction to $\mathsf{VAL}(\tau\text{-}\mathsf{FO}^2_2(\sim, \{\mathsf{dep}_1\}))$. Again, the proof for the satisfiability problem is analogous. ◀

7 Conclusion

In this paper, we proved that the logic $\mathsf{FO}^2(\sim)$ is complete for the class TOWER(poly) and hence decidable. In particular, it has the finite model property, but exhibits non-elementary succinctness compared to classical FO^2, which enjoys an exponential model property [19].

For $\mathsf{FO}^n_k(\sim, \mathcal{D})$, where $n \geq 1$ and $k \geq 0$, we proved a dichotomy regarding its model checking complexity: It is ATIME-ALT(exp, poly)-complete if $n = k = \omega$, and otherwise PSPACE-complete. This only requires that $\mathcal{D}$ is a p-uniformly FO-definable set of generalized

dependency atoms (cf. Definition 3.2), which covers first-order team logic TL as well as independence [17] and inclusion logic [10] augmented with Boolean negation.

We conclude with some open questions:

- Can the translation from $\mathsf{FO}^n_k(\sim, \mathcal{D})$ to $\mathsf{SO}[p]$ be inverted, i.e., can we translate every $\mathsf{SO}[p]$-formula to $\mathsf{FO}^n_k(\sim, \mathcal{D})$ for suitable n and k? This would be an interesting generalization of the translation from SO to TL given by Kontinen and Nurmi [25].
- What is the exact complexity of $\mathsf{SAT}(\mathsf{FO}^2_k(\sim))$? In the modal setting, every satisfiable MTL_k-formula has a $(k+1)$-fold exponential model. It would be interesting to learn whether the same holds for $\mathsf{FO}^2_k(\sim)$. Due to Corollary 3.6, a positive answer would immediately yield a tight ATIME-ALT$(\exp_{k+1}, \mathrm{poly})$ upper bound.
- It is a well-known fact that the standard translation of an ML-formula is in the two-variable guarded fragment GF^2. It is conceivable to consider a similar fragment $\mathsf{GF}^2(\sim)$ for the standard translation of MTL. Studying the corresponding fragments $\mathsf{GF}^2_k(\sim)$ of bounded quantifier rank could also be a first step towards finding the complexity of $\mathsf{FO}^2_k(\sim)$.

References

1 Samson Abramsky, Juha Kontinen, Jouko Väänänen, and Heribert Vollmer, editors. *Dependence Logic, Theory and Applications*. Springer, 2016. doi:10.1007/978-3-319-31803-5.

2 Hajnal Andréka, István Németi, and Johan van Benthem. Modal languages and bounded fragments of predicate logic. *J. Philosophical Logic*, 27(3):217–274, 1998. doi:10.1023/A:1004275029985.

3 Vince Bárány and Mikołaj Bojańczyk. Finite satisfiability for guarded fixpoint logic. *Inf. Process. Lett.*, 112(10):371–375, 2012. doi:10.1016/j.ipl.2012.02.005.

4 Leonard Berman. The complexitiy of logical theories. *Theor. Comput. Sci.*, 11:71–77, 1980. doi:10.1016/0304-3975(80)90037-7.

5 Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal logic*. Cambridge University Press, New York, NY, USA, 2001.

6 Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer, 1997.

7 Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.

8 Arnaud Durand, Juha Kontinen, and Heribert Vollmer. Expressivity and Complexity of Dependence Logic. In Samson Abramsky, Juha Kontinen, Jouko Väänänen, and Heribert Vollmer, editors, *Dependence Logic*, pages 5–32. Springer International Publishing, 2016. doi:10.1007/978-3-319-31803-5_2.

9 Johannes Ebbing, Lauri Hella, Arne Meier, Julian-Steffen Müller, Jonni Virtema, and Heribert Vollmer. Extended modal dependence logic. In *Logic, Language, Information, and Computation - 20th International Workshop, WoLLIC 2013.*, pages 126–137, 2013. doi:10.1007/978-3-642-39992-3_13.

10 Pietro Galliani. Inclusion and exclusion dependencies in team semantics - on some logics of imperfect information. *Ann. Pure Appl. Logic*, 163(1):68–84, 2012. doi:10.1016/j.apal.2011.08.005.

11 Pietro Galliani. General Models and Entailment Semantics for Independence Logic. *Notre Dame Journal of Formal Logic*, 54(2):253–275, 2013. doi:10.1215/00294527-1960506.

12 Pietro Galliani. Upwards closed dependencies in team semantics. *Inf. Comput.*, 245:124–135, 2015. doi:10.1016/j.ic.2015.06.008.

13 Pietro Galliani. On strongly first-order dependencies. In *Dependence Logic, Theory and Applications*, pages 53–71. Springer, 2016. doi:10.1007/978-3-319-31803-5_4.

14 Erich Grädel. Model-checking games for logics of imperfect information. *Theor. Comput. Sci.*, 493:2–14, 2013. doi:10.1016/j.tcs.2012.10.033.

15 Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, and Scott Weinstein. *Finite Model Theory and Its Applications*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2007. doi:10.1007/3-540-68804-8.

16 Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, 1997*, pages 306–317, 1997. doi:10.1109/LICS.1997.614957.

17 Erich Grädel and Jouko Väänänen. Dependence and independence. *Studia Logica*, 101(2):399–410, 2013. doi:10.1007/s11225-013-9479-2.

18 Erich Grädel and Igor Walukiewicz. Guarded fixed point logic. In *14th Annual IEEE Symposium on Logic in Computer Science*, pages 45–54, 1999. doi:10.1109/LICS.1999.782585.

19 Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997. doi:10.2307/421196.

20 Miika Hannula, Juha Kontinen, Martin Lück, and Jonni Virtema. On quantified propositional logics and the exponential time hierarchy. In *Proceedings of the Seventh International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2016.*, pages 198–212, 2016. doi:10.4204/EPTCS.226.14.

21 Miika Hannula, Juha Kontinen, Jonni Virtema, and Heribert Vollmer. Complexity of Propositional Logics in Team Semantic. *ACM Transactions on Computational Logic*, 19(1):1–14, jan 2018. doi:10.1145/3157054.

22 Wilfrid Hodges. Compositional semantics for a language of imperfect information. *Logic Journal of IGPL*, 5(4):539–563, 1997. doi:10.1093/jigpal/5.4.539.

23 Juha Kontinen, Antti Kuusisto, Peter Lohmann, and Jonni Virtema. Complexity of two-variable dependence logic and IF-logic. *Information and Computation*, 239:237–253, 2014. doi:10.1016/j.ic.2014.08.004.

24 Juha Kontinen, Antti Kuusisto, and Jonni Virtema. Decidability of Predicate Logics with Team Semantics. In *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 60:1–60:14, 2016. doi:10.4230/LIPIcs.MFCS.2016.60.

25 Juha Kontinen and Ville Nurmi. Team logic and second-order logic. *Fundam. Inform.*, 106(2-4):259–272, 2011. doi:10.3233/FI-2011-386.

26 Antti Kuusisto. A Double Team Semantics for Generalized Quantifiers. *Journal of Logic, Language and Information*, 24(2):149–191, 2015. doi:10.1007/s10849-015-9217-4.

27 Leopold Löwenheim. Über möglichkeiten im relativkalkül. *Mathematische Annalen*, 76:447–470, 1915. URL: http://eudml.org/doc/158703.

28 Martin Lück. Canonical Models and the Complexity of Modal Team Logic. Computer Science Logic (CSL) 2018. To appear.

29 Martin Lück. Axiomatizations of team logics. *Annals of Pure and Applied Logic*, 169(9):928–969, 2018. doi:10.1016/j.apal.2018.04.010.

30 Martin Lück. On the Complexity of Team Logic and its Two-Variable Fragment. *CoRR*, abs/1804.04968, 2018. URL: https://arxiv.org/abs/1804.04968.

31 Michael Mortimer. On languages with two variables. *Math. Log. Q.*, 21(1):135–140, 1975. doi:10.1002/malq.19750210118.

32 Julian-Steffen Müller. *Satisfiability and model checking in team based logics*. PhD thesis, University of Hanover, 2014. URL: http://d-nb.info/1054741921.

33 Ian Pratt-Hartmann. The two-variable fragment with counting revisited. In *Logic, Language, Information and Computation, 17th International Workshop, WoLLIC 2010.*, pages 42–54, 2010. doi:10.1007/978-3-642-13824-9_4.

34 Frank P. Ramsey. *On a Problem of Formal Logic*, pages 1–24. Birkhäuser Boston, Boston, MA, 1987. doi:10.1007/978-0-8176-4842-8_1.

35 Dana Scott. A decision method for validity of sentences in two variables. *Journal of Symbolic Logic*, 27(4):477, 1962.

36 Thomas Sturm, Marco Voigt, and Christoph Weidenbach. Deciding first-order satisfiability when universal and existential variables are separated. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2016*, pages 86–95, 2016. doi:10.1145/2933575.2934532.

37 Marco Voigt. A fine-grained hierarchy of hard problems in the separated fragment. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005094.

38 Jouko Väänänen. *Dependence logic: A New Approach to Independence Friendly Logic*. Number 70 in London Mathematical Society student texts. Cambridge University Press, Cambridge ; New York, 2007.

39 Fan Yang. *On extensions and variants of dependence logic*. PhD thesis, University of Helsinki, 2014. URL: http://www.math.helsinki.fi/logic/people/fan.yang/dissertation_fyang.pdf.

40 Fan Yang and Jouko Väänänen. Propositional team logics. *Ann. Pure Appl. Logic*, 168(7):1406–1441, 2017. doi:10.1016/j.apal.2017.01.007.

Algorithm 1: Decision procedure for MC(SO).

Algorithm: $\mathsf{check}(\alpha, \mathcal{A}, \mathcal{J})$ for $\alpha \in \tau\text{-SO}$ in negation normal form, a τ-structure $\mathcal{A}$, and a second-order interpretation $\mathcal{J}$ of $\mathsf{Fr}(\alpha)$.

if *α is an atomic formula or the negation of an atomic formula* **then**
  return `true` *if* $(\mathcal{A}, \mathcal{J}) \models \alpha$ *and* `false` *otherwise*;
else if $\alpha = \gamma_1 \vee \gamma_2$ **then** existentially choose $i \in \{1, 2\}$ and let $\alpha := \gamma_i$
else if $\alpha = \gamma_1 \wedge \gamma_2$ **then** universally choose $i \in \{1, 2\}$ and let $\alpha := \gamma_i$
else if $\alpha = \eth X \gamma$ *for* $\eth \in \{\exists, \forall\}$ *and* $X \in \mathsf{Var}(\alpha)$ **then**
  $\alpha := \gamma$
  if $X \in \mathsf{Fr}(\gamma)$ **then**
    if $\eth = \exists$ **then** switch to existential branching **else** switch to universal branching
    if *X is a first-order variable* **then**
      non-deterministically choose $a \in \mathcal{A}$ and let $\mathcal{J}(X) := a$
    else if *X is a function variable* **then**
      non-deterministically choose $F \colon \mathcal{A}^{\mathsf{arity}(X)} \to \mathcal{A}$ and let $\mathcal{J}(X) := F$
    else if *X is a relation variable* **then**
      non-deterministically choose $R \subseteq \mathcal{A}^{\mathsf{arity}(X)}$ and let $\mathcal{J}(X) := R$
return $\mathsf{check}(\alpha, \mathcal{A}, \mathcal{J}{\restriction}\mathsf{Fr}(\alpha))$

A Appendix

Proof of Proposition 2.6

▶ **Proposition 2.6.** MC(SO) *is decidable on input* $(\mathcal{A}, \mathcal{J}, \alpha)$ *in time* $2^{n^{\mathcal{O}(1)}}$ *and with* $|\alpha|$ *alternations.*

Proof. W.l.o.g., $\neg$ appears in α only in front of atomic formulas, and $\mathsf{dom}\,\mathcal{J} = \mathsf{Fr}(\alpha)$. Let $A := \mathsf{dom}\,\mathcal{A}$. We abbreviate

$$|\mathcal{J}| := \sum_{\substack{X \in \mathsf{dom}\,\mathcal{J} \\ X \text{ second-order}}} |\mathcal{J}(X)|,$$

i.e., the sum of the cardinalities of functions and relations in $\mathcal{J}$. Since for any second-order variable X it holds $|\mathcal{J}(X)| \leq |A|^{\mathsf{arity}(X)} \leq |A|^{|\alpha|}$, and furthermore $|\mathsf{dom}\,\mathcal{J}| = |\mathsf{Fr}(\alpha)| \leq |\alpha|$, the sum $|\mathcal{J}|$ is at most $|\alpha| \cdot |A|^{|\alpha|}$.

Now we run Algorithm 1. It performs at most $|\alpha|$ recursive calls, and clearly at most $|\alpha|$ alternations. Furthermore, the i-th recursive call is of the form $\mathsf{check}(\alpha_i, \mathcal{A}, \mathcal{J}_i)$ with $|\alpha_i| \leq |\alpha|$ and, by the same argument as before, $|\mathcal{J}_i| \leq |\alpha| \cdot |A|^{|\alpha|}$. For this reason, it is easy to see that the overall runtime is polynomial in $|\mathcal{J}|$ and $|A|^{|\alpha|}$, and consequently exponential in the input size. ◀

Proofs of Theorem 3.1 and 3.5

We require the next propositions in order to prove Theorem 3.1 and 3.5.

▶ **Proposition A.1.** *Let $\mathcal{A}$ be a structure, $\vec{t}$ a tuple of terms, and $X \supseteq \mathsf{Fr}(\vec{t})$. For $i \in \{1,2\}$, let T_i be a team in $\mathcal{A}$ with domain $X_i \supseteq X$. Then $T_1{\restriction}X = T_2{\restriction}X$ implies $\vec{t}\langle T_1\rangle = \vec{t}\langle T_2\rangle$. Furthermore, for any tuple $\vec{x} \subseteq X$ of variables, $\vec{x}\langle T_1\rangle = \vec{x}\langle T_2\rangle$ iff $T_1{\restriction}\vec{x} = T_2{\restriction}\vec{x}$.*

Proof. For the first part of the proposition, assume $T_1{\restriction}X = T_2{\restriction}X$. Exploiting symmetry, we only show that $\vec{t}\langle T_1\rangle \subseteq \vec{t}\langle T_2\rangle$. Hence, let $\vec{a} \in \vec{t}\langle T_1\rangle$ be arbitrary. Then $\vec{a} = \vec{t}\langle s\rangle$ for some $s \in T_1$. By assumption, there is $s' \in T_2$ such that $s{\restriction}X = s'{\restriction}X$. Since $\mathsf{Fr}(\vec{t}) \subseteq X$, clearly $\vec{t}\langle s\rangle = \vec{t}\langle s'\rangle$. Consequently, $\vec{a} \in \vec{t}\langle T_2\rangle$.

For the second part, suppose $\vec{x}\langle T_1\rangle = \vec{x}\langle T_2\rangle$ and let $s \in T_1{\restriction}\vec{x}$ be arbitrary. We show $s \in T_2{\restriction}\vec{x}$, which again suffices due to symmetry. Clearly, $s = s'{\restriction}\vec{x}$ for some $s' \in T_1$. Then $\vec{x}\langle s\rangle = \vec{x}\langle s'\rangle \in \vec{x}\langle T_1\rangle = \vec{x}\langle T_2\rangle$, and consequently, $\vec{x}\langle s\rangle \in \vec{x}\langle T_2\rangle$. But then $\vec{x}\langle s\rangle = \vec{x}\langle s''\rangle$ for some $s'' \in T_2$, which implies $s = s''{\restriction}\vec{x}$, and hence $s \in T_2{\restriction}\vec{x}$. ◀

▶ **Proposition A.2.** *Let $\mathcal{A}$ be a structure, $\vec{x}$ a tuple of variables, and $V := \{s\colon \vec{x} \to \mathcal{A}\}$. Then $\mathfrak{P}(V)$ is the set of all teams in $\mathcal{A}$ with domain $\vec{x}$, and the mapping $r\colon S \mapsto \vec{x}\langle S\rangle$ is an order isomorphism between $(\mathfrak{P}(V), \subseteq)$ and $(\mathfrak{P}((\mathsf{dom}\,\mathcal{A})^{|\vec{x}|}), \subseteq)$.*

Proof. Let $n := |\vec{x}|$. Clearly, every team with domain $\vec{x}$ is in $\mathfrak{P}(V)$. It is easy to show that r is surjective: Given $A \subseteq (\mathsf{dom}\,\mathcal{A})^n$, define the team $S := \{s \in V \mid \vec{x}\langle s\rangle \in A\}$. Then $r(S) = \vec{x}\langle S\rangle = \{\vec{x}\langle s\rangle \mid s \in V \text{ and } \vec{x}\langle s\rangle \in A\} = A$.

Moreover, r preserves $\subseteq$ in both directions: Suppose $S \subseteq S'$ and let $\vec{a} = (a_1, \dots, a_n) \in r(S)$ be arbitrary. We show $\vec{a} \in r(S')$, which proves $r(S) \subseteq r(S')$. Since $\vec{a} \in r(S) = \vec{x}\langle S\rangle$, there exists $s \in S$ such that $\vec{x}\langle s\rangle = \vec{a}$. By assumption, $s \in S'$. Consequently, $\vec{a} \in \vec{x}\langle S'\rangle = r(S')$.

Conversely, suppose $r(S) \subseteq r(S')$ and let $s \in S$ be arbitrary. As $\vec{x}\langle s\rangle \in r(S) \subseteq r(S') = \vec{x}\langle S'\rangle$, there exists an assignment $s' \in S'$ such that $\vec{x}\langle s\rangle = \vec{x}\langle s'\rangle$. However, as $\mathsf{dom}\,s = \mathsf{dom}\,s' = \vec{x}$, necessarily $s = s'$, i.e., $s \in S'$. As $S \subseteq S' \Leftrightarrow r(S) \subseteq r(S')$, and r is surjective, we conclude that r is also injective and hence an order isomorphism. ◀

As an alternative definition of supplementing functions, Galliani [11] coined the term *x-variations*, which are teams that "agree" on all variables but x:

▶ **Proposition A.3.** *Let T be a team with domain X and S a team with domain $X \cup \{x\}$ (with possibly $x \in X$), and let $X' := X \setminus \{x\}$. Then $S{\restriction}X' = T{\restriction}X'$ if and only if there is a supplementing function f such that $S = T^x_f$.*

Proof. Let $\mathcal{A}$ be the underlying structure.

"⇒": Suppose $S{\restriction}X' = T{\restriction}X'$. First, we show that for every $s' \in S$ there is $s \in T$ such that $s' = s^x_a$ for some a. By assumption, $s'{\restriction}X' = s{\restriction}X'$ for some $s \in T$. But then $s' = s^x_{s'(x)}$. We define the function $f(s) := \{\, a \in \mathcal{A} \mid s^x_a \in S \,\}$, and prove that it is a supplementing function of T. Here, it suffices to show that $f(s) \neq \emptyset$ for all $s \in T$, i.e., that for every $s \in T$ there exists $a \in \mathcal{A}$ such that $s^x_a \in S$. This follows again by $S{\restriction}X' = T{\restriction}X'$. Moreover, $T^x_f = \{s^x_a \mid s \in T, a \in f(s)\} = \{s^x_a \mid s \in T, s^x_a \in S\}$, which equals S by the above argument.

"⇐": First, we show "⊆", i.e., that $s \in T{\restriction}X'$ for arbitrary $s \in S{\restriction}X'$. By definition, for such s we have $s = s'{\restriction}X'$ for some $s' \in S$. Since $S = T^x_f$, there exists $s'' \in T$ and $a \in \mathcal{A}$ such that $s' = (s'')^x_a$. As $x \notin X'$, we have $s = s'{\restriction}X' = s''{\restriction}X' \in T{\restriction}X'$, as desired.

For the other direction, "⊇", let $s \in T{\restriction}X'$ be arbitrary. Then $s = s'{\restriction}X'$ for some $s' \in T$. As $S = T^x_f$, there exists some $s'' \in S$ and $a \in \mathcal{A}$ such that $s'' = (s')^x_a$. Again we have $s = s'{\restriction}X' = s''{\restriction}X'$, i.e., $s \in S{\restriction}X'$. ◀

▶ **Lemma A.4.** *Let T have domain $\vec{x}$ and S have domain $\vec{x} \cup \{y\}$ (with possibly $y \in X$), and let $X' := \vec{x} \setminus \{y\}$. Then $T{\restriction}X' = S{\restriction}X'$ if and only if $\mathcal{A} \vDash \pi(\vec{x}\langle T\rangle, \vec{x};y\langle S\rangle)$, where $\pi(T,S) := \forall\vec{x}((\exists y T\vec{x}) \leftrightarrow (\exists y S\vec{x};y))$.*

Proof. First, let us consider the case $y \notin X$, i.e., $X' = X$. Then:

$$\begin{aligned}
&T{\restriction}X' = S{\restriction}X' \\
\Leftrightarrow\ &\vec{x}\langle T\rangle = \vec{x}\langle S\rangle && \text{(by Proposition A.1)} \\
\Leftrightarrow\ &\forall \vec{a} : (\vec{a} \in \vec{x}\langle T\rangle \Leftrightarrow \exists b : (\vec{a}, b) \in (\vec{x}; y)\langle S\rangle) && \text{(since } T \text{ has domain } \vec{x}) \\
\Leftrightarrow\ &\mathcal{A} \vDash \pi(\vec{x}\langle T\rangle, \vec{x};y\langle S\rangle) && \text{(since } \exists y T\vec{x} \equiv T\vec{x})
\end{aligned}$$

Next, assume $y \in X$ and w.l.o.g. $y = x_n$. Then $\vec{x};y = \vec{x}$ and $X' = \{x_1, \dots, x_{n-1}\}$. Let $\vec{x}' = (x_1, \dots, x_{n-1})$. Analogously as before, we have:

$$\begin{aligned}
&T{\restriction}X' = S{\restriction}X' \\
\Leftrightarrow\ &\vec{x}'\langle T\rangle = \vec{x}'\langle S\rangle \\
\Leftrightarrow\ &\forall \vec{a} : \big((\exists b : (\vec{a}, b) \in \vec{x}\langle T\rangle) \Leftrightarrow (\exists b : (\vec{a}, b) \in \vec{x};y\langle S\rangle)\big) \\
\Leftrightarrow\ &\mathcal{A} \vDash \pi(\vec{x}\langle T\rangle, \vec{x};y\langle S\rangle)
\end{aligned}$$

◀

▶ **Theorem 3.1.** *Let $\varphi \in \mathsf{FO}(\sim, \mathcal{D})$, let $\vec{x} \supseteq \mathsf{Fr}(\varphi)$ be a tuple of variables, and T be a team in $\mathcal{A}$ with domain $Y \supseteq \vec{x}$. Then $(\mathcal{A}, T) \vDash \varphi$ if and only if $\mathcal{A} \vDash \eta_\varphi^{\vec{x}}(\vec{x}\langle T\rangle)$.*

Proof. Note that $(\mathcal{A}, T) \vDash \varphi \Leftrightarrow (\mathcal{A}, T{\restriction}\vec{x}) \vDash \varphi$ by Proposition 2.5, and $\vec{x}\langle T\rangle = \vec{x}\langle T{\restriction}\vec{x}\rangle$. For this reason, we can assume that T has domain $\vec{x}$. The proof is now by induction on φ.

- If φ is first-order, clearly $(\mathcal{A}, T) \vDash \varphi$ iff $\mathcal{A} \vDash \varphi(\vec{a})$ for all $\vec{a} \in \vec{x}\langle T\rangle$ iff $\mathcal{A} \vDash \eta_\varphi^{\vec{x}}(\vec{x}\langle T\rangle)$.
- If $\varphi = A_i(\vec{t})$ and $\delta_i \in \mathcal{D}$ is a k-ary dependency, then $(\mathcal{A}, T) \vDash A_i(\vec{t})$ iff $\mathcal{A} \vDash \delta_i(\vec{t}\langle T\rangle)$. We prove that this is again equivalent to $\mathcal{A} \vDash \exists S\, \rho(\vec{x}\langle T\rangle, S) \wedge \delta_i(S)$, where $\rho(R, S) := \forall \vec{z}\,(S\vec{z} \leftrightarrow (\exists \vec{x}\, R\vec{x} \wedge \vec{t} = \vec{z}))$.
  It suffices to show that $\mathcal{A} \vDash \rho(\vec{x}\langle T\rangle, S)$ if and only if $S = \vec{t}\langle T\rangle$. As it is straightforward that $\mathcal{A} \vDash \rho(\vec{x}\langle T\rangle, \vec{t}\langle T\rangle)$ holds, let us focus on the direction from left to right. Recall that $\vec{x} \cap \{z_1, \dots, z_k\} = \emptyset$ and that the z_i are pairwise distinct. On that account, suppose $\mathcal{A} \vDash \rho(\vec{x}\langle T\rangle, S)$ and $\vec{a} = (a_1, \dots, a_k) \in \mathcal{A}^k$. By definition of the formula, $\vec{a} \in S$ iff $\mathcal{A} \vDash \exists \vec{x}\, R\vec{x} \wedge \vec{t} = \vec{a}$. However, this is the case iff $\vec{t}\langle s\rangle = \vec{a}$ for some $s \in T$, i.e., $\vec{a} \in \vec{t}\langle T\rangle$.
- The cases $\varphi = {\sim}\psi$ and $\varphi = \psi \wedge \theta$ immediately follow by induction hypothesis.
- If $\varphi = \psi \vee \theta$, then by induction hypothesis, $(\mathcal{A}, T) \vDash \varphi$ iff there are $S, U \subseteq T$ such that $T = S \cup U$ and $\mathcal{A} \vDash \eta_\psi^{\vec{x}}(\vec{x}\langle S\rangle) \wedge \eta_\theta^{\vec{x}}(\vec{x}\langle U\rangle)$. Let $R := \vec{x}\langle T\rangle$.
  Then due to Proposition A.2, the above is equivalent to the existence of $P, Q \subseteq \mathcal{A}^n$ such that $R = P \cup Q$ and $\mathcal{A} \vDash \eta_\psi^{\vec{x}}(P) \wedge \eta_\theta^{\vec{x}}(Q)$, and consequently to $\mathcal{A} \vDash \exists S \exists U \forall \vec{x}(R\vec{x} \leftrightarrow (S\vec{x} \vee U\vec{x})) \wedge \eta_\psi^{\vec{x}}(S) \wedge \eta_\theta^{\vec{x}}(U)$.
- If $\varphi = \exists y\, \psi$, by Proposition A.3, then $(\mathcal{A}, T) \vDash \varphi$ iff $(\mathcal{A}, S) \vDash \psi$ for some team S with domain $\vec{x} \cup \{y\}$ such that $T{\restriction}X' = S{\restriction}X'$, where $X' := \vec{x} \setminus \{y\}$. By Lemma A.4 and by induction hypothesis, this is the case iff $(\mathcal{A}, \vec{x}\langle T\rangle) \vDash \exists S \forall \vec{x}((\exists y R\vec{x}) \leftrightarrow (\exists y S\vec{x}; y)) \wedge \eta_\psi^{\vec{x};y}(S)$.
- The case $\varphi = \forall y\, \psi$ is proven analogously to $\exists$. The additional clause $(R\vec{x} \to \forall y\, S\vec{x};y)$ ensures that the supplementing function is constant and $f(s) = \mathsf{dom}\,\mathcal{A}$. ◀

▶ **Theorem 3.5.** *Let $\varphi \in \mathsf{FO}(\sim, \mathcal{D})$, let $\vec{x} \supseteq \mathsf{Fr}(\varphi)$ be a tuple of variables, and T be a team in $\mathcal{A}$ with domain $Y \supseteq \vec{x}$. If $p(n) \geq |T| \cdot n^{\mathsf{qr}(\varphi)}$ or $p(n) \geq n^{\mathsf{w}(\varphi)}$, then $(\mathcal{A}, T) \vDash \varphi$ if and only if $\mathcal{A} \vDash \zeta_\varphi^{\vec{x},p}(\vec{x}\langle T\rangle)$.*

Proof for $p(n) \geq |T| \cdot n^{\mathsf{qr}(\varphi)}$. Assume $\mathcal{A}$, T as above, let $m := \mathsf{qr}(\varphi)$ and $p(n) \geq n^m$. The idea is to show that $\eta_\varphi^{\vec{x}}$ and $\zeta_\varphi^{\vec{x},p}$ agree on $(\mathcal{A}, \mathcal{J})$ for all "sufficiently sparse" $\mathcal{J}$ (cf. Theorem 3.1). Formally, let $\ell \leq m$ and let $(\mathcal{A}, \mathcal{J})$ be a second-order interpretation such that

$|\mathcal{J}(R)| \leq |T| \cdot |\mathcal{A}|^\ell$ for all relations $R \in \mathsf{dom}\,\mathcal{J}$. Then we prove for all $\varphi \in \mathsf{FO}(\sim, \mathcal{D})$ with $\mathsf{qr}(\varphi) \leq m - \ell$ and $\vec{x} \supseteq \mathsf{Fr}(\varphi)$ that $(\mathcal{A}, \mathcal{J}) \vDash \eta_\varphi^{\vec{x}}$ if and only if $(\mathcal{A}, \mathcal{J}) \vDash \zeta_\varphi^{\vec{x},p}$. For $\ell = 0$, this yields the theorem, since $|\vec{x}\langle T\rangle| \leq |T| \cdot |\mathcal{A}|^0$.

The proof is by induction on φ. We distinguish the following cases.

- If $\varphi \in \mathsf{FO}$, then there is nothing to prove, as $\eta_\varphi^{\vec{x}} = \zeta_\varphi^{\vec{x},p}$.
- If $\varphi = \sim\psi$ or $\varphi = \psi \wedge \theta$, then the inductive step is clear.
- If $\varphi = A_i\vec{t}$ for some k-ary $\delta_i \in \mathcal{D}$, then $\zeta_\varphi^{\vec{x},p}(R) = \exists^p S\, \rho(R,S)$ and $\eta_\varphi^{\vec{x}}(R) = \exists S\, \rho(R,S)$, where

$$\rho(R,S) = \forall \vec{z}\,(S\vec{z} \leftrightarrow (\exists \vec{x}\, R\vec{x} \wedge \vec{t} = \vec{z}\,)) \wedge \delta_i(S).$$

  We show that $\mathcal{A} \vDash \eta_\varphi^{\vec{x}}(R)$ implies $\mathcal{A} \vDash \zeta^{\vec{x},p}(R)$, as the other direction is trivial.
  On that account, suppose $\mathcal{A} \vDash \rho(R,S)$ for some $S \subseteq \mathcal{A}^k$. We prove that necessarily $|S| \leq |R|$ by constructing some injective $f\colon S \to R$. Then $\mathcal{A} \vDash \exists^p S\, \rho(R,S)$, as by assumption, $|S| \leq |R| \leq |T| \cdot |\mathcal{A}|^\ell \leq |T| \cdot |\mathcal{A}|^m \leq p(|\mathcal{A}|)$.
  We define f as follows. For every $\vec{a} \in S$, let $f(\vec{a})$ be some $\vec{b} \in R$ such that $\vec{t}\langle\{\vec{x} \mapsto \vec{b}\}\rangle = \vec{a}$. By $\rho(R,S)$, such $\vec{b}$ must exist. Clearly, f is injective.
- If $\varphi = \psi \vee \theta$, then $\zeta_\varphi^{\vec{x},p}(R) = \exists^p S\, \exists^p U\, \rho$ and $\eta_\varphi^{\vec{x}}(R) = \exists S\, \exists U\, \rho'$, where

$$\rho(R,S,U) = \forall\vec{x}(R\vec{x} \leftrightarrow (S\vec{x} \vee U\vec{x})) \wedge \zeta_\psi^{\vec{x},p}(S) \wedge \zeta_\theta^{\vec{x},p}(U),$$
$$\rho'(R,S,U) = \forall\vec{x}(R\vec{x} \leftrightarrow (S\vec{x} \vee U\vec{x})) \wedge \eta_\psi^{\vec{x}}(S) \wedge \eta_\theta^{\vec{x}}(U).$$

  Suppose $|R| \leq |T| \cdot |\mathcal{A}|^\ell$ and $\mathsf{qr}(\varphi) \leq m - \ell$. Clearly $\mathsf{qr}(\psi), \mathsf{qr}(\theta) \leq \mathsf{qr}(\varphi)$.
  Let $\mathcal{A} \vDash \eta_\varphi^{\vec{x}}(R)$, i.e., $\mathcal{A} \vDash \rho'(R,S,U)$ for some $S, U \subseteq \mathcal{A}^{|\vec{x}|}$.
  It is easy to see that ρ' forces $|S|, |U| \leq |R|$. Since $|R| \leq |T| \cdot |\mathcal{A}|^\ell$ by assumption, we can apply the induction hypothesis to $\eta_\psi^{\vec{x}}(S)$ and $\eta_\theta^{\vec{x}}(U)$ and derive $\mathcal{A} \vDash \rho(R,S,U)$ from $\mathcal{A} \vDash \rho'(R,S,U)$. Since in particular $|S|, |U| \leq p(|\mathcal{A}|)$, we conclude $\mathcal{A} \vDash \zeta_\varphi^{\vec{x},p}(R)$. The other direction is trivial due to the inductivion hypothesis, since $\rho(R,S)$ entails $\rho'(R,S)$.
- If $\varphi = \exists y\, \psi$, then $\zeta_\varphi^{\vec{x},p}(R) = \exists^p S\, \rho(R,S)$ and $\eta_\varphi^{\vec{x}}(R) = \exists S\, \rho'(R,S)$, where

$$\rho(R,S) = \forall\vec{x}((\exists y R\vec{x}) \leftrightarrow (\exists y\, S\vec{x};y)) \wedge \zeta_\psi^{\vec{x};y,p}(S),$$
$$\rho'(R,S) = \forall\vec{x}((\exists y R\vec{x}) \leftrightarrow (\exists y\, S\vec{x};y)) \wedge \eta_\psi^{\vec{x};y}(S).$$

  Suppose $|R| \leq |T| \cdot |\mathcal{A}|^\ell$ and $\mathsf{qr}(\varphi) \leq m - \ell$. We show that $\mathcal{A} \vDash \eta_\varphi^{\vec{x}}(R)$ implies $\mathcal{A} \vDash \zeta_\varphi^{\vec{x},p}(R)$. The other direction is then again similar.
  Assuming $\mathcal{A} \vDash \eta_\varphi^{\vec{x}}(R)$, there exists $S \subseteq \mathcal{A}^{|\vec{x};y|}$ such that $\mathcal{A} \vDash \rho'(R,S)$. As a first step, we erase unnecessary elements from S. Note that S occurs in ρ' only in atomic formulas $S\vec{x};y$, i.e., with a fixed argument tuple $\vec{x};y$. Let $(v_1, \ldots, v_r) := \vec{x};y$. If now $v_i = v_j$ for some $1 \leq i < j \leq r$, then every tuple $(a_1, \ldots, a_r)$ with $a_i \neq a_j$ can be safely deleted from S. Formally, if $S^* := \vec{x};y\langle V\rangle \cap S$, where $V = \{s\colon \vec{x} \cup \{y\} \to \mathcal{A}\}$ is the full team with domain $\vec{x} \cup \{y\}$, then $\mathcal{A} \vDash \rho'(R,S)$ if and only if $\mathcal{A} \vDash \rho'(R,S^*)$, which can be shown by straightforward induction.
  Note that $\mathsf{qr}(\psi) = \mathsf{qr}(\varphi) - 1 \leq m - (\ell+1)$. Consequently, to apply the induction hypothesis, we prove $|S^*| \leq |R| \cdot |\mathcal{A}| \leq |T| \cdot |\mathcal{A}|^{\ell+1}$ by presenting some injective $f\colon S^* \to R \times \mathcal{A}$.
  If $y \notin \vec{x}$, let f be the identity, as ρ' ensures that $(\vec{a}, b) \in S^*$ implies $\vec{a} \in R$. However, if $y \in \vec{x}$, then we define $f(\vec{a})$ as follows. By construction, $\vec{a} \in S^*$ equals $\vec{x}\langle s\rangle$ for some $s\colon \vec{x} \to \mathcal{A}$. Again by ρ', there is $\hat{s}\colon \vec{x} \to \mathcal{A}$ such that $\vec{x}\langle\hat{s}\rangle \in R$ and $s = \hat{s}^y_{s(y)}$. Let now $f(\vec{a}) := (\vec{x}\langle\hat{s}\rangle, s(y))$. Then f is injective.

Hence, by induction hypothesis, we can replace $\eta^{\vec{x}}_{\varphi}$ by $\zeta^{\vec{x},p}_{\varphi}$ and obtain $\mathcal{A} \vDash \rho(R, S^*)$. Since $|S^*| \leq |\mathcal{A}|^{\ell+1} \leq p(|\mathcal{A}|)$, we obtain $\mathcal{A} \vDash \exists^p S\, \rho(R, S)$.

- The case $\varphi = \forall y\, \psi$ is proven similarly to $\varphi = \exists y\, \psi$. ◀

Proof for $p(n) \geq n^{\mathsf{w}(\varphi)}$. We can apply the same argument as in the $\exists$-case of the previous proof. Suppose S is a second-order variable. Then S appears in $\eta^{\vec{x}}_{\varphi}$ only in atomic formulas of the form $S\vec{t}$ for a fixed $\vec{t}$. Accordingly, it suffices to let $\exists S$ range over subsets of $\vec{t}\langle V\rangle$, where $\vec{y} := \mathsf{Var}(\vec{t})$ and $V := \{s \colon \vec{y} \to \mathcal{A}\}$.

(We consider terms $\vec{t}$ instead of only variables to account for the translations of dependencies, where S can have terms as arguments.)

Since $\vec{y}$ contains at most $\mathsf{w}(\varphi)$ distinct variables, $|V| \leq |\mathcal{A}|^{\mathsf{w}(\varphi)} \leq p(|\mathcal{A}|)$. Consequently, every second-order quantifier $\exists S$ can be replaced by $\exists^p S$, which implies $\mathcal{A} \vDash \eta^{\vec{x}}_{\varphi}(\vec{x}\langle T\rangle) \Leftrightarrow \mathcal{A} \vDash \zeta^{\vec{x},p}_{\varphi}(\vec{x}\langle T\rangle)$. ◀

Proof of Lemma 4.1

▶ **Lemma A.5.** *The following laws hold for* $\mathsf{FO}(\sim)$*:*

$$\alpha \wedge \bigwedge_{i=1}^{n} \mathsf{E}\beta_i \equiv \bigvee_{i=1}^{n} (\alpha \wedge \mathsf{E}\beta_i) \tag{2}$$

$$\bigvee_{i=1}^{n} (\alpha_i \wedge \mathsf{E}\beta_i) \equiv \left(\bigvee_{i=1}^{n} \alpha_i\right) \wedge \bigwedge_{i=1}^{n} \mathsf{E}(\alpha_i \wedge \beta_i) \tag{3}$$

$$(\vartheta_1 \owedge \vartheta_2) \vee \vartheta_3 \equiv (\vartheta_1 \vee \vartheta_3) \owedge (\vartheta_2 \vee \vartheta_3) \tag{4}$$

$$\vartheta_1 \vee (\vartheta_2 \owedge \vartheta_3) \equiv (\vartheta_1 \vee \vartheta_2) \owedge (\vartheta_1 \vee \vartheta_3) \tag{5}$$

$$\exists x\, (\vartheta_1 \owedge \vartheta_2) \equiv (\exists x\, \vartheta_1) \owedge (\exists x\, \vartheta_2) \tag{6}$$

$$\exists x\, (\vartheta_1 \vee \vartheta_2) \equiv (\exists x\, \vartheta_1) \vee (\exists x\, \vartheta_2) \tag{7}$$

$$\exists x\, (\alpha \wedge \mathsf{E}\beta) \equiv (\exists x\, \alpha) \wedge \mathsf{E}\, \exists x\, (\alpha \wedge \beta) \tag{8}$$

$$\forall x\, (\vartheta_1 \wedge \vartheta_2) \equiv (\forall x\, \vartheta_1) \wedge (\forall x\, \vartheta_2) \tag{9}$$

$$\forall x \sim\vartheta \equiv \sim\forall x\, \vartheta \tag{10}$$

Proof. For (2), (3) and (7), see Lück [29, Lemma 4.13, 4.14 and D.1], respectively. For (4)–(6), see Galliani [13, Proposition 5]. For (9)–(10), see Väänänen [38, Chapter 8].

For (8), the direction "$\vDash$" is clear, as $\alpha \wedge \mathsf{E}\beta$ implies both α and $\mathsf{E}(\alpha \wedge \beta)$. For the converse direction, suppose $(\mathcal{A}, T) \vDash \exists x \alpha$ and $(\mathcal{A}, \hat{s}) \vDash \exists x(\alpha \wedge \beta)$ for some $\hat{s} \in T$. Then there are $f \colon T \to \mathfrak{P}(\mathcal{A}) \setminus \{\emptyset\}$ and $b \in \mathcal{A}$ such that $(\mathcal{A}, T^x_f) \vDash \alpha$ and $(\mathcal{A}, \hat{s}^x_b) \vDash \alpha \wedge \beta$. Define $g(\hat{s}) = f(\hat{s}) \cup \{b\}$ and $g(s) = f(s)$ for $s \in T \setminus \{\hat{s}\}$. Then $T^x_g = T^x_f \cup \{s^x_b\}$. Consequently, $(\mathcal{A}, T^x_g) \vDash \alpha \wedge \mathsf{E}\beta$, hence $(\mathcal{A}, T) \vDash \exists x\, (\alpha \wedge \mathsf{E}\beta)$. ◀

▶ **Lemma 4.1.** *Every τ-$\mathsf{FO}^n_k(\sim)$-formula φ is equivalent to a formula of the form*

$$\psi := \bigovee_{i=1}^{n} \left(\alpha_i \wedge \bigwedge_{j=1}^{m_i} \mathsf{E}\beta_{i,j} \right)$$

such that $\{\alpha_1, \ldots, \alpha_n, \beta_{1,1}, \ldots, \beta_{n,m_n}\} \subseteq \tau$-$\mathsf{FO}^n_k$ *and* $|\psi| \leq \exp_{\mathcal{O}(|\varphi|)}(1)$.

In what follows, *disjunctive normal form* (DNF) refers to formulas in the above form.

Proof. We construct the formula ψ by induction on φ. In each inductive step, it grows at most exponentially.

- If φ is a Boolean combination of FO_k^n-formulas (i.e., over $\sim$ and $\wedge$), then we obtain a DNF of size $\leq |\varphi| \cdot 2^{|\varphi|}$ similarly as for ordinary propositional logic.
- If $\varphi = \vartheta_1 \vee \vartheta_2$ for ϑ_1, ϑ_2 in DNF, then

$$\begin{aligned}
\varphi &= \bigovee_{i=1}^{n} \left(\alpha_i \wedge \bigwedge_{j=1}^{m_i} \mathsf{E}\beta_{i,j}\right) \vee \bigovee_{i=1}^{k} \left(\gamma_i \wedge \bigwedge_{j=1}^{\ell_i} \mathsf{E}\delta_{i,j}\right) \\
&\equiv \bigovee_{i=1}^{n} \bigvee_{j=1}^{m_i} (\alpha_i \wedge \mathsf{E}\beta_{i,j}) \vee \bigovee_{i=1}^{k} \bigvee_{j=1}^{\ell_i} (\gamma_i \wedge \mathsf{E}\delta_{i,j}) && \text{(Lemma A.5, (2))} \\
&\equiv \bigovee_{\substack{1\leq i_1 \leq n \\ 1 \leq i_2 \leq k}} \bigvee_{j=1}^{m_{i_1}} (\alpha_{i_1} \wedge \mathsf{E}\beta_{i_1,j}) \vee \bigvee_{j=1}^{\ell_{i_2}} (\gamma_{i_2} \wedge \mathsf{E}\delta_{i_2,j}) && \text{(Lemma A.5, (4) and (5))} \\
&\equiv \bigovee_{i=1}^{n\cdot k} \bigvee_{j=1}^{o_i} (\mu_{i,j} \wedge \mathsf{E}\nu_{i,j}) && \text{(for some } \mu_{i,j}, \nu_{i,j} \in \mathsf{FO}_k^n\text{)} \\
&\equiv \bigovee_{i=1}^{n\cdot k} \Big(\bigvee_{j=1}^{o_i} \mu_{i,j}\Big) \wedge \bigwedge_{j=1}^{o_i} \mathsf{E}(\mu_{i,j} \wedge \nu_{i,j}), && \text{(Lemma A.5, (3))}
\end{aligned}$$

where the final DNF has size polynomial in $|\vartheta_1| + |\vartheta_2| \leq |\varphi|$.
- If $\varphi = \exists x\, \vartheta$ for ϑ in DNF, then

$$\begin{aligned}
\varphi &\equiv \exists x \bigovee_{i=1}^{n} \bigvee_{j=1}^{m_i} (\alpha_i \wedge \mathsf{E}\beta_{i,j}) && \text{(Lemma A.5, (2))} \\
&\equiv \bigovee_{i=1}^{n} \bigvee_{j=1}^{m_i} \exists x\, (\alpha_i \wedge \mathsf{E}\beta_{i,j}) && \text{(Lemma A.5, (6) and (7))} \\
&\equiv \bigovee_{i=1}^{n} \bigvee_{j=1}^{m_i} \Big((\exists x\, \alpha_i) \wedge \mathsf{E}\exists x\, (\alpha_i \wedge \beta_{i,j})\Big) && \text{(Lemma A.5, (8))} \\
&\equiv \bigovee_{i=1}^{n} \Big(\bigvee_{j=1}^{m_i} \exists x\, \alpha_i\Big) \wedge \bigwedge_{j=1}^{m_i} \mathsf{E}\exists x\, (\alpha_i \wedge \beta_{i,j})), && \text{(Lemma A.5, (3))}
\end{aligned}$$

which is again a DNF of polynomial size.
- Finally, the $\forall$ case is by repeated application of (9) and (10) of Lemma A.5. ◀

A Tight Analysis of the Parallel Undecided-State Dynamics with Two Colors

Andrea Clementi
Università *Tor Vergata* di Roma, Italy
clementi@mat.uniroma2.it

Mohsen Ghaffari
ETH Zürich, Switzerland
ghaffari@inf.ethz.ch

Luciano Gualà
Università *Tor Vergata* di Roma, Italy
guala@mat.uniroma2.it

Emanuele Natale
Max Planck Institute for Informatics, Germany
enatale@mpi-inf.mpg.de

Francesco Pasquale[1]
Università *Tor Vergata* di Roma, Italy
pasquale@mat.uniroma2.it

Giacomo Scornavacca
Università degli Studi dell'Aquila, Italy
giacomo.scornavacca@graduate.univaq.it

Abstract

The *Undecided-State Dynamics* is a well-known protocol for distributed consensus. We analyze it in the parallel $\mathcal{PULL}$ communication model on the complete graph with n nodes for the *binary* case (every node can either support one of *two* possible colors, or be in the undecided state).

An interesting open question is whether this dynamics is an efficient *Self-Stabilizing* protocol, namely, starting from an arbitrary initial configuration, it reaches consensus *quickly* (i.e., within a polylogarithmic number of rounds). Previous work in this setting only considers initial color configurations with no undecided nodes and a large *bias* (i.e., $\Theta(n)$) towards the majority color.

In this paper we present an *unconditional* analysis of the Undecided-State Dynamics that answers to the above question in the affirmative. We prove that, starting from *any* initial configuration, the process reaches a monochromatic configuration within $\mathcal{O}(\log n)$ rounds, with high probability. This bound turns out to be tight. Our analysis also shows that, if the initial configuration has bias $\Omega(\sqrt{n \log n})$, then the dynamics converges toward the initial majority color, with high probability.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed Consensus, Self-Stabilization, $\mathcal{PULL}$ Model, Markov Chains

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.28

Related Version A full version of the paper is available at [14], `https://arxiv.org/abs/1707.05135v3`.

[1] Partly supported by the University of "Tor Vergata" under research programme "Mission: Sustainability" project ISIDE (grant no. E81I18000110005)

1 Introduction

Simple local mechanisms for *Consensus* problems in distributed systems recently received a lot of attention [2, 1, 18, 19, 28, 31]. In one of the basic versions of the consensus problem the system consists of anonymous entities (*nodes*) each one initially supporting a *color* out of a finite set of colors Σ. Nodes run elementary operations and interact by exchanging messages. A *Consensus Protocol* is a local procedure that makes the system converge to a *monochromatic* configuration, where all nodes support the same color. Consensus has to be *valid*, i.e., the "winning" color must be one of those initially supported by at least one node. A crucial property of a consensus protocol is *self-stabilization* [6, 18, 30]: Informally, if the system is "perturbed" by some external event and moved to an arbitrary configuration, then the protocol must bring the system back to a valid consensus and, moreover, once the system reaches consensus, it must remain in that configuration forever, unless a further external event takes place[2]. Self-stabilizing consensus processes are fundamental building-blocks that play an important role in coordination tasks and self-organizing behavior in population systems [11, 13, 19, 29].

We study the consensus problem in the $\mathcal{PULL}$ *communication model* [12, 17, 24] where, at every round, each active node of a communication network contacts one neighbor uniformly at random to pull information. A natural consensus protocol in this model is the *Undecided-State Dynamics*[3] (for short, the *U-Dynamics*) in which the state of a node can be either a color or the *undecided state*. When a node is activated, it pulls the state of a random neighbor and updates its state according to the following updating rule (see Table 1): If a colored node pulls a different color from its current one, then it becomes undecided, while in all other cases it keeps its color; moreover, if the node is in the undecided state then it will take the state of the pulled neighbor.

The U-Dynamics has been previously studied in both *sequential* [2] and *parallel* [4] models: Informally, in the former only one random node is activated at every round and it updates its state according to the local rule, while in the latter all nodes are activated at every round and they update their state, synchronously.

As for the sequential model[4], [2] provides an unconditional analysis showing (among other results) that the U-Dynamics is a self-stabilizing protocol for *binary* consensus (i.e., when $|\Sigma| = 2$) in the complete graph with n nodes. They show the convergence time is $\mathcal{O}(n \log n)$ (and, thus, work per node is $\mathcal{O}(\log n)$), *with high probability*[5]. This result also clarifies the algorithmic interest for this process. Indeed, the U-Dynamics can be seen as a variant of the popular *Voter Model* [9, 23, 26] where every active node simply takes the color it pulls at every round. On one hand, the Voter Model uses minimal number of node states (i.e. $|\Sigma|$) and takes $\Theta(n)$ work per node to reach consensus (see for instance [25]). On the other hand, the U-Dynamics exponentially improves the work complexity by using one additional state, only. Further motivations on the U-Dynamics are discussed in Subsection 1.2.

We remark that the stochastic process induced by the parallel dynamics significantly departs from the one induced by the sequential dynamics. As a simple evidence of such qualitative differences, observe that, starting from a configuration with no undecided nodes,

---

[2] Notice that, according to previous work [6, 18], we require self-stabilization to hold *with high probability*.

[3] In some previous papers [31] on the binary case ($|\Sigma| = 2$), this protocol has been also called the *Third-State Dynamics*. We here prefer the term "undecided" since it also holds for the non-binary case and, moreover, the term well captures the role of this additional state.

[4] [2] in fact considers the *Population-Protocol* model which is, in our specific context, equivalent to the sequential $\mathcal{PULL}$ model.

[5] As usual, we say that an event $\mathcal{E}_n$ holds *w.h.p.* if $\mathbf{P}(\mathcal{E}_n) \geqslant 1 - n^{-\Theta(1)}$.

in the parallel case the system might end up in the *non-valid*, monochromatic configuration where all nodes are undecided (this would happen if, for example, at the first round every node pulled a node with the other color). On the other hand, it is easy to see that in the sequential case the process always ends up in a monochromatic configuration with no undecided nodes, unless it starts from a configuration with all nodes undecided. The crucial difference lies in the random number of nodes that may change color at every round: In the sequential model, this is at most one[6], while in the parallel one, *all* nodes may change state in one round and, for most phases of the process, the expected number of changes is indeed linear in n. The above difference is one of the main reasons why no general techniques are currently available to extend any quantitative analysis for the sequential process to the corresponding parallel one (and vice versa). In particular, the analysis in [2] strongly uses the fact that only one node can change state in one round in order to derive a suitable supermartingale argument to bound the stopping time of the process. It thus fully covers the case of sequential interaction models, but it is not helpful to understand the evolution of the U-Dynamics process on any interaction model in which the number of nodes that may change state in one round is not bounded by some absolute constant.

As for the parallel $\mathcal{PULL}$ model, while it is easy to verify that the U-Dynamics achieves consensus in the complete graph (with high probability), the convergence time of this dynamics is still an interesting open issue, even in the binary case. Indeed, in [4] the authors analyze the U-Dynamics in the parallel $\mathcal{PULL}$ model on the complete graph for any number $k = o(n^{1/3})$ of colors. However, their analysis requires the initial configuration to have a relatively-large *bias* $s = c_1 - c_2$ between the size c_1 of the (unique) initial plurality and the size c_2 of the second-largest color. More in details, in [4] it is assumed that $c_1 \geqslant \alpha c_2$, for some absolute constant $\alpha > 1$ and, thus, this condition for the binary case would result into requiring a very-large initial bias, i.e., $s = \Theta(n)$. This analysis clearly does not show that the U-Dynamics efficiently solves the binary consensus problem, mainly because it does not manage *balanced* initial configurations.

1.1 Our results

We prove that, starting from any color configuration[7] on the complete graph, the U-Dynamics reaches a monochromatic configuration (thus consensus) within $\mathcal{O}(\log n)$ rounds, with high probability. This bound is tight since, for some (in fact, a large number of) initial configurations, the process requires $\Omega(\log n)$ rounds to converge.

Not assuming a large initial bias of the majority color significantly complicates the analysis. Indeed, the major technical issues arise from the analysis of *balanced* initial configurations where the system "needs" to *break symmetry* without having a strong expected drift towards any color. Previous analysis of this phase consider either *sequential* processes of interacting particles that can be modeled as *birth-and-death* chains [2] or parallel processes whose local rule is fully symmetric w.r.t. the states/colors of the nodes (such as majority rules) [6, 18]. The U-Dynamics process falls neither in the former nor in the latter scenario: It works in parallel rounds and the role of the undecided nodes makes the local rule not symmetric. We believe this issue has a *per-se* scientific interest since symmetry-breaking phenomena yielded by simple and local mechanisms plays a central role in key aspects of population systems [10] and, more generally, in the emerging field of natural algorithms [13].

[6] This number actually becomes 2 if the sequential communication model activates a random edge per round, rather than one single node [2].

[7] Our analysis also considers initial configurations with undecided nodes.

Informally speaking, in Section 4 we deal with almost-balanced starting configurations. By devising a coupling to a "simplified" pruned process, we show that the analysis of this *symmetry-breaking* phase essentially reduces to the analysis of a specific regime where the number q of undecided nodes remains a suitable constant fraction of n *until* the magnitude of the bias s reaches $\Omega(\sqrt{n \log n})$: In other words, during this regime, with very high probability the system never jumps to almost-balanced configurations having either too many or too few undecided nodes. This fact is crucial for two main reasons: Along this regime, (i) the *variance* of the bias s is large (i.e. $\Theta(n)$) and (ii) whenever the bias s is $\Omega(\sqrt{n})$, its drift turns out to be *exponential* with non-negligible, increasing probability (w.r.t. s itself). Then, we prove a variant of a general Lemma [18] that provides a logarithmic bound on the hitting time of Markov chains satisfying Properties (i) and (ii) above.

The symmetry-breaking phase terminates when the U-Process reaches some configuration having a bias $s = \Omega(\sqrt{n \log n})$. Then we prove that, starting from *any* configuration having that bias, the process reaches consensus within $\mathcal{O}(\log n)$ rounds, with high probability. Even though our analysis of this "majority" part of the process is based on standard concentration arguments, it must cope with some *non-monotone* behavior of the key random variables (such as the bias and the number of undecided nodes at the next round): Again, this is due to the non-symmetric role played by the undecided nodes. A good intuition about this "non-monotone" process can be gained by looking at the mutually-related formulas giving the expectation of such key random variables (see Equations (1)-(3)). Our refined analysis shows that, during this majority phase, the winning color never changes and, thus, the U-Dynamics also ensures Plurality Consensus in logarithmic time whenever the initial bias is $s = \Omega(\sqrt{n \log n})$.

Interestingly enough, we also show that configurations with $s = \mathcal{O}(\sqrt{n})$ exist so that the system may converge toward the minority color with non-negligible probability.

1.2 Further motivation and related work

On the U-Dynamics. The interest in the U-Dynamics arises in fields beyond the borders of Computer Science and it seems to have a key-role in important biological processes modeled as so-called chemical reaction networks [11, 19]. For such reasons, the convergence time of this dynamics has been analyzed on different communication models [2, 3, 4, 27, 31]. As previously mentioned, the U-Dynamics has been analyzed in the parallel $\mathcal{PULL}$ model in [4] and their results concern the evolution of the process for the multi-color case when there is a significant initial bias (as a protocol for plurality consensus).

As for the sequential model, the U-Dynamics has been introduced and analyzed in [2] on the complete graph. They prove that this dynamics, with high probability, converges to a valid consensus within $\mathcal{O}(n \log n)$ activations and, moreover, it converges to the majority whenever the initial bias is $\omega\left(\sqrt{n \log n}\right)$.

Still concerning the sequential model, [27] recently analyzes, besides other protocols, the U-Dynamics in arbitrary graphs where in the initial configuration each node samples uniformly at random one out of two colors. In this (average-case) setting, they prove that the system converges to the initial majority color with higher probability than the initial minority one. They also give results for special classes of graphs where the minority can win with large probability if the initial configuration is chosen in a suitable way. Their proof for this result relies on an exponentially-small upper bound on the probability that a certain minority can win in the complete graph (see [27] for more details). In [3, 7, 20, 31], the same dynamics for the binary case has been analyzed in other sequential communication models.

On some other consensus dynamics. Recently, further simple consensus protocols have been deeply analyzed in several papers, thus witnessing the high interest of the scientific community on such processes [2, 5, 9, 11, 15, 16, 18, 31].

The parallel 3-Majority is a protocol where at every round, each node picks the colors of three random neighbors and updates its color according to the majority rule (taking the first one or a random one to break ties). The authors of [5] assume that the bias is $\Omega(\min\{\sqrt{2k}, (n/\log n)^{1/6}\} \cdot \sqrt{n \log n})$. Under this assumption, they prove that consensus is reached with high probability in $\mathcal{O}(\min\{k, (n/\log n)^{1/3}\} \cdot \log n)$ rounds, and that this is tight if $k \leqslant (n/\log n)^{1/4}$. The first result without bias [6] restricts the number of initial colors to $k = \mathcal{O}(n^{1/3})$. Under this assumption, they prove that 3-Majority reaches consensus with high probability in $\mathcal{O}((k^2(\log n)^{1/2} + k \log n) \cdot (k + \log n))$ rounds. Very recently, such result has been generalized to the whole range of k in [8].

In [18] the authors provide an analysis of the *3-median* rule, in which every node updates its value to the median of its current value and the values of two randomly chosen neighbors. They show that this dynamics converges to an almost-agreement configuration (which is even a good approximation of the global median) within $\mathcal{O}(\log k \cdot \log\log n + \log n)$ rounds, w.h.p. It turns out that, in the binary case, the median rule is equivalent to the 2-Choices dynamics, a variant of 3-Majority, thus their result implies that this is a stabilizing consensus protocol with $\mathcal{O}(\log n)$ convergence time. As mentioned earlier, our analysis borrows a hitting-time bound on general Markov chains from [18].

Very recently, [22] provides an optimal bound $\Theta(k \log n)$ for the 2-Choices dynamics on the complete graph even under some dynamic adversary. In [15, 16], the authors consider the 2-Choices dynamics for plurality consensus in the binary case (i.e. $k = 2$). For random d-regular graphs, [15] proves that all nodes agree on the majority color in $\mathcal{O}(\log n)$ rounds, provided that the bias is $\omega(n \cdot \sqrt{1/d + d/n})$. The same holds for arbitrary d-regular graphs if the bias is $\Omega(\lambda_2 \cdot n)$, where λ_2 is the second largest eigenvalue of the transition matrix. In [16], these results are extended to general expander graphs.

1.3 Structure of the paper

In Section 2, we provide some preliminaries and an informal description of the expected evolution of the U-Process. In Section 3, we formally state the main results of this paper and describe an outline of the corresponding proofs. Section 4 is devoted to the description of the tight analysis of the symmetry-breaking phase. The analysis of the "majority" phase of the process is given in Section 5. Conclusions and some open questions are discussed in Section 6. Due to lack of space, all the omitted proofs can be found in the full-version of the paper [14].

2 Preliminaries

We analyze the parallel version of the dynamics called U-Dynamics in the (uniform) $\mathcal{PULL}$ model on the complete graph: Starting from an initial configuration where every node supports a color, i.e. a value from a set Σ of k possible colors[8], at every round, each node u pulls the color of a randomly-selected neighbor v. If the color of node v differs from its own color, then node u enters in an *undecided* state (an extra state with no color). When a

[8] W.l.o.g. we can define $\Sigma = [k]$ where $[k] = \{1, 2, \cdots, k\}$.

Table 1 The update rule of the U-Dynamics where $i, j \in [k]$ and $i \neq j$.

$u \backslash v$	undecided	color i	color j
undecided	undecided	i	j
i	i	i	undecided
j	j	undecided	j

node is in the undecided state and pulls a color, it gets that color. Finally, a node that pulls either an undecided node or a node with its own color remains in its current state.

In this paper we consider the case in which there are two possible colors (say color `Alpha` and color `Beta`). Let us name $\mathcal{C}$ the space of all possible configurations and observe that, since the graph is complete, a configuration $\mathbf{x} \in \mathcal{C}$ is uniquely determined by fixing the number of `Alpha`-colored nodes and the number of `Beta`-colored ones, say $a(\mathbf{x})$ and $b(\mathbf{x})$, respectively.

It is convenient to give names also to two other quantities that will appear often in the analysis: The number $q(\mathbf{x}) = n - a(\mathbf{x}) - b(\mathbf{x})$ of undecided nodes and the difference $s(\mathbf{x}) = a(\mathbf{x}) - b(\mathbf{x})$ called the *bias* of $\mathbf{x}$. Notice that any two of the quantities $a(\mathbf{x}), b(\mathbf{x}), q(\mathbf{x})$, and $s(\mathbf{x})$ uniquely determine the configuration. When it will be clear from the context, we will omit $\mathbf{x}$ and write a, b, q, and s instead of $a(\mathbf{x}), b(\mathbf{x}), q(\mathbf{x})$, and $s(\mathbf{x})$.

Observe that the U-Dynamics defines a finite-state Markov chain $\{\mathbf{X}_t\}_{t \geqslant 0}$ with state space $\mathcal{C}$ and three absorbing states, namely, $q = n$, $a = n$, and $b = n$. We call *U-Process* the random process obtained by applying the U-Dynamics starting at a given state. Once we fix the configuration $\mathbf{x}$ at round t of the process, i.e. $\mathbf{X}_t = \mathbf{x}$, we use the capital letters A, B, Q, and S to refer to the random variables $a(\mathbf{X}_{t+1}), b(\mathbf{X}_{t+1}), q(\mathbf{X}_{t+1})$, $s(\mathbf{X}_{t+1})$.

From the definition of U-Dynamics it is easy to calculate the following expected values (see also Section 3 in [4]):

$$\mathbf{E}\left[A \mid \mathbf{X}_t = \mathbf{x}\right] = a\left(\frac{a+2q}{n}\right), \tag{1}$$

$$\mathbf{E}\left[Q \mid \mathbf{X}_t = \mathbf{x}\right] = \frac{q^2+2ab}{n}, \tag{2}$$

$$\mathbf{E}\left[S \mid \mathbf{X}_t = \mathbf{x}\right] = \frac{a(a+2q)}{n} - \frac{b(b+2q)}{n} = s\left(1+\frac{q}{n}\right). \tag{3}$$

2.1 The expected evolution of the U-Process

Equations (1)-(3) can be used to have a preliminary intuitive idea on the expected evolution of the U-Process. From (3) it follows that the bias s increases exponentially, in expectation, as long as the number q of undecided nodes is a constant fraction of n (say, $q \geqslant \delta n$, for some positive constant δ). By rewriting (2) in terms of q and s we have that

$$\mathbf{E}\left[Q \mid \mathbf{X}_t = \mathbf{x}\right] = \frac{q^2+2ab}{n} = \frac{2q^2+(n-q)^2-s^2}{2n} \geqslant \frac{n}{3} - \frac{s^2}{2n}, \tag{4}$$

where in the inequality we used the fact that the minimum of $2q^2 + (n-q)^2$ is achieved at $q = \frac{n}{3}$ and its value is $\frac{2}{3}n^2$. From (4) it thus follows that, as long as the magnitude of the bias is smaller than a constant fraction of n (say $s < \frac{2}{3}n$), the expected number of undecided nodes will be larger than a constant fraction of n at the next round (say, $\mathbf{E}\left[Q \mid \mathbf{X}_t = \mathbf{x}\right] \geqslant \frac{n}{9}$).

When the magnitude of the bias s reaches $\frac{2}{3}n$, it is easy to see that the expected number of nodes with the *minority* color decreases exponentially. Indeed, suppose w.l.o.g. that `Beta`

is the minority color and rewrite (1) for B and in terms of b and s. We get

$$\mathbf{E}\left[B \mid \mathbf{X}_t = \mathbf{x}\right] = b\left(\frac{b+2q}{n}\right) = b\left(1 - \frac{2s+3b-n}{n}\right). \tag{5}$$

Hence, when $s > \frac{2}{3}n$ we have that $\mathbf{E}\left[B \mid \mathbf{X}_t = \mathbf{x}\right] \leqslant \frac{2}{3}b$.

The above sketch of the analysis *in expectation* would suggest that the process should end up in a monochromatic configuration within $\mathcal{O}(\log n)$ rounds. Indeed, in Theorem 2 we prove that this is what happens with high probability (w.h.p., from now on) when the process starts from a configuration that already has some bias, namely $s = \Omega(\sqrt{n \log n})$.

When the process starts from a configuration with a smaller bias, the analysis *in expectation* looses its predictive power. As an extreme example, observe that when $a = b = \frac{n}{3}$ the system is "in equilibrium" according to (1)-(3). However, the equilibrium is "unstable" and the symmetry is broken by the *variance* of the process (as long as $s = o(\sqrt{n})$) and by the increasing drift towards majority (as soon as $s > \sqrt{n}$). As mentioned in the Introduction, the analysis of this *symmetry-breaking* phase is the key technical contribution of the paper and it will be described in Section 4. This analysis will show that, starting from any initial configuration, the system reaches a configuration where the magnitude of the bias is $\Omega(\sqrt{n \log n})$ within $\mathcal{O}(\log n)$ rounds, w.h.p.

3 Main results and the digraph of the U-Process' phases

As informally discussed in the introduction, we prove the two following results characterizing the evolution of the U-Dynamics on the synchronous $\mathcal{PULL}$ model in the complete graph.

▶ **Theorem 1** (Consensus). *Let the U-Process start from any configuration in $\mathcal{C}$. Then the process converges to a (valid) monochromatic configuration within $\mathcal{O}(\log n)$ rounds, w.h.p. Furthermore, if the initial configuration has at least one colored node (i.e. $q \leqslant n-1$), then the process converges to a configuration such that $|s| = n$, w.h.p.*

▶ **Theorem 2** (Plurality consensus). *Let γ be any positive constant. Assume that the U-Process starts at any biased configuration such that $|s| \geqslant \gamma\sqrt{n \log n}$ and assume w.l.o.g. the majority color is* `Alpha`*. Then the process converges to the monochromatic configuration with $a = n$ within $\mathcal{O}(\log n)$ rounds, w.h.p. Furthermore, the result is almost tight in a twofold sense: (i) An initial configuration exists, with $|s| = \Omega(\sqrt{n \log n})$, such that the process requires $\Omega(\log n)$ rounds to converge w.h.p. and (ii) there is an initial configuration with $|s| = \Theta(\sqrt{n})$ such that the process converges to the minority color with constant probability.*

Outline of the two proofs. The two theorems above are consequences of our refined analysis[9] of the evolution of the U-Process. The analysis is organized into a set of possible process phases, each of them is defined by specific ranges of parameters q and s. A high-level description of this structure is shown in Fig. 1 where every rectangular region represents a subset of configurations with specific ranges of s and q and it is associated to a specific phase. In details, let γ be any positive constant, then the regions are defined as follows: H_1 is the set of configurations such that $s \leqslant \gamma\sqrt{n \log n}$ and $q \geqslant \frac{1}{2}n$; H_2 is the set of configurations

[9] We remark that our analysis focuses on asymptotic bounds and it does not definitely optimize the corresponding constants: However, using technicalities and loosing readability, all such constants can be largely improved.

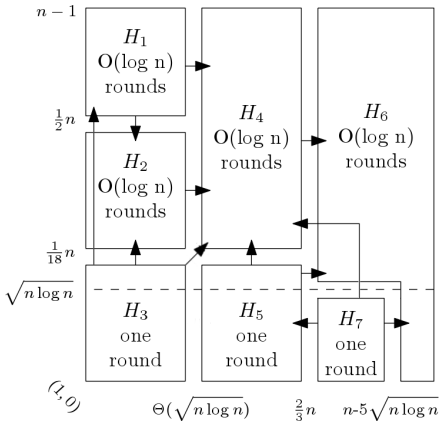


Figure 1 $\{H_1, \ldots, H_7\}$ is the considered partitioning of the configuration space $\mathcal{C}$. On the x axis we represent the bias s, on the y axis the number of undecided nodes q. Missing arrows are transitions that have negligible probabilities.

such that $s \leqslant \gamma\sqrt{n\log n}$ and $\frac{1}{18}n \leqslant q \leqslant \frac{1}{2}n$; H_3 is the set of configurations such that $s \leqslant \gamma\sqrt{n\log n}$ and $q \leqslant \frac{1}{18}n$. H_4 is the set of configurations such that $\gamma\sqrt{n\log n} \leqslant s \leqslant \frac{2}{3}n$ and $q \geqslant \frac{1}{18}n$; H_5 is the set of configurations such that $\gamma\sqrt{n\log n} \leqslant s \leqslant \frac{2}{3}n$ and $q \leqslant \frac{1}{18}n$; H_7 is the set of configurations such that $\frac{2}{3}n \leqslant s \leqslant n - 5\sqrt{n\log n}$ and $q \leqslant \sqrt{n\log n}$. H_6 is the set of configurations such that $s \geqslant \frac{2}{3}n$ minus H_7.

For each region, Fig. 1 specifies our upper bound on the exit time of the corresponding phase, while black arrows represent the phase transitions which may happen with non-negligible probability.

Observe that the scheme highlighted in Fig. 1 can be seen as a directed acyclic graph G having a single sink, H_6, that is reachable from any other region. We remark that, starting from certain configurations, a monochromatic state may be reached via different paths in G. This departs from previous analysis of consensus processes [4, 5, 18] in which the phase transition graph is essentially a path.

We now outline the proofs of the two main results of this paper.

Outline of the Proof of Theorem 2. Consider an initial configuration $\mathbf{x}$ such that $s(\mathbf{x}) \geqslant \gamma\sqrt{n\log n}$, for some positive constant γ, and assume w.l.o.g. that the majority color in $\mathbf{x}$ is `Alpha`. We first show that, if the initial configuration $\mathbf{x}$ is in H_4, then the bias grows exponentially fast and thus the process enters in H_6 within $\mathcal{O}(\log n)$ rounds. Then we prove that, once in H_6, the process ends in the monochromatic configuration where $a = n$ within $\mathcal{O}(\log n)$ rounds, w.h.p. All the other cases "reduce" to the above ones in at most two rounds. Indeed, we show that, starting from any configuration in H_5, the process falls into H_4 or H_6 in one round and that, starting from any configuration in H_7, the process falls into H_4, H_5 or H_6 in one round. As for the tightness of the result stated in the second part of the theorem, we have that the lower bound (Claim (i)) on the convergence time is an immediate consequence of Claim (ii) of Lemma 12, while the second claim, concerning the lower bound on the initial bias, is proved in the full version of the paper [14].

Outline of the Proof of Theorem 1. We first observe that the configuration where all nodes are undecided (i.e. $q = n$) is an absorbing state of the U-Process and thus, for this initial configuration, Theorem 1 trivially holds. In Section 4, we will show that, starting from any *balanced* configuration, i.e. with $s = o(\sqrt{n\log n})$, the U-Process "breaks symmetry"

reaching a configuration $\mathbf{y}$ with $s(\mathbf{y}) = \Omega(\sqrt{n \log n})$ within $\mathcal{O}(\log n)$ rounds, w.h.p. Then, the thesis easily follows by applying Theorem 2 with initial configuration $\mathbf{y}$. As for the symmetry-breaking phase, in Lemma 3 we prove that, if the process starts from a configuration in H_1 or H_3 (see Figure 1), then after $\mathcal{O}(\log n)$ rounds either the bias between the two colors becomes $\Omega(\sqrt{n \log n})$ or the system reaches some configuration in H_2, w.h.p. In Lemma 8 we then prove that, if the process is in a configuration in H_2, then the bias s will become $\Omega(\sqrt{n \log n})$ within $\mathcal{O}(\log n)$ rounds, w.h.p.

4 Symmetry breaking

In this section we show that, starting from any (almost-) balanced configuration, i.e. those with $s = o(\sqrt{n \log n})$, the U-Process "breaks symmetry" reaching a configuration with $s = \Omega(\sqrt{n \log n})$ within $\mathcal{O}(\log n)$ rounds, w.h.p. This part of our analysis is organized as follows.

In Lemma 3 we prove that, if the process starts at a configuration in H_1 or H_3 (see Figure 1), i.e., when the number of undecided nodes is either smaller than $n/18$ or larger than $n/2$, then, after $\mathcal{O}(\log n)$ rounds, either the bias between the two colors already gets magnitude $\Omega(\sqrt{n \log n})$ or the system reaches some configuration in H_2 (i.e., a configuration where the number of undecided nodes is between $n/18$ and $n/2$). In Lemma 8 we then prove that, if the process is in a configuration in H_2, then the bias between the two colors will get magnitude $\Omega(\sqrt{n \log n})$ within $\mathcal{O}(\log n)$ rounds, w.h.p.

▶ **Lemma 3** (Phases H_1 and H_3: Starters)**.**

- *Starting from any configuration $\boldsymbol{x} \in H_3$, the U-Process reaches a configuration $\boldsymbol{X}' \in (H_1 \cup H_2 \cup H_4)$ in one round, w.h.p.*
- *Starting from any configuration $\boldsymbol{x} \in H_1$, the U-Process reaches a configuration $\boldsymbol{X}' \in (H_2 \cup H_4)$ within $\mathcal{O}(\log n)$ rounds, w.h.p.*

If the system lies in a configuration of H_2, we need more complex probabilistic arguments to prove that the bias between the two colors reaches $\Omega(\sqrt{n \log n})$ within $\mathcal{O}(\log n)$ rounds w.h.p. We will make use of the following bound on the hitting time of any Markov chain having suitable drift properties. This result is a variant of Claim 2.9 in [18] that requires a new proof.

▶ **Lemma 4.** *Let $\{X_t\}_{t \in \mathbb{N}}$ be a Markov Chain with finite state space Ω and let $f : \Omega \mapsto [0, n]$ be a function that maps states to integer values. Let c_3 be any positive constant and let $m = c_3 \sqrt{n} \log n$ be a target value. Assume the following properties hold:*

1. *For any positive constant h, a positive constant $c_1 < 1$ exists such that, for any $x \in \Omega$ with $f(x) < m$, it holds that*

$$\mathbf{P}\left(f(X_{t+1}) < h\sqrt{n} | X_t = x\right) < c_1\,,$$

2. *Two positive constants ε, c_2 exist such that, for any $x \in \Omega$ with $f(x) < m$, it holds that*

$$\mathbf{P}\left(f(X_{t+1}) < (1+\varepsilon) f(X_t) | X_t = x\right) < e^{-c_2 f(x)^2/n}\,.$$

Then the process reaches a state x such that $f(x) \geqslant m$ within $\mathcal{O}(\log n)$ rounds, w.h.p.

Proof. We first define a set of hitting times $T = \{\tau(i)\}_{i \in \mathbb{N}}$ where

$$\tau(i) = \inf_{t \in \mathbb{N}} \{t : t > \tau(i-1), f(X_t) \geqslant h\sqrt{n}\}$$

setting $\tau(0) = 0$. By Hypothesis (1), for every $i \in \mathbb{N}$, the expectation of $\tau(i)$ is finite. Then we define the following stochastic process which is a subsequence of $\{X_t\}_{t\in\mathbb{N}}$: $\{R_i\}_{i\in\mathbb{N}} = \{X_{\tau(i)}\}_{i\in\mathbb{N}}$. Observe that $\{R_i\}_{i\in\mathbb{N}}$ is still a Markov Chain. Indeed, let $\{x_1, \dots, x_{i-1}\}$ a set of states in Ω:

$$
\begin{aligned}
&\mathbf{P}\left(R_i = x | R_{i-1} = x_{i-1} \wedge \cdots \wedge R_1 = x_1\right) \\
&= \mathbf{P}\left(X_{\tau(i)} = x | X_{\tau(i-1)} = x_{i-1} \wedge \cdots \wedge X_{\tau(1)} = x_1\right) \\
&= \sum_{t(i)\wedge\cdots\wedge t(0)\in\mathbb{N}} \mathbf{P}\left(X_{t(i)} = x | X_{t(i-1)} = x_{i-1} \wedge \cdots \wedge X_{t(1)} = x_1\right) \\
&\quad \cdot \mathbf{P}\left(\tau(i) = t(i) \wedge \tau(i-1) = t(i-1) \wedge \cdots \wedge \tau(1) = t(1)\right) \\
&= \sum_{t(i)\wedge\cdots\wedge t(0)\in\mathbb{N}} \mathbf{P}\left(X_{t(i)} = x | X_{t(i-1)} = x_{i-1}\right) \\
&\quad \cdot \mathbf{P}\left(\tau(i) = t(i) \wedge \tau(i-1) = t(i-1) \wedge \cdots \wedge \tau(1) = t(1)\right) \\
&= \mathbf{P}\left(X_{\tau(i)} = x | X_{\tau(i-1)} = x_{i-1}\right) = \mathbf{P}\left(R_i = x | R_{i-1} = x_{i-1}\right).
\end{aligned}
$$

By definition the state space of R is $\{x \in \Omega : f(x) \geqslant h\sqrt{n}\}$. Moreover Hypothesis (2) still holds for this new Markov Chain. Indeed:

$$
\begin{aligned}
&\mathbf{P}\left(f(R_{i+1}) < (1+\varepsilon) f(R_i) | R_i = x\right) \\
&= 1 - \mathbf{P}\left(f(R_{i+1}) \geqslant (1+\varepsilon) f(R_i) | R_i = x\right) \\
&= 1 - \mathbf{P}\left(f(X_{\tau(i+1)}) \geqslant (1+\varepsilon) f(X_{\tau(i)}) | X_{\tau(i)} = x\right) \\
&\leqslant 1 - \mathbf{P}\left(f(X_{\tau(i+1)}) \geqslant (1+\varepsilon) f(X_{\tau(i)}) \wedge \tau(i+1) = \tau(i) + 1 | X_{\tau(i)} = x\right) \\
&= 1 - \mathbf{P}\left(f(X_{\tau(i)+1}) \geqslant (1+\varepsilon) f(X_{\tau(i)}) | X_{\tau(i)} = x\right) \\
&= 1 - \mathbf{P}\left(f(X_{t+1} \geqslant (1+\varepsilon) f(X_t) | X_t = x\right) < e^{-c_2 f(x)^2/n}.
\end{aligned}
$$

These two properties are sufficient to study the number of rounds required by the new Markov Chain $\{R_i\}_{i\in\mathbb{N}}$ to reach the target value m. Indeed, by defining the random variable $Z_i = \frac{f(R_i)}{\sqrt{n}}$ and considering the following "potential" function, $Y_i = \exp(\frac{m}{\sqrt{n}} - Z_i)$ we can compute its expectation at the next round as follows. Let us fix any state $x \in \Omega$ such that $h\sqrt{n} \leqslant f(x) < m$ and define $z = \frac{f(x)}{\sqrt{n}}$ and $y = \exp(\frac{m}{\sqrt{n}} - z)$. We get:

$$
\begin{aligned}
\mathbf{E}\left[Y_{i+1} | R_i = x\right] &\leqslant \mathbf{P}\left(f(R_{i+1}) < (1+\varepsilon) f(x)\right) e^{m/\sqrt{n}} \\
&\quad + \mathbf{P}\left(f(R_{i+1}) \geqslant (1+\varepsilon) f(x)\right) e^{m/\sqrt{n} - (1+\varepsilon)z} \\
\text{(from Hypothesis (2))} \quad &\leqslant e^{-c_2 z^2} \cdot e^{m/\sqrt{n}} + 1 \cdot e^{m/\sqrt{n} - (1+\varepsilon)z} \qquad (6) \\
&= e^{m/\sqrt{n} - c_2 z^2} + e^{m/\sqrt{n} - z - \varepsilon z} \\
&= e^{m/\sqrt{n} - z}(e^{z - c_2 z^2} + e^{-\varepsilon z}) \leqslant e^{m/\sqrt{n} - z}(e^{-2} + e^{-2}) \qquad (7) \\
&< \frac{e^{m/\sqrt{n} - z}}{e} < \frac{y}{e},
\end{aligned}
$$

where in (7) we used that z is always at least h and thanks to Hypothesis (1) we can choose a sufficiently large h. By applying the Markov inequality and iterating the above bound, we get:

$$
\mathbf{P}\left(Y_i > 1\right) \leqslant \frac{\mathbf{E}\left[Y_i\right]}{1} \leqslant \frac{\mathbf{E}\left[Y_{i-1}\right]}{e} \leqslant \cdots \leqslant \frac{\mathbf{E}\left[Y_0\right]}{e^{\tau_R}} \leqslant \frac{e^{m/\sqrt{n}}}{e^i}.
$$

We observe that if $Y_i \leqslant 1$ then $R_i \geqslant m$, thus by setting $i = m/\sqrt{n} + \log n = (c_3+1)\log n$, we get:

$$\mathbf{P}\left(R_{(c_3+1)\log n} < m\right) = \mathbf{P}\left(Y_{(c_3+1)\log n} > 1\right) < \frac{1}{n}. \tag{8}$$

Our next goal is to give an upper bound on the hitting time $\tau_{(c_3+1)\log n}$. Note that the event "$\tau_{(c_3+1)\log n} > c_4 \log n$" holds if and only if the number of rounds such that $f(X_t) \geqslant h\sqrt{n}$ (before round $c_4 \log n$) is less than $(c_3+1)\log n$. Thanks to Hypothesis (1), at each round t there is at least probability $1 - c_1$ that $f(X_t) \geqslant h\sqrt{n}$. This implies that, for any positive constant c_4, the probability $\mathbf{P}\left(\tau_{(c_3+1)\log n} > c_4 \log n\right)$ is bounded by the probability that, within $c_4 \log n$ independent Bernoulli trials, we get less than $(c_3+1)\log n$ successes, where the success probability is at least $1 - c_1$. We can thus choose a sufficiently large c_4 and apply the multiplicative form of the Chernoff bound (see e.g. Theorem 1.1 in[21]) and obtain:

$$\mathbf{P}\left(\tau_{(c_3+1)\log n} > c_4 \log n\right) < \frac{1}{n}. \tag{9}$$

We are now ready to prove the Lemma using (8) and (9), indeed:

$$\begin{aligned}
\mathbf{P}\left(\exists t \leqslant c_4 \log n : X_t \geqslant m\right) &> \mathbf{P}\left(R_{(c_3+1)\log n} \geqslant m \wedge \tau_{(c_3+1)\log n} \leqslant c_4 \log n\right) \\
&= 1 - \mathbf{P}\left(R_{(c_3+1)\log n} < m \vee \tau_{(c_3+1)\log n} > c_4 \log n\right) \\
&\geqslant 1 - \mathbf{P}\left(R_{(c_3+1)\log n} < m\right) - \mathbf{P}\left(\tau_{(c_3+1)\log n} > c_4 \log n\right) \\
&> 1 - \frac{2}{n}.
\end{aligned}$$

Hence, choosing a suitable big c_4, we have shown that in $c_4 \log n$ rounds the process reaches the target value m, w.h.p. ◀

The basic idea would be to apply the above lemma to the U-Process with $f(X_t) = |s(X_t)|$ in order to get an upper bound on the number of rounds needed to reach a configuration having bias $\Omega(\sqrt{n \log n})$. To this aim, we first show that, for any configuration in H_2, Properties 1 and 2 in Lemma 4 are satisfied.

▶ **Claim 5.** *Let $\boldsymbol{x} \in \mathcal{C}$ be any configuration such that $\frac{n}{18} \leqslant q(\boldsymbol{x}) \leqslant \frac{n}{2}$ and $|s(\boldsymbol{x})| < c_4\sqrt{n}\log n$ for any positive constant c_4, then it holds:*

1. *for any constant $h > 0$ a constant $c_1 < 1$ exists such that $\mathbf{P}\left(|S| < h\sqrt{n} \mid \boldsymbol{X}_t = \boldsymbol{x}\right) < c_1$,*
2. *two positive constants c_2, ε exist such that $\mathbf{P}\left(|S| \geqslant (1+\varepsilon)s \mid \boldsymbol{X}_t = \boldsymbol{x}\right) \geqslant 1 - e^{-c_2 s^2/n}$.*

It is important to observe that the above claim ensures Properties 1 and 2 of Lemma 4 whenever $\frac{1}{18}n \leqslant q \leqslant \frac{1}{2}n$. Unfortunately, Lemma 4 requires such properties to hold for *any* (almost-)balanced configuration: If $q = n - o(n)$, Property 1 does not hold, while Property 2 is not satisfied if $q = o(n)$. In order to manage this issue, in Subsection 4.1, we define a *pruned* process, a variant of U-Process where it is possible to apply Lemma 4. Then, in Subsection 4.2 we show a coupling between the U-Process and the pruned one.

4.1 The pruned process

The helpful, key point is that, starting from any configuration $\mathbf{x} \in H_2$, the probability that the process goes in one of those "bad" configurations with $q < \frac{1}{18}n$ or $q \geqslant \frac{1}{2}n$ is negligible (see Claim 7). Thus, intuitively speaking, all the configurations *actually visited* by the process

before leaving H_2 do satisfy Lemma 4. In order to make this intuitive argument rigorous, in what follows, we define a suitably *pruned* process by removing from H_2 all the *unwanted* transitions.

Let $\bar{s} \in [n]$ and $\mathbf{z}(\bar{s})$ be the configuration such that $s(\mathbf{z}(\bar{s})) = \bar{s}$ and $q(\mathbf{z}(\bar{s})) = \frac{1}{2}n$. Let $p_{\mathbf{x},\mathbf{y}}$ be the probability of a transition from the configuration $\mathbf{x}$ to the configuration $\mathbf{y}$ in the U-Process. We define a new stochastic process: The U-Pruned-Process. The U-Pruned-Process behaves exactly like the original process but every transition from a configuration $\mathbf{x} \in H_2$ to a configuration $\mathbf{y}$ such that $q(\mathbf{y}) < \frac{1}{18}n$ or $q(\mathbf{y}) > \frac{1}{2}n$ now have probability $p'_{\mathbf{x},\mathbf{y}} = 0$. Moreover, for any $\bar{s} \in [n]$, starting from any configuration $\mathbf{x} \in H_2$, the probability of reaching the configuration $\mathbf{z}(\bar{s})$ is:

$$p'_{\mathbf{x},\mathbf{z}(\bar{s})} = p_{\mathbf{x},\mathbf{z}(\bar{s})} + \sum_{\mathbf{y}:\left(q(\mathbf{y})<\frac{1}{18}n \vee q(\mathbf{y})>\frac{1}{2}n\right) \bigwedge s(\mathbf{y})=\bar{s}} p_{\mathbf{x},\mathbf{y}}.$$

Finally, all the other transition probabilities remain the same.

Observe that, since the U-Pruned-Process is defined in such a way it has exactly the same marginal probability of the original process with respect to the random variable s, then Claim 5 holds for the U-Pruned-Process as well. Thus, we can choose constants h, c_1, c_2, ε such that the two properties of Lemma 4 are satisfied.

▶ **Corollary 6.** *Starting from any configuration $\boldsymbol{x} \in H_2$, the U-Pruned-Process reaches a configuration $\boldsymbol{X}' \in H_4$ within $\mathcal{O}(\log n)$ rounds, w.h.p.*

4.2 Back to the original process

The definition of the U-Pruned-Process suggests a natural coupling between the original process and the pruned one: If the two processes are in different states then they act independently, while, if they are in the same configuration $\mathbf{x}$, they move together unless the U-Process goes in a configuration $\mathbf{y}$ such that $q(\mathbf{y}) < \frac{1}{18}n$ or $q(\mathbf{y}) > \frac{1}{2}n$. In that case the U-Pruned-Process goes in $\mathbf{z}(s(\mathbf{y}))$. Using this coupling, we first show that, if the two processes are in the same configuration, the probability that they get separated is negligible.

▶ **Claim 7.** *For every configuration $\boldsymbol{x} \in H_2$, the probability that the number of undecided nodes in the next round of the U-Process is not between $n/18$ and $n/2$ is*

$$\mathbf{P}\left(q(\boldsymbol{X}_{t+1}) \notin \left[\frac{n}{18}, \frac{n}{2}\right] \mid \boldsymbol{X}_t = \boldsymbol{x}\right) \leqslant e^{-\Theta(n)}.$$

Then, thanks to the above claim, we can show that the H_2 exit time of the pruned procedure stochastically dominates the H_2 exit time of the original process. Thus, using Corollary 6, we get the main result of this section.

▶ **Lemma 8** (Phase H_2)**.** *Starting from any configuration $\boldsymbol{x} \in H_2$, the U-Process reaches a configuration $\boldsymbol{X}' \in H_4$ within $\mathcal{O}(\log n)$ rounds, w.h.p.*

5 Convergence to the majority

In this section we state the key technical lemmas we use to prove our second main result, namely Theorem 2, which essentially establishes that, starting from any sufficiently-biased configuration, the U-Process converges to the monochromatic configuration where all nodes support the initial majority color within $\Theta(\log n)$ rounds, w.h.p.

The proofs of the technical lemmas, as well as the almost-tightness result on the minimal magnitude of the initial bias stated in Theorem 2, can be found in the full version of the paper [14].

Phases H_5 and H_7 (Starters II)

We show that if the process is in a configuration where the number of the undecided nodes is relatively small with respect to the bias, then in the next round the number of the undecided nodes becomes large while the bias does not decrease too much, w.h.p. This essentially implies that if the process starts in H_5 then in the next round the process moves to a configuration belonging to H_4 or H_6 (Lemma 9), while if it starts in H_7 then in the next round it moves to H_4 or H_5 or H_6 (Lemma 10).

▶ **Lemma 9** (Phase H_5). *Starting from any configuration $\boldsymbol{x} \in H_5$ with $a > b$, the U-Process reaches a configuration $\boldsymbol{X}' \in (H_4 \cup H_6)$ with $a > b$ in one round, w.h.p.*

▶ **Lemma 10** (Phase H_7). *Starting from any configuration $\boldsymbol{x} \in H_7$ with $a > b$, the U-Process reaches a configuration $\boldsymbol{X}' \in (H_4 \cup H_5 \cup H_6)$ with $a > b$ in one round, w.h.p.*

Phase H_4 (Age of the undecideds)

We first show that, under some parameter ranges including H_4 (and hence when the number of the undecideds are large enough), the growth of the bias is exponential.

▶ **Claim 11.** *Let γ be any positive constant and $\boldsymbol{x} \in \mathcal{C}$ be any configuration such that $s \geqslant \gamma\sqrt{n \log n}$ and $q \geqslant \frac{1}{18}n$. Then, it holds that $s\,(1 + 1/36) < S < 2s$, w.h.p.*

The above result allows us to prove the following bounds on the time the process requires to reach Phase H_6.

▶ **Lemma 12** (Phase H_4). *Let $\boldsymbol{x} \in H_4$ be a configuration with $a > b$. Then, (i) starting from $\boldsymbol{x}$, the U-Process reaches a configuration $\boldsymbol{X}' \in H_6$ with $a > b$ within $\mathcal{O}(\log n)$ rounds, w.h.p. Moreover, (ii) an initial configuration $\boldsymbol{y} \in H_4$ exists such that the U-Process stays in H_4 for $\Omega(\log n)$ rounds, w.h.p.*

Phase H_6 (Majority takeover)

This is the phase in which, due to the large bias, the nodes converge to the majority color within a logarithmic number of rounds. We first prove that the number of nodes that support the minority color decreases exponentially fast and that the bias is preserved round by round. Then, when $b \leqslant 2\sqrt{n \log n}$, the number of undecided nodes starts to decrease exponentially fast as well. At the very end, when there are only few nodes (i.e., $\mathcal{O}(\sqrt{n \log n})$) that do not support the majority color yet, the minority color disappears in few steps and thus the U-Process converges to majority within $\mathcal{O}(\log n)$ rounds

▶ **Lemma 13** (Phase H_6). *Starting from any configuration $\boldsymbol{x} \in H_6$ with $a > b$, the U-Process ends in the monochromatic configuration where $a = n$ within $\mathcal{O}(\log n)$ rounds, w.h.p.*

6 Conclusions

We provided a full analysis of the U-Dynamics in the parallel $\mathcal{PULL}$ model for the binary case showing it is an efficient self-stabilizing consensus protocol. Besides giving tight bounds on the convergence time, our set of results well-clarifies the main aspects of the process

evolution and the crucial role of the undecided nodes in each phase of this evolution. An interesting open question is that of considering the same process in the multi-color case and to derive bounds on the time required to break symmetry from balanced configurations, as well. Finally, we believe that our analysis can be suitably adapted in order to show that the U-Dynamics efficiently stabilizes to a valid consensus "regime" [10] even in the presence of a dynamic adversary that can change the state of a subset of nodes of size $o(\sqrt{n})$ provided that the initial number of colored nodes is $\Omega(\sqrt{n})$.

References

1 Mohammed Amin Abdullah and Moez Draief. Global majority consensus by local majority polling on graphs of a given degree sequence. *Discrete Applied Mathematics*, 180:1–10, 2015.

2 Dana Angluin, James Aspnes, and David Eisenstat. A Simple Population Protocol for Fast Robust Approximate Majority. *Distributed Computing*, 21(2):87–102, 2008. (Preliminary version in DISC'07).

3 Arta Babaee and Moez Draief. Distributed Multivalued Consensus. In *Computer and Information Sciences III*, pages 271–279. Springer, 2013.

4 Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, and Riccardo Silvestri. Plurality Consensus in the Gossip Model. In *ACM-SIAM SODA'15*, pages 371–390, 2015.

5 Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, Riccardo Silvestri, and Luca Trevisan. Simple dynamics for plurality consensus. In *ACM SPAA'14*, pages 247–256, 2014.

6 Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, and Luca Trevisan. Stabilizing consensus with many opinions. In *ACM-SIAM SODA'16*, pages 620–635, 2016.

7 Florence Bénézit, Patrick Thiran, and Martin Vetterli. Interval consensus: from quantized gossip to voting. In *IEEE ICASSP'09*, pages 3661–3664, 2009.

8 Petra Berenbrink, Andrea Clementi, Peter Kling, Robert Elsässer, Frederik Mallmann-Trenn, and Emanuele Natale. Ignore or Comply? On Breaking Symmetry in Consensus. In *ACM PODC'17*, pages 335–344, 2017. (Tech. Rep. in arXiv:1702.04921 [cs.DC]).

9 Petra Berenbrink, George Giakkoupis, Anne-Marie Kermarrec, and Frederik Mallmann-Trenn. Bounds on the Voter Model in Dynamic Networks. In *ICALP'16*, 2016.

10 Alan A. Berryman and Pavel Kindlmann. *Population Systems: A General Introduction.* Springer, 2008.

11 Luca Cardelli and Attila Csikász-Nagy. The cell cycle switch computes approximate majority. *Scientific Reports*, Vol. 2, 2012.

12 Keren Censor-Hillel, Bernhard Haeupler, Jonathan Kelner, and Petar Maymounkov. Global Computation in a Poorly Connected World: Fast Rumor Spreading with No Dependence on Conductance. In *ACM STOC'12*, pages 961–970, 2012.

13 Bernard Chazelle. Natural Algorithms and Influence Systems. *Commun. ACM*, 55(12):101–110, 2012.

14 Andrea Clementi, Mohsen Ghaffari, Luciano Gualà, Emanuele Natale, Francesco Pasquale, and Giacomo Scornavacca. A Tight Analysis of the Parallel Undecided-State Dynamics with Two Colors. *CoRR*, abs/1707.05135v3, 2018. URL: `https://arxiv.org/abs/1707.05135v3`.

[10] According to the notion of *stabilizing almost-consensus protocol* given in [2, 6].

15 Colin Cooper, Robert Elsässer, and Tomasz Radzik. The Power of Two Choices in Distributed Voting. In *ICALP'14*, pages 435–446, 2014.

16 Colin Cooper, Robert Elsässer, Tomasz Radzik, Nicolas Rivera, and Takeharu Shiraga. Fast Consensus for Voting on General Expander Graphs. In *DISC'15*, pages 248–262, 2015.

17 Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *ACM PODC'87*, pages 1–12, 1987.

18 Benjamin Doerr, Leslie Ann Goldberg, Lorenz Minder, Thomas Sauerwald, and Christian Scheideler. Stabilizing consensus with the power of two choices. In *ACM SPAA'11*, pages 149–158, 2011.

19 David Doty. Timing in chemical reaction networks. In *ACM-SIAM SODA'14*, pages 772–784, 2014.

20 Moez Draief and Milan Vojnović. Convergence speed of binary interval consensus. *SIAM Journal on Control and Optimisation*, 50(3):1087–1109, 2012.

21 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

22 Mohsen Ghaffari and Johannes Lengler. Tight analysis for the 3-majority consensus dynamics. *CoRR*, abs/1705.05583, 2017. `arXiv:1705.05583`.

23 Yehuda Hassin and David Peleg. Distributed probabilistic polling and applications to proportionate agreement. *Information and Computation*, 171(2):248–268, 2001.

24 David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *IEEE FOCS'03*, pages 482–491, 2003.

25 David Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov Chains and Mixing Times*. American Mathematical Society, 2008.

26 Thomas M. Liggett. *Stochastic Interacting Systems: Contact, Voter and Exclusion Processes*. Springer-Verlag, 1999.

27 George B. Mertzios, Sotiris E. Nikoletseas, Christoforos Raptopoulos, and Paul G. Spirakis. Determining Majority in Networks with Local Interactions and Very Small Local Memory. In *ICALP'14*, pages 871–882, 2014.

28 Elchanan Mossel, Joe Neeman, and Omer Tamuz. Majority dynamics and aggregation of information in social networks. *Autonomous Agents and Multi-Agent Systems*, 28(3):408–429, 2014.

29 Saket Navlakha and Ziv Bar-Joseph. Algorithms in nature: the convergence of systems biology and computational thinking. *Mol. Syst. Biol.*, 7, 2011.

30 Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

31 Etienne Perron, Dinkar Vasudevan, and Milan Vojnovic. Using Three States for Binary Consensus on Complete Graphs. In *IEEE INFOCOM'09*, pages 2527–1535, 2009.

Recovering Sparse Graphs

Jakub Gajarský[1]
Technical University Berlin, Berlin, Germany
jakub.gajarsky@tu-berlin.de

Daniel Král'[2]
Mathematics Institute, DIMAP and Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK
d.kral@warwick.ac.uk

Abstract

We construct a fixed parameter algorithm parameterized by d and k that takes as an input a graph G' obtained from a d-degenerate graph G by complementing on at most k arbitrary subsets of the vertex set of G and outputs a graph H such that G and H agree on all but $f(d,k)$ vertices.

Our work is motivated by the first order model checking in graph classes that are first order interpretable in classes of sparse graphs. We derive as a corollary that if $\mathcal{G}$ is a graph class with bounded expansion, then the first order model checking is fixed parameter tractable in the class of all graphs that can obtained from a graph $G \in \mathcal{G}$ by complementing on at most k arbitrary subsets of the vertex set of G; this implies an earlier result that the first order model checking is fixed parameter tractable in graph classes interpretable in classes of graphs with bounded maximum degree.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms, Theory of computation → Logic, Theory of computation → Finite Model Theory

Keywords and phrases model checking, degenerate graphs, interpretations, bounded expansion

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.29

1 Introduction

The work presented in this paper is motivated by the line of research on algorithmic metatheorems, general algorithmic results that guarantee the existence of efficient algorithms for wide classes of problems. The most classical example of such a result is the celebrated theorem of Courcelle [2] asserting that every monadic second order property can be model checked in linear time in every class of graphs with bounded tree-width; further results of this kind can be found in the survey [17]. Specifically, our motivation comes from the first order model checking in sparse graph classes and attempts to extend these results to classes of dense graphs with structural properties close to sparse graph classes.

The two very classical algorithms for the first order model checking in sparse graph classes are the linear time algorithm of Seese [24] for graphs with bounded maximum degree and the linear time algorithm of Frick and Grohe [10] for planar graphs, which can also be adapted

[1] This author's research was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC Consolidator Grant DISTRUCT, grant agreement No 648527). This publication reflects only its authors' view; the European Research Council Executive Agency is not responsible for any use that may be made of the information it contains.

[2] The work of this author was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC Consolidator Grant LADIST, grant agreement No 648509). This publication reflects only its authors' view; the European Research Council Executive Agency is not responsible for any use that may be made of the information it contains.

to an almost linear time algorithm for graphs with locally bounded tree-width. These results were extended to many other classes of sparse graphs, in particular to graphs locally excluding a minor by Dawar, Grohe and Kreutzer [5] and to the very general graph classes with bounded expansion, which were introduced in [18–21], by Dawar and Kreutzer [6] (see [14] for further details) and, independently, by Dvořák, Král' and Thomas [7, 8]. This line of research ultimately culminated with the result of Grohe, Kreutzer and Siebertz [15], who proved that the first order model checking is fixed-parameter tractable in nowhere-dense classes of graphs by giving an almost linear time algorithm for this problem when parameterized by the class and the property.

The results that we have just mentioned concern classes of sparse graphs. While they cannot be extended to all somewhere-dense classes of graphs, see e.g. [8], it is still possible to hope for proving tractability results for dense graphs that possess structural properties making first order model checking feasible. For example, a well-known theorem of Courcelle, Makowsky and Rotics [4] on monadic second order model checking in classes of graphs with bounded clique-width implies that first order model checking is tractable for classes of graphs with bounded clique-width; in relation to the results that we present further, it is interesting to note that graph classes that can be first order interpreted in classes of graphs with bounded clique-width also have bounded clique-width [3, Corollary 7.38]). Another approach is studying graphs defined by geometric means [9, 13, 16]. The approach that we are interested in here lies in considering graph classes derived from sparse graph classes by first order interpretations as in [11, 12]; the definition of a first order interpretation can be found in Section 2.

Specifically, we are motivated by the following very general folklore conjecture.

▶ **Conjecture 1.** *The first order model checking is fixed parameter tractable in $I(\mathcal{G})$ when parameterized by a graph class $\mathcal{G}$ with bounded expansion, a simple first order graph interpretation scheme I and a first order property to be tested.*

The first step towards this conjecture was obtained in [11], where it was shown that Conjecture 1 holds for classes of graphs with bounded maximum degree.

▶ **Theorem 2.** *The first order model checking is fixed parameter tractable in $I(\mathcal{G})$ when parameterized by a class $\mathcal{G}$ of graphs with bounded maximum degree, a simple first order graph interpretation scheme I and a first order property to be tested.*

A combinatorial characterization of classes of graphs interpretable in graph classes of bounded expansion was given in [12]. However, the characterization does not come with an efficient algorithm to compute the corresponding decomposition. So, Conjecture 1 remains open. The approach taken in this paper can be seen as complementary to the one used in [12] since we attempt to directly reverse the effect of the first order interpretation.

To motivate our approach, we sketch the proof of Theorem 2 from [11]. The core of the proof lies in considering first order graph interpretation schemes I where the vertex sets of G and $I(G)$ are the same and constructing an algorithm that recovers a graph H from $I(G)$ such that the graphs G and H have the same vertex set and they agree on most of the edges. We now describe the approach from [11] phrased in the terminology used in this paper.

We start with introducing additional notation. A *pattern* is a graph R that may contain loops and it does not contain a pair of adjacent twins that both have loops, or a pair of non-adjacent twins that neither of them has a loop, i.e., a graph that has no non-trivial induced endomorphism. To make our exposition more transparent, we will further refer to vertices of patterns as to *nodes* and generally denote them by u with different subscripts

and superscripts; vertices of graphs that are not patterns will generally be denoted by v with different subscripts and superscripts. Let G be a graph, R a pattern and $(V_u)_{u \in V(R)}$ a partition of the vertices of G into parts indexed by the nodes of R. The graph G^R is the graph with the same vertex set as G such that if $v, v' \in V(G)$, $v \in V_u$ and $v' \in V_{u'}$, then vv' is an edge in G^R if and only if either vv' is an edge of G and uu' is not an edge of R or vv' is not an edge of G or uu' is an edge of R. Alternatively, we may define the graph G^R to be the graph obtained from G by complementing all edges inside sets V_u for each node u with a loop and between sets V_u and $V_{u'}$ for each edge uu' of R. Note that the graph G^R depends on the chosen partition of the vertex set of G; this partition will always be clear from the context.

A very simple example of the introduced notion is a pattern R that consists of a single node u with a loop. For every graph G, there is only one single class partition of $V(G)$, i.e., $V_u = V(G)$, and G^R is then the complement of G. Similarly, if R has two vertices u and u' and the edges uu (loop) and uu', and the vertex set of a graph G is partitioned into sets V_u and $V_{u'}$, then G^R is obtained from G by complementing all edges inside V_u and all edges between V_u and $V_{u'}$.

We now continue with the exposition of the proof of Theorem 2 from [11]. Simple first order graph interpretation schemes of graphs with bounded maximum degree are very closely linked to patterns as given in the next proposition, which directly follows from Gaifman's theorem [11]. The proposition essentially says that for every integer d and interpretation scheme I, there exist a pattern R and an integer D such that the graph $I(G)$ for any graph G with maximum degree d is equal to H^R for a suitable graph H with maximum degree D; note that R and D depend on I and d only.

▶ **Proposition 3.** *Let $\mathcal{G}_d$ be the class of graphs of maximum degree d and I a simple first order graph interpretation scheme. There exists an integer D and a pattern R such that for every graph $I(G)$ obtained from $G \in \mathcal{G}_d$ there exists a graph $H \in \mathcal{G}_D$ and a partition $(V_u)_{u \in V(R)}$ of the vertex set of H such that the graphs $I(G)$ and H^R are the same.*

This characterization of graphs that can be interpreted in a class of graphs with bounded maximum degree is then combined with the following "recovery" algorithm, which is implicit in [11], to get a proof of Theorem 2. Note the algorithm $\mathcal{A}$ from Theorem 4 has two parameters, one controls the complexity of the structure of a graph and the other controls the complexity of its transformation.

▶ **Theorem 4.** *There exists an algorithm $\mathcal{A}$ that is fixed parameter with respect to an integer parameter D and a pattern R and has the following property: for all D and R, there exist an integer D' and a pattern R' such that the algorithm $\mathcal{A}$ takes as an input a graph G^R, where G is a graph with maximum degree at most D, and outputs a graph H such that G^R and $H^{R'}$ are the same and the maximum degree of H is at most D'.*

One of our main results is an extension of Theorem 4 to classes of d-degenerate graphs. Note that such graph classes include classes with bounded expansion concerned by Conjecture 1. This may look like an innocent extension of Theorem 4 at the first sight. However, the proof of Theorem 4 relies on the fact that the degrees of any two vertices of G^R that are contained in the same part V_u, $u \in V(R)$, differ by at most $2d$, i.e., it is easy to recognize vertices that belong to the same part. This is far from being true in the setting of d-degenerate graphs, which leads to a need for a much finer analysis of the structure of an input graph.

▶ **Theorem 5.** *There exists an algorithm $\mathcal{A}$ that is fixed parameter with respect to integer parameters d and K and has the following property: for all d and K, there exists an integer m such that the algorithm $\mathcal{A}$ takes as an input a graph G^R and integers d and K, where G is a d-degenerate graph and R is a K-node pattern (both unknown to $\mathcal{A}$), and outputs a graph H such that G and H agree on all but at most m vertices. In particular, the graph H is $(d+m)$-degenerate.*

We next present a corollary of Theorem 5, which we believe to be of independent interest. First observe that complementing edges between two subsets V and V' of the vertex set of G is equivalent to complementing on the following three subsets of vertex set: $V \cup V'$, V and V'. Hence, the graph G^R is obtained from G by complementing on at most $K + \binom{K}{2}$ subsets of vertices of G, where K is the number of nodes of R. In the other direction, if a graph H is obtained from G by complementing on at most k subsets of vertices, there exists a pattern R with at most 2^k nodes such that $H = G^R$. Hence, Theorem 5 implies the following.

▶ **Corollary 6.** *There exists an FPT algorithm $\mathcal{A}$ with the following property: for every integer d and an integer k, there exists an integer m such that the algorithm $\mathcal{A}$ takes as an input a graph G' obtained from a d-degenerate graph G by complementing on at most k subsets of the vertex set of G and outputs a graph H such that G and H agree on all but at most m vertices.*

In relation to the first order model checking, Corollary 6 yields the following theorem, which we prove in Section 4.

▶ **Theorem 7.** *Let $\mathcal{G}$ be a graph class with bounded expansion and let $\mathcal{G}^k$ be the class containing all graphs that can obtained from a graph $G \in \mathcal{G}$ by complementing on at most k subsets of the vertex set of G. For every k, the first order model checking is fixed parameter tractable on $\mathcal{G}^k$.*

Observe that Proposition 3 implies the following: if $\mathcal{G}$ is a class of graphs with bounded maximum degree and I is a simple first order graph interpretation scheme, then $I(\mathcal{G}) \subseteq \mathcal{G}_D^k$ for some integers D and k, where $\mathcal{G}_D$ is the class of all graphs with maximum degree at most D. Hence, Theorem 7 gives an alternative proof of Theorem 2. On the other hand, since Proposition 3 does not hold in the setting of graph classes with bounded expansion, Theorems 5 and 7 do not yield an analogous result in this more general setting, which is concerned by Conjecture 1; we discuss further details in Section 5.

2 Preliminaries

In this section, we briefly introduce the notation used throughout the paper, and present the concepts that we need further.

Graphs considered in this paper are simple, i.e., they do not contain loops or parallel edges unless stated otherwise. If G is a graph, then $V(G)$ denotes the set of its vertices. The *neighborhood* of a vertex v in a graph G, denoted by $N_G(v)$, is the set of all vertices adjacent to v. The *degree* of a vertex v of a graph G is the size of its neighborhood, and the *relative degree* of v with respect to a subset $X \subseteq V(G)$ is the number of the neighbors of v in X. If G is a graph and W a subset of its vertices, then the subgraph of G *induced* by W, denoted by $G[W]$, is the subgraph of G with the vertex set W such that two vertices are adjacent in $G[W]$ if and only if they are adjacent in G. Finally, a graph G is *d-degenerate*, if its vertices can be ordered in such a way that each vertex has at most d of its neighbors preceding it.

Let G be a graph. Two vertices v and v' of G are *twins* if every vertex w different from v and v' is adjacent to either both v and v' or none of them. The binary relation of "being a twin" on $V(G)$ is an equivalence relation; we will call the equivalence classes of this relation *twin-classes*. Note that each twin-class induces either a complete subgraph or an empty subgraph of G.

A graph G' is an *r-shallow minor* of a graph G if it can be obtained from a subgraph of G by contracting vertex-disjoint subgraphs of radii at most r (and removing arising loops and parallel edges). We say that a graph class $\mathcal{G}$ has *bounded expansion* if $\mathcal{G}$ is monotone, i.e., closed under taking subgraphs, and there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that the average degree of every r-shallow minor of any graph from G is at most $f(r)$. As we have already mentioned, examples of classes of graphs with bounded expansion are classes of graphs with bounded maximum degree and minor-closed classes of graphs. The latter include classes of graphs with bounded tree-width or graphs embeddable in a fixed surface.

If G is a graph, then a K-apex of G is a graph obtained by at adding at most K vertices to G and joining them to the remaining vertices and between themselves arbitrarily. The next proposition easily follows from the basic results on classes of graphs with bounded expansion; see e.g. [22, Chapter 5] for further details.

▶ **Proposition 8.** *Let $\mathcal{G}$ be a class of graph with bounded expansion, and let K be a positive integer. The class of graphs formed by K-apices of graphs from $\mathcal{G}$ has bounded expansion.*

Finally, a *simple first order interpretation scheme* I consists of a pair of formulas $\psi_V(x)$ and $\psi_E(x, y)$. If G is a graph, then the graph $I(G)$ has vertex set equal to the set $\{v \in V(G) \mid G \models \psi_V(x)\}$, i.e., it is the subset of vertices x of G such that $\psi_V(x)$ holds, and two vertices u and v of $I(G)$ are adjacent iff $G \models \psi_E(u, v) \vee \psi_E(v, u)$.

3 Recovering degenerate graphs

This section is devoted to the proof of Theorem 5, one of our two main results. We need to start with introducing additional notation that will be used in our analysis of complemented graphs. Let G be a graph. Two subsets X and Y of the vertex set $V(G)$ are *k-similar* if their symmetric difference is at most k, i.e., $|X \triangle Y| \leq k$. We say that two vertices of G are *k-similar* if their neighborhoods are k-similar, and we define the *k-similarity graph* of G to be the graph with the vertex set $V(G)$ where two vertices are adjacent if they are k-similar.

Further fix a pattern R and a partition $(V_u)_{u \in V(R)}$ of $V(G)$. If u is a node of R, then the *u-perfect* set is the union of the sets $V_{u'}$ where the union is taken over all neighbors u' of u in R. Note that the u-perfect set includes V_u iff u has a loop. A subset X of the vertex set of G is *(u, k)-perfect* if it is k-similar to the u-perfect set, and a vertex of G is *(u, k)-perfect* if its neighbors in G^R form a *(u, k)-perfect* set. In particular, when saying that a vertex of G is *(u, k)-perfect*, this always concerns its neighborhood in G^R or in the induced subgraph of G^R.

Our goal is to approximately recover graph G from G^R given the size K of R and assuming that G is d-degenerate. We achieve this by finding a partition of $V(G^R)$ that approximates the partition of $(V_u)_{u \in V(R)}$ of $V(G)$. To find the approximate partition, we use (u, k)-perfect vertices introduced above: if we identify a (u, k)-perfect vertex for each class V_u of $(V_u)_{u \in V(R)}$, then the structure of the neighborhoods of these vertices leads to a good approximation of the partition $(V_u)_{u \in V(R)}$. The structural lemmas presented in the next subsection lead to a simple condition (Lemma 11) that allows us to find a (u, C)-perfect vertex in the input graph, where the constant C depends on d and K only. The presented structural results are then be used to design Algorithm 1, which outputs an approximation of the graph G.

3.1 Structural results

In this subsection, we present structural results on complemented graphs. These results will be used in the next subsection to analyze our algorithm. We start with observing that most vertices of each substantially large part are almost perfect.

▶ **Lemma 9.** *Let R be a K-node pattern, G a d-degenerate graph with a vertex partition $(V_u)_{u\in V(R)}$, and M the maximum size of a part V_u, $u \in V(R)$. If a part V_u, $u \in V(R)$, contains at least $\frac{M}{4K}$ vertices, then it contains at least $\left(1-\frac{1}{10K}\right)|V_u|$ vertices that are $(u, 80dK^3)$-perfect.*

Proof. Fix a node u such that the size of the part V_u is at least $\frac{M}{4K}$, and observe that a vertex v of V_u is $(u, 80dK^3)$-perfect if and only if its degree in G is at most $80dK^3$. Hence, we need to show that at least $\left(1-\frac{1}{10K}\right)|V_u|$ vertices of V_u have degree at most $80dK^3$.

Suppose that more than $\frac{1}{10K}|V_u|$ vertices of V_u have degree strictly larger than $80dK^3$. This implies that the sum of the degrees of the vertices of V_u is strictly larger than

$$8dK^2|V_u| \geq 2dKM\ .$$

This is impossible since G contains at most $dn \leq dKM$ edges in total and thus the sum of the degrees of all vertices of G is at most $2dKM$. The statement of the lemma now follows. ◀

The next lemma shows that almost all vertices with similar neighborhoods must belong to the same part.

▶ **Lemma 10.** *Let R be a K-node pattern and G a d-degenerate graph with a vertex partition $(V_u)_{u\in V(R)}$ such that each V_u contains at least $330dK^3$ vertices. For every $W \subseteq V(G)$, there exists a node $u \in V(R)$ such that all but at most $330dK^4$ vertices with their neighborhoods $(160dK^3)$-similar to W in G^R belong to V_u.*

Proof. Suppose that the statement is false and fix a set W that violates the statement. This implies that there are two different nodes u and u' such that each of the sets V_u and $V_{u'}$ contains at least $330dK^3$ vertices with $(160dK^3)$-similar to W in G^R. Indeed, take a node $u \in V(R)$ such that V_u contains the largest number of vertices with their neighborhoods $(160dK^3)$-similar to W in G^R; note that V_u contains at least $330dK^3$ such vertices (otherwise, any node u would satisfy the statement of the lemma since there would be at most $330dK^4$ such vertices in total). Since the set W violates the statement, there are at least $330dK^4$ vertices with their neighborhoods $(160dK^3)$-similar to W in G^R that do not belong to V_u. This implies that there exists a node $u' \in V(R)$ such that $V_{u'}$ also contains at least $330dK^3$ such vertices.

To simplify our notation, fix n to be $330dK^3$. Choose an n-vertex subset A of V_u such that their neighborhoods are $(160dK^3)$-similar to W and an n-vertex subset A' of $V_{u'}$ such that their neighborhoods are $(160dK^3)$-similar to W. Observe that any two vertices in $A \cup A'$ are $(320dK^3)$-similar.

We next distinguish three cases based on whether the nodes u and u' have loops in R and whether they are adjacent in R.

- **At least one of the two nodes, say u, has a loop, and R does not contain the edge uu'.**
  The subgraph $G[A \cup A']$ contains at most $2dn$ edges, which yields that the sum of the degrees of the vertices of $G[A \cup A']$ is at most $4dn$. We next compare relative degrees of the vertices of $A \cup A'$ with respect to A in G^R. Since the neighbors of the vertices of A' in A are the same in $G[A \cup A']$ and in $G^R[A \cup A']$, the sum of the relative degrees of

the vertices of A' with respect to A is at most $4dn$. On the other hand, the sum of the relative degrees of the vertices of A in $G^R[A \cup A']$ is at least $n(n-1) - 4dn$. Since any two vertices in $A \cup A'$ are $(320dK^3)$-similar in G^R and thus in $G^R[A \cup A']$, their relative degrees in G^R with respect to A differ by at most $320dK^3$. Consequently, the sums of the relative degrees of the vertices of A and those of A' with respect to A in G^R can differ by at most $320dK^3n$. However, the difference of these two sums is at least

$$n(n-1) - 8dn = n(n-1-8d) \geq n(330dK^3 - 1 - 8d) \geq 321dK^3n > 320dK^3n \,.$$

- **At least one of the two nodes, say u, does not have a loop, and R contains the edge uu'.**
  An analogous argument to that used in the first case yields that the sum of the relative degrees of the vertices of A with respect to A in G^R is at most $4dn$ and the sum of the relative degrees of the vertices of A' with respect to A in G^R is at least $n^2 - 4dn$. Consequently, the difference of these two sums is at least $n^2 - 8dn > 320dK^3n$ while it cannot exceed $320dK^3n$.
- **The nodes u and u' either both have loops and are adjacent or both do not have a loop and are non-adjacent in R.**
  Since R is a pattern, there must exist a node u'', which is different from u and u', such that either uu'' is not an edge and $u'u''$ is an edge, or vice versa. By symmetry, we can assume the former to be the case. Let A'' be a set of n vertices contained in $V_{u''}$. The number of edges between A and A'' in G is at most $2dn$. Hence, the sum of the relative degrees of the vertices of A with respect to A'' is at most $2dn$ both in G and in G^R. On the other hand, the sum of the relative degrees of the vertices of A' with respect to A'' is at most $2dn$ in G, and thus at least $n^2 - 2dn$ in G^R. Since any two vertices of $A \cup A'$ are $(320dK^3)$-similar, their relative degrees with respect to A'' in G^R can differ by at most $320dK^3$. Consequently, the sums of the relative degrees of the vertices of A and A' can differ by at most $320dK^3n$. However, the difference of the two sums is at least $n^2 - 4dn > 320dK^3n$.

In each of the three cases, we have obtained a contradiction, which concludes the proof of the lemma. ◀

To prove the next lemma, we need to introduce some additional notation. Let G be a graph, R a pattern, $(V_u)_{u \in V(R)}$ a partition of $V(G)$, and U a subset of the nodes of R. The graph $R \setminus U$ need not be a pattern but there is a unique pattern to that $R \setminus U$ has an induced homomorphism. This pattern can be obtained as follows. Let R' be $R \setminus U$. As long as R' contains either two adjacent twins u and u' that both have loops or two non-adjacent twins u and u' that none of them has a loop, identify the nodes u and u' and merge the parts V_u and $V_{u'}$. The resulting pattern R_0 is called the *reduction* of $R \setminus U$; the reduction R_0 is uniquely determined by the pattern R and the set U. If W is the union of V_u with $u \notin U$, then the new parts V_u indexed by $u \in V(R_0)$ form a partition of the vertex set $G[W]$. This partition is called the *reduced partition* and it is easy to observe that the graphs $G^R[W]$ and $G[W]^{R_0}$ are the same.

▶ **Lemma 11.** *Let R be a K-node pattern and G a d-degenerate graph with a vertex partition $(V_u)_{u \in V(R)}$. If G has at least $1100dK^5$ vertices, then the vertex of the maximum degree in the $(160dK^3)$-similarity graph of G^R is $(u, 570dK^4)$-perfect for some $u \in V(R)$.*

Proof. Let u_M be the node of R such that V_{u_M} is the largest part of the partition $(V_u)_{u \in V(R)}$ and let M be its size, i.e., $M = |V_{u_M}|$. Observe that $M \geq 1100dK^4$. By Lemma 9, V_{u_M} contains at least $\left(1 - \frac{1}{10K}\right) M \geq \frac{9M}{10}$ vertices that are $(u_M, 80dK^3)$-perfect. All these vertices

are mutually adjacent in the $(160dK^3)$-similarity graph of G^R, which implies that the maximum degree of the $(160dK^3)$-similarity graph of G^R is at least $9M/10-1$. Let w be the vertex of the maximum degree in the $(160dK^3)$-similarity graph of G^R, W the neighborhood of w in G^R, and W_s the neighborhood of w in the $(160dK^3)$-similarity graph. Note that $|W_s| \geq 9M/10-1$ and each vertex of W_s is $(160dK^3)$-similar to w in G^R.

Let U' be the set of the nodes $u \in V(R)$ such that $|V_u| \leq 330dK^3$, and let V' be the union of the parts V_u with $u \in U'$. Observe that $|V'| \leq 330dK^4$. Let R_0 be the reduction of $R \setminus U'$, and let G_0 be the graph $G \setminus V'$ with the reduced partition $V_{0,u}$, $u \in V(R_0)$. Observe that $G_0^{R_0} = G^R \setminus V'$ and each part $V_{0,u}$, $u \in V(R_0)$, has at least $330dK^3$ vertices. Further, let $W_0 = W \setminus V'$, and note that W_0 is the neighborhood of w in $G_0^{R_0}$ and that each vertex of $W_s \setminus V'$ is $(160dK^3)$-similar to w in $G_0^{R_0}$.

We now apply Lemma 10 to the graph G_0 with the pattern R_0 and the set W_0. The lemma implies that there exists a node u_0 of R_0 such that there are at most $330dK^4$ vertices outside V_{0,u_0} with their neighborhood $(160dK^3)$-similar to W_0 in $G_0^{R_0}$. Hence, the set $W_s \subseteq V(G)$ contains at most $660dK^4$ vertices that are not contained in V_{0,u_0}: all such vertices are contained in V' or are $(160dK^3)$-similar to w in $G_0^{R_0}$. It follows that the part V_{0,u_0} contains at least $9M/10-1-660dK^4 \geq 9M/10-661dK^4 \geq M/4$ vertices of W_s. In particular, the part V_{0,u_0} contains at least $M/4$ vertices in total.

By Lemma 9, the part V_{0,u_0} contains at least $\left(1-\frac{1}{10K}\right)|V_{0,u_0}|$ vertices that are $(u_0, 80dK^3)$-perfect with respect to the graph G_0 and the pattern R_0, i.e., there are at most $\frac{|V_{0,u_0}|}{10K} \leq M/10$ vertices of V_{0,u_0} that are not $(u_0, 80dK^3)$-perfect. Hence, there is a vertex v that is contained in $W_s \cap V_{0,u_0}$ and that is $(u_0, 80dK^3)$-perfect with respect to the graph G_0 and the pattern R_0.

Since the vertex v is $(u_0, 80dK^3)$-perfect with respect to the graph G_0 and the pattern R_0, there exists a node $u \in V(R)$ such that the vertex v is $(u, 80dK^3+|V'|)$-perfect with respect to the graph G and the pattern R, i.e., v is $(u, 80dK^3+330dK^4)$-perfect. Since the vertex v is contained in W_s, i.e., it is a neighbor of w in the $(160dK^3)$-similarity graph, we get that the vertex w is $(u, 240dK^3+330dK^4)$-perfect. Since $240dK^3+330dK^4 \leq 570dK^4$, the lemma now follows. ◀

3.2 Algorithm

We are now ready to present an algorithm that can be used to recover the original d-degenerate graph G from the graph G^R where R is an a priori unknown K-pattern. The algorithm is given as Algorithm 1. The algorithm takes the graph G^R as an input and outputs a graph F that differs from the perfect blow-up E^R of the pattern R only on constantly many vertices, where E is the graph with the vertex set $V(G)$ and no edges. Algorithm 1 is analyzed in the next lemma.

▶ **Lemma 12.** *Let R be a K-node pattern and G a d-degenerate graph with a vertex partition $(V_u)_{u \in V(R)}$. Suppose that Algorithm 1 is applied for $H = G^R$, d and K, and the algorithm outputs a graph F. There exists a subset U of at most $4000dK^6$ vertices of H such that the graph $F \setminus U$ and $E^R \setminus U$ are the same, where E is the empty graph with the vertex set $V(H)$.*

Proof. Let W_i be the set W at the point when the set S_i is fixed by Algorithm 1, and let k be the final value of this variables at the end of the algorithm. Further let W_0 be the set W at the end of the algorithm. By Lemma 11, the set S_i is $(u_i, 570dK^4)$-perfect in $H[W_i]$ for some $u_i \in V(R)$. Note that the set S_i is $(u_i, 570dK^4)$-perfect in $H[W_j]$ for every $j = i+1, \ldots, k$, since this property cannot be affected by deleting vertices. At the point when the set S_i

Algorithm 1: Algorithm producing an approximation of the perfect blow-up of an unknown K-node pattern.

Input: a graph H, integers d and K
Output: a graph F on the vertex set $V(H)$
$W := V(H)$;
F :=empty graph on the vertex set $V(H)$;
$k := 0$;
$\mathcal{S} := \emptyset$;
while $|W| \geq 1100dK^5$ **do**
  if $\exists\, v \in W$ *s.t.* $N_{H[W]}(v)$ *is* $(1140dK^4)$*-similar to a set* $S_i \cap W$, $S_i \in \mathcal{S}$ **then**
    $W := W \setminus \{v\}$;
    join v in F to all the vertices of $S_i \cap W$;
  else
    v :=max. degree vertex in the $(160dK^3)$-similarity graph of $H[W]$;
    $k := k + 1$;
    S_k :=the neighbors of v in $H[W]$;
    add S_k to $\mathcal{S}$;
output F.

was fixed, the set S_i was not $(1140dK^4)$-similar to any of the sets $S_1 \cap W_i, \ldots, S_{i-1} \cap W_i$. It follows that the nodes $u_1, \ldots, u_k$ are mutually distinct, which implies $k \leq K$.

Let T_i be the set of at most $570dK^4$ vertices of $H[W_i]$ such that S_i is u_i-perfect in $H[W_i \setminus T_i]$, and let $T = T_1 \cup \cdots \cup T_k$. Further, for each node $u \in V(R)$, let V'_u be the last $1143dK^4$ vertices of $V_u \setminus T$ removed by Algorithm 1 from the set W if such vertices exist; otherwise, let $V'_u = V_u \setminus (T \cup W_0)$. Note that $|V'_u| \leq 1143dK^4$ in either of the cases.

Consider the point when the algorithm removes a vertex $v \in V_u$ from the set W because the neighborhood of v is $(1140dK^4)$-similar to the set $S_i \cap W$, where W is the value of the variable at the time of the removal of v. We say that the vertex v is u'*-erroneous* for $u' \in V(R)$ if at least one of the vertices of $V_{u'} \setminus (V'_{u'} \cup T \cup W_0)$ has not yet been removed from W and

- either uu' is an edge of R but $V'_{u'}$ and S_i are disjoint, or
- uu' is not an edge of R but $V'_{u'}$ is a subset of S_i.

Note that it can be the case that the nodes u and u' in the above definition coincide, and a vertex v can be u'-erroneous for several choices of u'. Also note that if v is u'-erroneous, then $V_{u'} \setminus (V'_{u'} \cup T \cup W_0) \neq \emptyset$, which implies that $|V'_{u'}| = 1143dK^4$. Let $V_{u,u'}$ be the set of vertices of $V_u \setminus (V'_u \cup T)$ that are u'-erroneous.

The set U will contain the following vertices:

- at most $1100dK^5$ vertices contained in W_0,
- at most $k \cdot 570dK^4 \leq 570dK^5$ vertices contained in T,
- at most $K \cdot 1143dK^4 \leq 1143dK^5$ vertices contained in the set V'_u, $u \in V(R)$, and
- the vertices of all sets $V_{u,u'}$, $u, u' \in V(R)$.

We next show that each of the sets $V_{u,u'}$ contains at most $1143dK^4$ vertices, which would imply that the size of U does not exceed $4000dK^6$.

Set $n = 1143dK^4$ to simplify the notation, and suppose that there exists a set $V_{u,u'}$ containing more than n vertices for some $u, u' \in V(R)$ (possibly $u = u'$). Let X be a subset of $V_{u,u'}$ containing exactly n vertices. Note that that if $u = u'$, the sets X and $V'_{u'}$ are disjoint because all vertices of $V'_{u'}$ are removed from W after those of X. We first consider the case

that uu' is not an edge of R, which includes the case that $u = u'$ and u does not have a loop. Since the vertices of X are u'-erroneous, the set $V'_{u'}$ contains n vertices and all vertices of $V'_{u'}$ are removed from W later than the vertices of X, the d-degeneracy of G implies that the number of edges between X and $V'_{u'}$ in G is at most $2dn$. When a vertex $v \in X$ is removed from W by Algorithm 1, it is adjacent to at least $|V'_{u'}| - 1140dK^4 \geq 3dK^4$ vertices of $V'_{u'}$ in $H = G^R$ since the neighborhood of v is $(1140dK^4)$-similar to S_i and $V'_{u'} \subseteq S_i$. Hence, the number of edges between X and $V'_{u'}$ in $H = G^R$ is at least $3dK^4 n \geq 3dn$. However, the edges between the vertices of X and those of $V'_{u'}$ are the same in G and G^R, which is impossible.

The other case that we need to consider is that when uu' is an edge of R; this case also includes the case that $u = u'$ and u has a loop. The arguments are analogous to the first case but we include them for completeness. We again observe that the number of edges between X and $V'_{u'}$ in G is at most $2dn$. When a vertex $v \in X$ is removed from W, it is adjacent to at most $1140dK^4$ vertices of $V'_{u'}$ in $H = G^R$ since its neighborhood is $(1140dK^4)$-similar to S_i and the sets S_i and $V'_{u'}$ are disjoint. It follows that each vertex $v \in X$ is adjacent to at least $|V'_{u'}| - 1140dK^4 \geq 3dK^4$ vertices of $V'_{u'}$ in G. This implies that the number of edges between X and $V'_{u'}$ in G is at least $3dK^4 n \geq 3dn$, which is again impossible.

To complete the proof of the lemma, we need to show that the graphs $F \setminus U$ and $E^R \setminus U$ are the same. Let v and v' be two vertices of $V(H) \setminus U$ such that $v \in V_u$ and $v' \in V_{u'}$. By symmetry, we can assume that v is removed before v'. Suppose that the vertex v was removed by Algorithm 1 because the neighborhood of v in $H[W]$ was $(1140dK^4)$-similar to a set S_i where W is the value of the set at the time of the removal of v from W. Since the vertex v' does not belong to U, it is not contained in $V'_{u'} \cup W_0 \cup T$, which implies that $V'_{u'} \subseteq (V_{u'} \cap W) \setminus T$. Further, since the set S_i is u_i-perfect in $H[W \setminus T]$, the set S_i either contains $(V_{u'} \cap W) \setminus T$ or is disjoint from $(V_{u'} \cap W) \setminus T$. Since v is not u'-erroneous, the former happens if and only if uu' is an edge in R, and the latter happens otherwise. Hence, the vertices v and v' are joined by an edge in F if and only if uu' is an edge of R. ◀

Lemma 12 yields the proof of Theorem 5 as follows.

Proof of Theorem 5. Fix d and K, and set $m = 4000dK^6$. Let G_0 be the input graph, and suppose that G is the d-degenerate graph and R is the K-node pattern such that $G_0 = G^R$. Note that both G and R are not given to the algorithm $\mathcal{A}$.

The algorithm $\mathcal{A}$ applies Algorithm 1 to the graph G_0 and integers d and K, and Algorithm 1 outputs a graph F. By Lemma 12, the graphs E^R and F agree on all but at most m vertices, where E is the empty graph on the same vertex set as G_0. The algorithm $\mathcal{A}$ then outputs the graph $G_0 \triangle F$, i.e., the graph with the same vertex set as G_0 and with the edge set that is the symmetric difference of the edge sets of G_0 and F. Observe that the graph $G = G^R \triangle E^R$ and the output graph $G_0 \triangle F = G^R \triangle F$ differ exactly where the graphs E^R and F differ. It follows that the output graph $G_0 \triangle F$ and the graph G agree on all but at most m vertices, which implies that the output graph $G_0 \triangle F$ is $(d + m)$-degenerate. ◀

4 FO model checking

In this section, we prove Theorem 7, which is our second main result, and also discuss first order model checking in graphs obtained by complementing parts of degenerate graphs. We start with proving Theorem 7.

Proof of Theorem 7. Fix a graph class $\mathcal{G}$ with bounded expansion and an integer k, and set $K = 2^k$. Since the graph class $\mathcal{G}$ has bounded expansion, there exists an integer d such that every graph in $\mathcal{G}$ is d-degenerate. Set $m = 4000dK^6$ and let $\mathcal{H}$ be the graph class that

contain all m-apices of subgraphs of graphs contained in $\mathcal{G}$. By Proposition 8, the graph class $\mathcal{H}$ has bounded expansion.

Let G' be a graph obtained from a graph $G \in \mathcal{G}$ by complementing on at most k subsets of the vertex set of G, and let V be the common vertex set of G and G'. Note that there exists a K-node pattern R (which can be chosen independently of G and G' but this fact is not needed in our proof) and a partition $(V_u)_{u \in V(R)}$ of the vertex set V such that $G' = G^R$. Apply Algorithm 1 to G', d and K, and let F be the output graph. Since the graphs F and E^R, where E is the empty graph on the vertex set V, coincide on all but at most m vertices by Lemma 12, there exists a $(K+m)$-node pattern R_F such that $F = E^{R_F}$ for a suitable partition $(V'_u)_{u \in V(R_F)}$ of the vertex set V. Moreover, the pattern R_F and the partition $(V'_u)_{u \in V(R_F)}$ can be efficiently constructed: the at most $K+m$ twin-classes of the graph F form the partition $(V'_u)_{u \in V(R_F)}$ and the partition into twin-classes uniquely determine the pattern.

Let H be the graph with the vertex set V and the edge set being the symmetric difference of the edge sets of G' and F. Observe that $H^{R_F} = G'$. By Lemma 12, the graphs G and H agree on all but at most m vertices, which implies that the graph H belongs to the class $\mathcal{H}$. The application of the pattern R_F to H can be simulated by viewing the partition $(V'_u)_{u \in V(R_F)}$ as a vertex $(K+m)$-coloring and encoding the application of the pattern R_F by a first order formula. In particular, there exists a simple first order graph interpretation scheme I of $(K+m)$-vertex colored graphs such that $I(H) = G'$. Since there are only finitely many choices of R_F (because the number of nodes of R_F is bounded) and it is possible to use disjoint sets of colors to encode applications of different patterns R_F, there exists such an interpretation scheme I that is universal for all patterns R_F. The fixed parameter tractability of the first order model checking in $\mathcal{G}^k$ is now implied by the fixed parameter tractability of the first order model checking in graph classes with bounded expansion that contain graphs vertex-colored by a bounded number of colors, which directly follows from the results of [6–8]. ◀

The first order model checking in d-degenerate graphs is hard from the point of fixed parameter tractability, however, many parameterized problems that are hard for general graphs become fixed parameter tractable when restricted to d-degenerate graphs. Two prominent examples of such problems are the k-clique problem, which asks whether the input graph contains a complete subgraph with k vertices, and the k-independent set problem, which asks whether the input graph contains k independent vertices. Both these problems are fixed parameter tractable when parameterized by d and k.

To explore hopes of extending the fixed parameter tractability results for d-degenerate graphs to classes of graphs obtained by complementing d-degenerate graphs, we provide a brief analysis of the fixed parameter tractability of the k-clique problem in graphs obtained from d-degenerate graphs by applying patterns In the rest of this section, $\mathcal{G}_d$ denotes the class of d-degenerate graphs and $\mathcal{G}_d^R$ for a pattern R will be the class of all graphs that can be obtained from a graph $G \in \mathcal{G}_d$ by applying the pattern R, i.e., the class of all graphs G^R for $G \in \mathcal{G}_d$. We start with considering the parameterization by both R and k, where the problem turns out to be tractable for $d = 1$ and hard for $d \geq 2$ as given in the following two propositions.

▶ **Proposition 13.** *The k-clique problem in the class $\mathcal{G}_1^R$ is fixed parameter tractable when parameterized by a pattern R and an integer k.*

Proof. The class $\mathcal{G}_1$ of 1-degenerate graphs is the class of all forests. Recall that a rank-width of a graph G is defined as the minimum r such that there exists a tree T with leaves one-to-one

corresponding to the vertices of G such that each edge e of T determines a vertex cut (A, B) of G (A and B are the vertices assigned to the leaves of the two components of $T \setminus e$) such that the adjacency matrix of the cut (A, B) has rank at most r. It is not hard to see that each forest has rank-width at most one. Next observe that if the adjacency matrix of a vertex cut (A, B) in a graph G has rank r, then the adjacency matrix of the cut (A, B) in G^R has rank at most $r + K$. Consequently, if G is a graph with rank-width r and R is a K-node pattern, then the rank-width of G^R is at most $r + K$. We conclude that all graphs contained in the class $\mathcal{G}_1^R$ have bounded rank-width, which implies that all graphs contained in the class $\mathcal{G}_1^R$ have bounded clique-width [23]. Since monadic second order model checking is fixed parameter tractable in classes of graphs with bounded clique-width [4], the statement of the proposition follows. ◀

▶ **Proposition 14.** *The k-clique problem in the class $\mathcal{G}_2^R$ is $W[1]$-hard when parameterized by a pattern R and an integer k.*

Proof. We present a reduction from the multicolored k-clique problem, which is a well-known $W[1]$-hard problem. The multicolored k-clique problem asks whether a given k-partite graph contains a clique of order k. Let G be an arbitrary k-partite graph, let $V_1, \ldots, V_k$ be its vertex parts, and let H be the graph obtained from G by subdividing each edge. Note that H can be viewed as a $\left(k + \binom{k}{2}\right)$-partite graph with parts $V_1, \ldots, V_k$ and parts V_{ij}, $1 \leq i < j \leq k$, formed by vertices of degree two associated with edges between the parts V_i and V_j in the graph G. Let R be a pattern with $k + \binom{k}{2}$ nodes u_i, $1 \leq i \leq k$, and u_{ij}, $1 \leq i < j \leq k$, such that R has no loops but all pairs of nodes of R are joined edges except for pairs u_i and u_{ij} and pairs u_j and u_{ij}, $1 \leq i < j \leq k$. Set $V_{u_i} = V_i$, $1 \leq i \leq k$, and $V_{u_{ij}} = V_{ij}$, $1 \leq i < j \leq k$; this yields a vertex partition $(V_u)_{u \in V(R)}$ of the graph H. The graph H^R is a $\left(k + \binom{k}{2}\right)$-partite graph. Note that if $k \geq 4$, then H^R contains a clique with $k + \binom{k}{2}$ vertices if and only if H contains a subdivision of a clique with k vertices. Consequently, if $k \geq 4$, then G contains a clique with k vertices if and only if H^R contains a clique with $k + \binom{k}{2}$ vertices. Since H is a 2-degenerate graph, the proposition now follows. ◀

Proposition 14 leaves it open whether the k-clique problem is fixed parameter tractable when d and R are fixed and k is the parameter. We address this affirmatively in the next proposition.

▶ **Proposition 15.** *For every integer d and every pattern R, the k-clique problem in the class $\mathcal{G}_d^R$ is fixed parameter tractable when parameterized by k.*

Proof. We present an algorithm that decides whether a graph $H \in \mathcal{G}_d^R$ contains a complete subgraph with k vertices. In view of Theorem 5 and Lemma 12, we may assume (at the expense of considering a larger integer d and a larger pattern R) that the algorithm is given a graph $G \in \mathcal{G}_d$, a pattern R and a vertex partition $(V_u)_{u \in V(R)}$ such that $H = G^R$. If R contains a node u with a loop such that $|V_u| > dk$, then H contains a complete subgraph with k vertices: indeed, since the subgraph $G[V_u]$ is $(d+1)$-colorable, $G[V_u]$ contains an independent set of at least k vertices; this set forms a complete subgraph in $H = G^R$. Hence, we may assume that the following holds for every node u of R: u has no loop or $|V_u| \leq dk$.

We next observe that $H[V_u]$ contains at most $\max\{2^{dk}, 2^d|V_u|\}$ (not necessarily inclusion-wise maximal) complete subgraphs. Indeed, if $|V_u| \leq dk$, then there are at most 2^{dk} subsets of V_u and the claim follows. Otherwise, u has no loop and $G[V_u] = H[V_u]$ and the claim follows since $H[V_u]$ is d-degenerate. Let $\mathcal{C}_u$ be the set of all complete subgraphs of $H[V_u]$ (including the one with no vertices, i.e., the one induced by the empty set). The algorithm

now tests all possible combinations of subgraphs from $\mathcal{C}_u$, $u \in V(R)$, whether they form a complete subgraph in H. This identifies all complete subgraphs of H. The running time of the algorithm is bounded by the product of the sizes of the set $\mathcal{C}_u$, $u \in V(R)$, i.e., the algorithm runs in time $O\left(2^{dkK} n^{K+O(1)}\right)$, where n is the number of vertices of the input graph H and K is the number of nodes of the pattern R. ◀

5 Conclusion

Our results have been motivated by the characterization of graphs that are first interpretable in graphs with bounded maximum degree as given in Proposition 3. While we were able to translate Theorem 4 to the setting of Conjecture 1 and even the more general setting of degenerate graphs, Proposition 3 fails to extend to the setting of Conjecture 1, which we now outline. Consider a class $\mathcal{G}$ of all star forests, one of the simplest classes of sparse graphs with unbounded maximum degree, and also consider the simple first order graph interpretation scheme I such that two vertices in $I(G)$ are joined by an edge iff their distance in a graph G is at most two. The graph class $I(\mathcal{G})$ contains all graphs G such that each component of G is a complete graph. Let $\mathcal{H}$ be a graph class and R a pattern such that $I(\mathcal{G}) \subseteq \mathcal{H}^R$, where $\mathcal{H}^R$ is the class of graphs H^R, $H \in \mathcal{H}$. Let K be the number of nodes of R and consider a graph $G \in \mathcal{G}$ formed by $k \cdot K$ stars each with $k \cdot K - 1$ leaves for an integer $k \geq K + 1$. The graph $I(G)$ consists of $k \cdot K$ cliques each having $k \cdot K$ vertices; let $C_1, \ldots, C_{k \cdot K}$ be the vertex sets of the k cliques forming the graph $I(G)$. Suppose that $I(G) = H^R$ for a graph $H \in \mathcal{H}$ and a vertex partition $(V_u)_{u \in V(R)}$ of H. There exist a node u such that $|V_u \cap C_i| \geq k$ for at least two different indices i; by symmetry we can assume that $|V_u \cap C_1| \geq k$ and $|V_u \cap C_2| \geq k$. If the node u has a loop in R, then the graph H contains all edges between $V_u \cap C_1$ and $V_u \cap C_2$, i.e., H contains a complete bipartite subgraph with parts of sizes k. If the node u does not have a loop in R, then $H[V_u \cap C_1]$ is a complete subgraph with k vertices, i.e., H contains a complete bipartite subgraph with parts of sizes $\lfloor k/2 \rfloor$. We conclude that the graph class $\mathcal{H}$ contains graphs with arbitrary large complete bipartite subgraphs; this implies that the graph class $\mathcal{H}$ does not have bounded expansion.

In view of the results presented in Section 4, it is natural to wonder about the fixed parameter tractability of other important graph problems. One of such problems is the k-dominating set problem, which asks whether the input graph contains k vertices such that each vertex of the graph is one of these k vertices or adjacent to at least one of them. The k-dominating set problem is known to be fixed parameter tractable for d-degenerate graphs [1] when parameterized by d and k. However, we were not able to resolve the fixed parameter complexity of the k-dominating set problem in graphs obtained by complementing vertex subsets of d-degenerate graphs and even the following particular case seems to be challenging.

▶ **Problem 16.** *Is the k-dominating set problem in the complements of d-degenerate graphs fixed parameter tractable when parameterized by d and k?*

References

1 Noga Alon and Shai Gutner. Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. In *Computing and Combinatorics, 13th Annual International Conference, COCOON, Proceedings*, volume 4598 of *Lecture Notes in Computer Science*, pages 394–405. Springer, 2007. doi:10.1007/978-3-540-73545-8.

2 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

3 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012. URL: http://www.cambridge.org/fr/knowledge/isbn/item5758776/?site_locale=fr_FR.

4 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

5 Anuj Dawar, Martin Grohe, and Stephan Kreutzer. Locally excluding a minor. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), Proceedings*, pages 270–279. IEEE Computer Society, 2007. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4276538.

6 Anuj Dawar and Stephan Kreutzer. Parameterized complexity of first-order logic. *Electronic Colloquium on Computational Complexity (ECCC)*, 16:131, 2009. URL: http://eccc.hpi-web.de/report/2009/131.

7 Zdenek Dvořák, Daniel Král', and Robin Thomas. Deciding first-order properties for sparse graphs. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS, Proceedings*, pages 133–142. IEEE Computer Society, 2010. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5669376.

8 Zdenek Dvořák, Daniel Král', and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36:1–36:24, 2013. doi:10.1145/2499483.

9 Kord Eickmeyer and Ken-ichi Kawarabayashi. FO model checking on map graphs. In *Fundamentals of Computation Theory - 21st International Symposium, FCT, Proceedings*, volume 10472 of *Lecture Notes in Computer Science*, pages 204–216. Springer, 2017. doi:10.1007/978-3-662-55751-8.

10 Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001. doi:10.1145/504794.504798.

11 Jakub Gajarský, Petr Hliněný, Jan Obdržálek, Daniel Lokshtanov, and M. S. Ramanujan. A new perspective on FO model checking of dense graph classes. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, pages 176–184. ACM, 2016. doi:10.1145/2933575.

12 Jakub Gajarský, Stephan Kreutzer, Jaroslav Nesetřil, Patrice Ossona de Mendez, Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. First-order interpretations of bounded expansion classes. To appear in proceedings of *Automata, Languages, and Programming - 40th International Colloquium, ICALP*, 2018.

13 Robert Ganian, Petr Hliněný, Daniel Král', Jan Obdržálek, Jarett Schwartz, and Jakub Teska. FO model checking of interval graphs. *Log. Methods Comp. Science*, 11(4), 2015. doi:10.2168/LMCS-11(4:11)2015.

14 Martin Grohe and Stephan Kreutzer. Methods for algorithmic meta theorems. In *Model Theoretic Methods in Finite Combinatorics, Contemporary Mathematics: AMS-ASL Joint Special Session*, volume 558, pages 181–206, 2011.

15 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. In *Symposium on Theory of Computing, STOC, proceedings*, pages 89–98. ACM, 2014. URL: http://dl.acm.org/citation.cfm?id=2591796.

16 Petr Hliněný, Filip Pokrývka, and Bodhayan Roy. FO model checking of geometric graphs. In *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, proceedings*, volume 89 of *LIPIcs*, pages 19:1–19:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. URL: http://www.dagstuhl.de/dagpub/978-3-95977-051-4.

17 Stephan Kreutzer. Algorithmic meta-theorems. *Electronic Colloquium on Computational Complexity (ECCC)*, 16:147, 2009. URL: http://eccc.hpi-web.de/report/2009/147.

18 Jaroslav Nešetřil and Patrice Ossona de Mendez. Linear time low tree-width partitions and algorithmic consequences. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, STOC*, pages 391–400. ACM, 2006.

19 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion I. decompositions. *Eur. J. Comb.*, 29(3):760–776, 2008. doi:10.1016/j.ejc.2006.07.013.

20 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion II. algorithmic aspects. *Eur. J. Comb.*, 29(3):777–791, 2008. doi:10.1016/j.ejc.2006.07.014.

21 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion III. restricted graph homomorphism dualities. *Eur. J. Comb.*, 29(4):1012–1024, 2008. doi:10.1016/j.ejc.2007.11.019.

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

23 Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

24 Detlef Seese. Linear time computable problems and logical descriptions. *Electr. Notes Theor. Comput. Sci.*, 2:246–259, 1995. doi:10.1016/S1571-0661(05)80203-8.

Average-Case Polynomial-Time Computability of Hamiltonian Dynamics

Akitoshi Kawamura
Kyushu University
Fukuoka, Japan
kawamura@inf.kyushu-u.ac.jp

Holger Thies
University of Tokyo
Tokyo, Japan
info@holgerthies.com

Martin Ziegler
KAIST
Daejeon, Republic of Korea
ziegler@cs.kaist.ac.kr
https://orcid.org/0000-0001-6734-7875

Abstract

We apply average-case complexity theory to physical problems modeled by continuous-time dynamical systems. The computational complexity when simulating such systems for a bounded time-frame mainly stems from trajectories coming close to complex singularities of the system. We show that if for most initial values the trajectories do not come close to singularities the simulation can be done in polynomial time on average. For Hamiltonian systems we relate this to the volume of "almost singularities" in phase space and give some general criteria to show that a Hamiltonian system can be simulated efficiently on average. As an application we show that the planar circular-restricted three-body problem is average-case polynomial-time computable.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases Computable Analysis, Real computation, Dynamical systems, Average-case complexity, Computation in physics

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.30

Acknowledgements This work was supported by the Japan Society for the Promotion of Science (JSPS), Core-to-Core Program (A. Advanced Research Networks), JSPS KAKENHI Grant Numbers JP18H03203 and JP18J10407, the Korean Ministry of Science and ICT grant NRF-2016K1A3A7A03950702. We thank Florian Steinberg for discussions on average-case complexity in analysis and the anonymous reviewers for many helpful comments.

1 Introduction

Many phenomena in nature can be modeled by continuous-time dynamical systems. Analyzing such phenomena is usually done by simulating the evolution of a system with digital computers. It is therefore crucial to better understand the computational properties of dynamical systems. One of the most basic questions one can ask about such a system is given the state of the system at some time t_0 what will be the state at some time $t > t_0$, i.e., to simulate the evolution for a finite time-frame.

The extended Church-Turing thesis is the statement that any physical computation device can be simulated efficiently with a Turing machine. While the thesis might be incompatible with quantum computation, at least for classical (non-quantum) physical computation the assumption seems reasonable. Thus, it should be possible to simulate trajectories of dynamical systems for problems in (classical) physics for a bounded time-frame efficiently as nature already provides an efficient "computation device". However, accurate numerical simulation of dynamical systems can be very hard. For example, Miller pointed out the difficulties when integrating the famous gravitational N-body problem [16]. The N-body problem is the problem of predicting the motion of N point masses under their mutual gravitational attraction. For a moderate number N of point masses, the main difficulty arises because close encounters of particles cause instabilities. Indeed, two particles colliding leads to a singularity in the analytic function describing the dynamics. On the other hand, Saari could show that at least for $N \leq 4$ singularities are rare in the sense that the set of initial values leading to singularities has Lebesgue measure zero [21, 22] (for $N > 4$ this is an open problem). Thus, a possible resolution to the inconsistency with the extended Church-Turing thesis might be that hard instances are extremely rare in nature and therefore it is possible to do the simulation efficiently on typical inputs. For further discussion on the extended Church-Turing thesis in classical physics, see e.g. [26].

The theory of real number computation based on a realistic model of approximation is known as *computable analysis*. Using this model, computational complexity theory can be applied to real functions [7, 8]. Simulating a continuous-time dynamical system corresponds to solving an initial value problem (IVP) for systems of ordinary differential equations. There are already several results on the computational complexity of solving IVPs in computable analysis. In particular, if the right-hand side function f of the equation $\dot{y} = f(y)$ is polynomial-time computable and Lipschitz-continuous, the unique solution y can be computed in PSPACE and can be hard for this class [6, 4]. On the other hand, for analytic right-hand side function the solution is also a polynomial-time computable function [18, 9]. However, this formulation does not really capture the notion of what is usually understood by simulating a dynamical system, as it is assumed that the solution exists on the whole time interval and only takes values in a known compact set, and there are several hidden factors depending on the function and the initial value that heavily influence the efficiency in practice.

Instead of fixing the initial value, it is therefore more natural to study the complexity of the function mapping initial values and time to the corresponding solution. This, however, poses the problem that the worst-case complexity for most interesting systems is unbounded. Indeed, it is quite obvious that the simulation should take longer the closer a trajectory approaches a singularity of the system as then higher precision is required. Nonetheless, the system might behave well for most initial values in the sense that the trajectory does stay far away from any singularities. We would then expect efficient simulation to be possible on typical inputs.

In this paper we want to formalize this intuition. In classical complexity theory, the notion of being efficiently computable on typical inputs is usually captured by *average-case complexity theory*, which often provides a more significant measure of the performance of an algorithm than worst-case complexity when the hard instances are rare. However, finding the right notion of average-case complexity poses some subtle difficulties. A structural theory of average-case complexity for discrete problems was introduced by Levin [13]. Schröder, Steinberg and Ziegler recently extended Levin's definition of average-case complexity to problems on real numbers [23].

In this paper we use their definition to show that many physical problems are indeed efficiently solvable on average. We first give a short introduction to the model of computation in Section 2 and define the most important notions that we need in the rest of the paper. In Section 3 we formalize a parameterized result for IVPs with analytic right-hand side. We show that restricted to a compact domain where the dynamics take place the solution can be computed in time polynomial in the output precision and a natural parameter depending on the function and the domain. In Section 4 we use this to show that if the "probability of trajectories to get close to complex singularities of the system" is small and if the right-hand side function can be evaluated efficiently on points not close to singularities, the simulation can be done in polynomial time on average. We then focus on a special case of dynamical systems that play an important role in classical physics, the Hamiltonian systems, and show that there is a simple way to bound the above probability in terms of the volume of singularities in phase-space. Finally, we apply our theorem to show that a special case of the three-body problem, the planar circular restricted three-body problem, can be simulated in polynomial time on average.

2 Computability and complexity in analysis

Computable Analysis extends classical computability theory to deal with uncountable quantities such as real numbers. The theory already dates back to Turing [24] with later important contributions for example by Grzegorczyk [3] and Lacombe [12]. The rigorous study of complexity in this model was initiated by Ko and Friedman [8]. In this section we briefly summarize the most important definitions. For a more detailed overview the reader is referred, e.g., to Weihrauch's monograph [25].

2.1 Computing real numbers and functions

Classically, computability is typically defined using Turing machines or an equivalent model of computation. Turing machines compute functions from finite strings to finite strings, that is, functions $F : \Sigma^* \to \Sigma^*$ for some fixed finite alphabet Σ. While computations over discrete objects like natural numbers, rational numbers or graphs can be defined by choosing an appropriate encoding for these objects as finite strings, the set of reals is uncountable and it is therefore impossible to find such an encoding. On the other hand, any real number can be approximated with arbitrarily small error by rational numbers. A real number x can therefore be encoded by a function that gives arbitrary exact approximations of x. More formally: A function $\varphi : \Sigma^* \to \Sigma^*$ is called a *name* for a real number $x \in \mathbb{R}$ if it maps strings of length n to the binary encoding of some integer z such that $\left|x - \frac{z}{2^n}\right| \leq 2^{-n}$. Any real number has infinitely many different names. We say x is *computable* if it has a computable name.

Similarly, a function $f : \mathbb{R} \to \mathbb{R}$ is computable if there is a computable function mapping names of x to names of $f(x)$, i.e., an algorithm that computes approximations of the output $f(x)$ with arbitrary precision while having access to arbitrary exact approximations of the input x. Such computation on names can be formally defined using oracle machines. Oracle machines can make queries to an oracle, a function $\varphi : \Sigma^* \to \Sigma^*$, during the computation. When making such a query the string w on a special tape, the so-called oracle tape, is replaced by the value of $\varphi(w)$ in a single time-step. We denote the oracle machine M with oracle φ by M^φ. M^φ again computes a function $\Sigma^* \to \Sigma^*$. A computable real function can then be defined as follows: A partial function $f :\subseteq \mathbb{R} \to \mathbb{R}$ is computable if there is an oracle machine M such that whenever φ is a name for $x \in \operatorname{dom} f$ the machine M^φ computes a name

for $f(x)$. The machine can behave arbitrarily on elements outside the domain of the function. The definition can be easily generalized to multidimensional functions $f :\subseteq \mathbb{R}^n \to \mathbb{R}^m$ by allowing multiple oracle and output tapes.

The running time $T_M(\varphi, w)$ for an oracle machine M with oracle $\varphi : \Sigma^* \to \Sigma^*$ and input $w \in \Sigma^*$ is defined as the number of steps the machine makes. If the machine M computes a real function $f : \mathbb{R} \to \mathbb{R}$ we can thus define the (worst-case) running time $T_M(x, n) \in \mathbb{N} \cup \{\infty\}$ of M for an argument $x \in \mathbb{R}$ and output precision $n \in \mathbb{N}$ as an upper bound of $T_M(\varphi, 0^n)$ over all names φ for x. We further say that a computable function $f : \mathbb{R} \to \mathbb{R}$ has (worst-case) time-complexity $T : \mathbb{N} \to \mathbb{N}$ if $T_M(x, n) \leq T(n)$ for all $x \in \operatorname{dom} f$ and that f is polynomial-time computable if T is a polynomial. Thus, the time-complexity of a real function is given in terms of the number of steps necessary to approximate the solution up to precision 2^{-n} depending on the output precision $n \in \mathbb{N}$.

Most work in real complexity theory restricts $\operatorname{dom} f$ to be a fixed compact set as this makes sure that a (finite) complexity bound exists. A more general theory taking into account the "size" of an oracle can be found in [5].

In this paper we sometimes use the notation that f is computable on some maximal domain $\operatorname{dom} f$ and *uniformly* allows some time-bound on a restricted subset of the domain. By this we mean that there is a single algorithm to compute f on its domain and the running time of this algorithm can be bounded on this subset.

2.2 Average-case complexity for real functions

Worst-case complexity as defined above has the disadvantage that the cost can be dominated by a small number of instances. A problem that can be solved efficiently on most inputs might thus still be hard from the perspective of computational complexity. *Average-case complexity* studies the average running time of an algorithm over all inputs and often gives a more realistic measure for the efficiency of an algorithm. Similarly to the class $\mathcal{P}$ in worst-case complexity we would like to have a class of average-case polynomial-time problems that we consider as "efficient on average". This notion should fulfill some basic robustness criteria. For example, composition of an average-case polynomial-time computable function with a worst-case polynomial-time computable function should still be computable in polynomial time on average. However, already in the discrete case this robustness property is not fulfilled by the most intuitive definitions. A definition that resolves this problem was given by Levin [13]. Schröder, Steinberg and Ziegler [23] recently extended Levin's notion to real number computations:

▶ **Definition 1.** Let (X, Σ, μ) denote a probability space and $T : X \times \mathbb{N} \to [0, \infty]$ a measurable function. We say that T is polynomial-time on average if there is some $\varepsilon > 0$ such that the function

$$n \mapsto \frac{1}{n} \int_X (T(x, n))^\varepsilon \, d\mu \tag{1}$$

is bounded by some constant $c > 0$.

If an algorithm runs in polynomial-time on average, there is a high probability that it runs in polynomial-time for a randomly selected input as by Markov's inequality for any $k > 0$ it is

$$\Pr[\, T(x, n) \geq k n^{\frac{1}{\varepsilon}} \,] \leq \frac{c}{k^\varepsilon}.$$

In this paper we only consider the case where $X \subseteq \mathbb{R}^d$, Σ is the Borel algebra over $\mathbb{R}^d$ and $\mu(A) = \frac{\lambda(A)}{\lambda(X)}$ where λ denotes the Lebesgue measure on $\mathbb{R}^d$.

3 Complexity of simulating dynamical systems

Dynamical systems are used to describe the evolution of a system over time. A dynamical system on the reals can be formally defined as follows:

▶ **Definition 2.** A dynamical system is a triple (X, T, Φ) of a non-empty set $X \subset \mathbb{R}^d$ called the *phase space*, a time set $T \subseteq \mathbb{R}$ and a (partial) function $\Phi :\subseteq X \times T \to X$ called *evolution operator* satisfying

1. $\Phi(x, 0) = x$, and
2. $\Phi(\Phi(x, t_1), t_2) = \Phi(x, t_1 + t_2)$

for $x \in X$ and all $t_1, t_2 \in T$ such that $(x, t_1), (\Phi(x, t_1), t_2) \in \operatorname{dom} \Phi$.

A point $x \in X$ is also called a *state* of the system and $\Phi(x_0, t)$ the state at time t (w.r.t. the initial value x_0). For a fixed initial value x_0 the function $\Phi(x_0, \cdot)$ is called *trajectory* through x_0. In this paper we only consider continuous-time systems where the time set is some real interval and the evolution operator is continuously differentiable with respect to time. The dynamics of such a system can be described by the solution of a system of autonomous first-order ODEs

$$\dot{y} = f(y), \quad y(t_0) = y_0 \tag{2}$$

for some function $f : D \subseteq \mathbb{R}^d \to \mathbb{R}^d$ and $y_0 \in \mathbb{R}^d$.

We call Equation (2) an *initial value problem* (IVP) with initial value y_0. The corresponding solution function $y : \mathbb{R} \to \mathbb{R}^d$ of the IVP gives a trajectory of the system. We sometimes also write $y(t; t_0, y_0)$ for the solution function with initial value $y(t_0) = y_0$ to make the dependency on the initial value explicit.

In the following, we use multi-index notation for tuples of non-negative integers $\beta = (\beta_1, \ldots, \beta_d)$, i.e., for $\beta \in \mathbb{N}^d$ and $x \in \mathbb{R}^d$ it is $|\beta| = \beta_1 + \cdots + \beta_d$, $\beta! = \beta_1! \ldots \beta_d!$, $x^\beta = x_1^{\beta_1} \ldots x_d^{\beta_d}$ and $D^\beta f = D_1^{\beta_1} \ldots D_d^{\beta_d} f$.

A function $f : U \to \mathbb{R}$ for $U \subseteq \mathbb{R}^d$ open is called *analytic* if for each $x_0 \in U$ there is an open neighborhood $V \subseteq U$ of x_0 such that for all $x \in V$ the power series expansion

$$\sum\nolimits_{\beta \in \mathbb{N}^d} a_\beta (x - x_0)^\beta$$

converges to $f(x)$. A function $f : D \to \mathbb{R}$ on a (possibly) non-open domain D is called analytic if it can be extended to an analytic function on some open domain $U \supseteq D$.

We further say a function $f : D \to \mathbb{R}^m, f(x) = (f_1(x), \ldots, f_m(x))$ is analytic if each of the component functions $f_1, \ldots, f_d : D \to \mathbb{R}$ are analytic. Any function analytic on some subset $D \subseteq \mathbb{R}^d$ can be extended to a complex analytic function on an open set $G \subseteq \mathbb{C}^d$. In this paper we only consider IVPs where the right-hand side function f is analytic. By the Cauchy-Kowalevski theorem the solution function y of such an IVP is again analytic.

3.1 Parameterized complexity for analytic initial value problems

Most results on IVPs in computable analysis assume the initial value to be fixed. In this work, however, we want to consider the average complexity over all initial values. We therefore first need to formalize how exactly the complexity of the solution depends on the chosen initial value. More formally, assume that we allow initial values from some set $A \subseteq \mathbb{R}^d$. We want to give a complexity bound for the solution function $Y :\subseteq A \times [0, 1] \to \mathbb{R}^d, (y_0, t) \mapsto y(t; 0, y_0)$ that maps initial values at time 0 and time $t \in [0, 1]$ to the solution of the initial value

problem at time t. Note that the restriction that the time is between 0 and 1 is somehow arbitrary. However, for the average-case analysis in the following chapter it is reasonable to assume that the time is bounded as there is no natural choice for a distribution on time.

The domain of Y are tuples $(y_0, t) \in A \times [0,1]$ where the solution with initial value y_0 exists up to time t. As this domain is not necessarily compact, we can not expect to find a simple complexity bound on the whole domain. However, we can subdivide the domain into subsets depending on some natural parameter of the system and give a complexity not only in terms of the output precision n but also in terms of an additional integer parameter. That is, we want to say that there is a uniform algorithm that computes the function on its whole domain and if we restrict the domain to some subset this algorithm runs in time polynomial in the output precision n and some parameter depending on the subset. We formalize this in the following definition.

▶ **Definition 3.** Let $f : D \subseteq \mathbb{R}^d \to \mathbb{R}^e$ be a computable real function and let $D = \bigcup_{i \in I} D_i$ be some (fixed) partition of its domain with index set I. We say f is C-polynomial-time computable for a function $C : I \to \mathbb{N}$ if there is an oracle machine M that

1. computes f on all inputs $x \in D$, and
2. there is a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that on inputs $x \in D_i$ the machine outputs a 2^{-n} approximation of $f(x)$ after at most $p(C(i) + n)$ steps.

We will usually partition the domain in terms of some positive real parameter α and use the short-hand notation "f is α^{-1}-polynomial-time computable on D_α". Formally, this means that f is C-polynomial-time computable w.r.t. the partition of the domain $D = \bigcup_{\alpha > 0} D_\alpha$ and the function $C : \mathbb{R}^+ \to \mathbb{N},\ C(\alpha) = \lceil \alpha^{-1} \rceil$.

Assume we are given an initial value problem (2) and the right-hand side function f is analytic on some compact $K \subseteq \operatorname{dom} f$. Then there is some integer C_K such that the bound

$$\left| D^\beta f(x) \right| \leq C_K^{|\beta|+1} \beta! \tag{3}$$

holds for all $\beta \in \mathbb{N}^d$ and $x \in K$ [10]. On the other hand, if for some $y_0 \in \operatorname{dom} f$ the solution exists up to time T then $y([0,T])$ is a compact subset of the domain. Thus, restricting f to this set suffices to compute the solution.

We usually can not assume that f is polynomial-time computable on its whole domain since its values can be unbounded. On the other hand on any compact subset $K \subseteq \operatorname{dom} f$ of the domain, C_K is an upper bound for the values of f. It is therefore a reasonable assumption that f is C-polynomial-time computable where C denotes a function that assigns each compact subset $K \subseteq \operatorname{dom} f$ an integer C_K as in Equation (3).

For simplicity let us from now assume that $T = 1$, i.e., we only consider initial values for which the solution exists up to time 1. The following theorem characterizes the complexity of the solution in terms of the parameterization given by C.

▶ **Theorem 4.** *Assume $f : D \subseteq \mathbb{R}^d \to \mathbb{R}^d$ is analytic and computable. Let $D_0 \subseteq D$ be some subset of initial values $y_0 \in D$ such that the solution y to the IVP (2) with initial value y_0 exists for all $t \in [0,1]$. For each $y_0 \in D_0$ let further $K(y_0) := y([0,1]; 0, y_0)$ and $D' := \bigcup_{y_0 \in D_0} K(y_0)$. Assume there is a function $C : D_0 \to \mathbb{N}$ such that*

- *f restricted to D' is C-polynomial-time computable, and*
- *$C(y_0)$ is a derivative bound for f on $K(y_0)$ as in (3).*

Then there exists a C-polynomial-time computable function $Z : \mathbb{R}^d \times [0,1] \times \mathbb{N} \to \mathbb{R}^d$ such that $Z(y_0, t, C(y_0)) = Y(y_0, t)$ for all $y_0 \in D_0$ and $t \in [0,1]$. That is, Z computes Y when given $C(y_0)$ as additional information.

Note that in general it is not possible to effectively get the parameter $C(y_0)$ from the function [2, 20]. Thus, in the above theorem we have to provide it as an additional input. For most natural systems, however, it is usually easy to get an upper bound for the parameter. We therefore assume that we can effectively get the parameter for the rest of the paper.

The main idea of the proof is that a simple power series based approach suffices to compute a local solution on some small time interval $[t_0, t_0 + \delta]$ in time polynomial in $n + C(y_0)$ (see e.g. [17]). To get a solution on a bigger interval this algorithm can be iterated several times. To show theorem 4 it therefore suffices to show that polynomially in $C(y_0)$ many iterations suffice and that it suffices to compute the intermediate values in each iteration with polynomial precision. The proof of the theorem is very similar to the proof of a recent result by Bournez, Graça and Pouly [1] for polynomial ODEs over unbounded time. We therefore chose to omit the details here. However, as both the statement and notation of our theorem are quite different from theirs, we included a proof in the appendix for completeness.

4 Average-case complexity for dynamical systems

While theorem 4 gives a very general characterization for the complexity of simulating initial value problems, it is usually not very useful as the complexity bound can differ for each initial value y_0. In general there is no way around this as the trajectories for distinct initial values can be quite different. On the other hand, it might be the case that most trajectories behave nicely in the sense that they stay far away from singularities of the system. In that case, we would like to say that the simulation can "usually" be done efficiently. This notion can be made formal using average-case complexity.

Such an average-case analysis, however, requires that we can get a bound on the probability for a trajectory to come close to a singularity. Usually, there is no easy way to assign such a probability. On the other hand, if the system has the special property that the volume of subsets of phase-space is preserved over time then a small volume of singularities in phase space indicates that the set of initial values coming close to singularities is also small. A large class of dynamical systems that have this property are *Hamiltonian systems*.

A d degree-of-freedom (time-independent) Hamiltonian system is a $2d$-dimensional dynamical system $\dot{y} = f(y)$ where the state can be split into two variables $y(t) = (q(t), p(t))$ for smooth functions $q, p : \mathbb{R} \to \mathbb{R}^d$ that satisfy the system of $2d$ first order ordinary differential equations

$$\dot{q}(t) = \frac{\partial H(p,q)}{\partial p}, \quad \dot{p}(t) = -\frac{\partial H(p,q)}{\partial q} \tag{4}$$

for some smooth, real-valued function $H : \mathbb{R}^{2d} \to \mathbb{R}$. The functions q and p are called the *position* and *momentum* of the system and H is called the *Hamiltonian*. We only consider time-independent systems where the Hamiltonian is constant over time and therefore is sometimes also called the *energy* of the system.

4.1 Average-case complexity for Hamiltonian system

In this chapter we define some criteria under which a given Hamiltonian system can be simulated in polynomial time on average. Here, it is more natural to formalize the results in terms of distance to complex singularities of a complex analytic extension instead of a derivative bound. We first (independently of the system being Hamiltonian or not) formalize the notion of a point being close to a (complex) singularity of an analytic function.

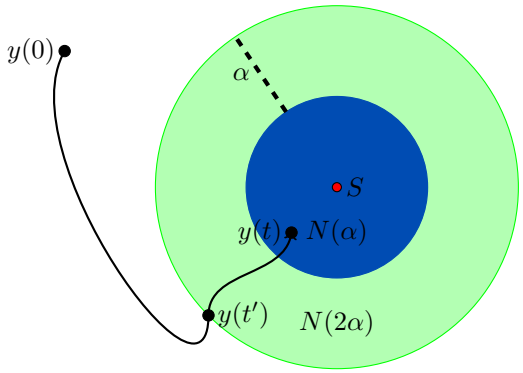


Figure 1 An α-singularity is a state of the system where the distance to a singularity is at most α. If the distance at time $t = 0$ is at least 2α and there is an α-singularity at time $t > 0$ there has to be some time t' when $y(t')$ is 2α close to the singularity. While the distance is between 2α and α the velocity is bounded which can be used to bound the minimum time the particle has to spend in the green region.

▶ **Definition 5.** Let $f : D \subseteq \mathbb{R}^d \to \mathbb{R}^d$ analytic. For any $\alpha > 0$ and $A \subseteq D$ we define the following sets:

1. $G(\alpha)$ is the set of points $x \in D$ such that there is a complex analytic extension $\widetilde{f}$ of f that is analytic on all $z \in \mathbb{C}^d$ with $|x - z| \leq \alpha$,
2. $N(\alpha) := D \setminus G(\alpha)$,
3. $R_A \subseteq D$ is the set of points reachable from A in time $t \leq 1$, i.e., $x \in R_A$ if there is $y_0 \in A$ and $t \in [0, 1]$ such that $y(t; 0, y_0) = x$,
4. $G_A(\alpha) := G(\alpha) \cap R_A$ and $N_A(\alpha) := N(\alpha) \cap R_A$,
5. $A(\alpha) := \{y_0 \in A : y(t; 0, y_0) \in G(\alpha) \text{ for all } t \in [0, 1]\}$,
6. $B(\alpha) := A \setminus A(\alpha)$.

We call a point $x \in N(\alpha)$ an *α-singularity*, $x \in A(\alpha)$ an *α-good* initial value (*α-bad* for $x \in B(\alpha)$) and points in R_A *reachable points*.

If we restrict the initial values to be contained in a set A and the volume of $B(\alpha)$ is small in comparison to A and if the growth of f behaves "nicely" on $G(\alpha)$ then the simulation can be done in polynomial time on average:

▶ **Theorem 6.** *Let $f : D \subseteq \mathbb{R}^d \to \mathbb{R}^d$ analytic and computable. Consider the solution function Y to the IVP (2) restricted to a bounded subset of initial values $A \subseteq D$. Assume that there is some polynomial $m : \mathbb{N} \to \mathbb{N}$ such that $\left|\tilde{f}\right| \leq m(\alpha^{-1})$ holds for any point in $G_A(\frac{\alpha}{2})$ and that f on $G_A(\alpha)$ is α^{-1}-polynomial-time computable. Then*

1. *Y is α^{-1}-polynomial-time computable on initial values in $A(\alpha)$, and*
2. *If there is additionally some constant $\gamma > 0$ such that $\frac{\lambda(B(\alpha))}{\lambda(A)} \leq \alpha^{\gamma}$ then Y can be computed in polynomial time on average (w.r.t. the restriction of d-dimensional Lebesgue measure to A).*

Proof. We show how to apply theorem 4 to prove the first part of the theorem. Let us first show that the polynomial m can be used to get a derivative bound on $G(\alpha)$.

Assume $\left|\tilde{f}(z)\right| \leq M(\alpha)$ for all $z \in G(\frac{\alpha}{2})$ for a function M with $M(\alpha) \geq \frac{2}{\alpha}$. For $z_0 \in G(\alpha)$, consider the polydisc $D = \prod_{i=1}^{d} D_i$ with $D_i := \{z \in \mathbb{C} : |z_{0,i} - z| \leq \frac{\alpha}{2}\}$. Now, $\tilde{f}$ is analytic

on D by the definition of $G(\alpha)$. Furthermore $\left|\tilde{f}\right|$ is bounded on D by $M(\alpha)$. By Cauchy's integral formula it follows

$$\begin{aligned}\left|D^\beta f(z_0)\right| &= \left|\frac{\beta!}{(2\pi i)^d}\int_{\xi_1\in\partial D_1}\cdots\int_{\xi_d\in\partial D_d}\frac{f(\xi_1,\ldots,\xi_d)}{(\xi_1-z_{0,1})^{\beta_1+1}\cdots(\xi_d-z_{0,d})^{\beta_d+1}}\right| d\xi_1\cdots d\xi_d \\ &\leq \frac{\beta!}{(2\pi)^d}\left(2\pi\frac{\alpha}{2}\right)^d\frac{M(\alpha)}{(\frac{\alpha}{2})^{\beta+d}} = \beta!M(\alpha)\left(\frac{2}{\alpha}\right)^\beta.\end{aligned}$$

Therefore, $\left|D^\beta f(z_0)\right| \leq \beta!M(\alpha)^{|\beta|+1}$ for all $z_0 \in G(\alpha)$. As we assume that the set of initial values is bounded and $t \in [0,1]$, it further follows that the absolute value of points in $G_A(\alpha)$ is bounded by a linear function in $M(\alpha)$. Thus we can apply theorem 4 using the partition of the domain into sets $K_\alpha = G_A(\alpha)$ which concludes the proof of the first part of the theorem.

For the second part let μ be the restriction of the Lebesgue measure to A, i.e., $\mu(B) = \frac{\lambda(B)}{\lambda(A)}$ for all measurable subsets $B \subseteq A$. Let further $A_i = A(2^{-(i+1)}) \setminus A(2^{-i})$, i.e., A_i contains the initial values where the minimum distance to a complex singularity for any $t \in [0,1]$ is between 2^{-i} and $2^{-(i+1)}$. Since $A_i \subseteq B(2^{-i})$ it holds $\lambda(A_i) \leq \lambda(B(2^{-i}))$ and by assumption $\mu(A_i) \leq 2^{-i\gamma}$. On the other hand A_i is contained in $A(2^{-(i+1)})$ thus by the first part of the theorem and the assumption on the time bound of f it follows that Y is computable on initial values in A_i in time $u(n+2^i)$ for a polynomial u. Since $A = \bigcup_{i=0}^\infty A_i$ it holds

$$\int_A \frac{1}{n}T(x,n)^\varepsilon d\mu \leq \frac{1}{n}\sum_{i=0}^\infty \mu(A_i)T(A_i,n)^\varepsilon \leq \frac{1}{n}\sum_{i=0}^\infty 2^{-i\gamma}u(n+2^i)^\varepsilon$$

Let m be the highest coefficient in the polynomial u then for $\varepsilon \leq \frac{\gamma}{2m}$ the sum converges and thus $\int_A \frac{1}{n}T(x,n)^\varepsilon d\mu$ is bounded. ◀

Theorem 6 holds even if the system is not Hamiltonian, but usually there is no simple way to get a bound for the measure for the second part of the theorem. For Hamiltonian systems, on the other hand, we can exploit the fact that they preserve phase-space volume over time (this is known as Liouville's theorem). Thus, there is a relation between the volume $\lambda(A(\alpha))$ of initial values leading to almost singularities and the volume $\lambda(N_A(\alpha))$ of almost singularities in phase-space.

Saari uses a similar idea to show that for the three-body problem the set of initial values leading to collisions has measure 0. However, for our application we need a stronger quantification of how the measure correlates to the distance of the particles. The main difficulty is that almost singularities can occur at any time $t \in [0,1]$. Thus, theoretically we would have to consider infinitely many "copies" of $N_A(\alpha)$, one for each possible time. Saari manages to replace this by only countably infinite many copies which suffices to show that the set of initial values leading to collisions has measure zero. However, this approach does does not suffice to give a bound for α-singularities with $\alpha > 0$. We therefore need a slightly more complicated idea to show that finitely many copies suffice in our case. Note, however, that Saari's result holds for unbounded time while our idea is based on the time being bounded. In fact, a generalization of our result to unbounded time turns out to be false [27].

Let us now state our main theorem. As the property of preserving phase-space volume is in fact the only attribute we use about Hamiltonian systems, we can formulate the theorem in terms of the bigger class of volume-preserving (sometimes also called conservative) dynamical systems.

▶ **Theorem 7.** *Let $\dot{y} = f(y)$ be a volume-preserving dynamical system with $f : D \subseteq \mathbb{R}^d \to \mathbb{R}^d$ computable and analytic and let $A \subseteq D$ be a bounded subset of initial values. Assume that*

- *f is α^{-1}-polynomial-time computable on α-good initial values $y_0 \in G_A(\alpha)$,*
- *for all $\alpha > 0$ the measure of α-bad subsets of phase space is bounded by $\lambda(N_A(\alpha)) \leq \alpha^\gamma$ for some $\gamma > 0$,*
- *$\left|\widetilde{f}\right|$ is bounded by $M_\alpha := M(\lceil \alpha^{-1} \rceil)$ for a polynomial $M : \mathbb{N} \to \mathbb{N}$ on $N(2\alpha) \setminus N(\alpha)$.*

Then

1. *$\lambda(B(\alpha)) \leq \left(\frac{2M_\alpha}{\alpha} + 1\right)\lambda(N_A(2\alpha))$*
2. *The solution function $Y :\subseteq A \times [0,1] \to \mathbb{R}^d$ is polynomial-time computable on average.*

Proof. We only prove the first part. The second part then follows by applying theorem 6. For any $t \in [0,1]$ let $B_t(\alpha) \subseteq A$ be the subset of initial values that lead to an α-singularity at time t, i.e., $y_0 \in B_t(\alpha)$ iff $y(t; 0, y_0) \in N(\alpha)$. As the system is volume-preserving it holds $\lambda(B_t(\alpha)) \leq \lambda(N_A(\alpha))$ for all t and α. Obviously, $B(\alpha) = \bigcup_{t\in[0,1]} B_t(\alpha)$.

We now show that we can replace the infinite union by a union of finitely many slightly bigger sets. Let $y(0) \in B_t(\alpha)$ for some $t \in [0,1]$, i.e., $y(0)$ is an initial value such that $y(t) \in N(\alpha)$. If $y(0) \in N(2\alpha)$ then it holds that $y(0) \in B_0(2\alpha)$ by definition. Otherwise there has to be some time $t' > 0$ where $y(t') \in N(2\alpha) \setminus N(\alpha)$ for the first time (see Fig. 1). As for any $z \in N(2\alpha) \setminus N(\alpha)$ by assumption $\left|\tilde{f}(z)\right| \leq M_\alpha$. It follows that for any $s \in [t', t' + \frac{\alpha}{M_\alpha}]$ the difference of the state at time t' and the state at time s can be at most α, i.e., $y(s) \in G(\alpha)$. In particular, if $y(0) \in G(2\alpha)$ and for some $t \in [0,1]$ it holds that $y(t) \in N(\alpha)$ then there is some non-trivial time interval $[t_1, t_2]$ of length at least $\frac{\alpha}{M_\alpha}$ such that $y(s) \in N(2\alpha) \setminus N(\alpha)$ for all $s \in [t_1, t_2]$. Therefore, $[t_1, t_2]$ contains some multiple t^* of $\frac{\alpha}{2M_\alpha}$. In particular, $y(t^*) \in N(2\alpha)$ and thus by definition $y(0) \in B_{t^*}(2\alpha)$. This shows that for the finite subset $\{t_k\}_k \subset [0,1]$ of multiples of $\frac{\alpha}{2M\alpha}$ it holds that $B(\alpha) \subseteq \bigcup_{t_k} B_{t_k}(2\alpha)$. By applying the fact that the system is volume-preserving it holds $\lambda(B_{t_k}(2\alpha)) \leq \lambda(N_A(2\alpha))$ and thus the statement follows. ◀

4.2 Average-case complexity for the restricted three-body problem

As an application of theorem 7 we show that the solution of the restricted three-body problem can be computed in polynomial-time on average. The classical three-body problem is the problem of predicting the motion of three point masses under their mutual gravitational attraction. An important special case is that we assume that one particle P_3 has much smaller mass than the other two P_1 and P_2. In that case P_3 does not influence the motion of P_1 and P_2 significantly. The problem can therefore be simplified by assuming that P_3 is massless. The motion of the heavy particles can then be seen as a two-body problem. A further simplification can be achieved by assuming that the heavy particles move on a circular orbit around their common center of mass and P_3 only moves in the plane defined by P_1 and P_2. This problem is known as the *planar circular restricted three-body problem*. In spite of being much simpler than the general problem, the restricted problem shares many of the properties of the N-body problem that makes it interesting for our analysis. In particular, the restricted problem is a Hamiltonian dynamical system where the equation of motion is given by an analytic initial value problem. There is no general closed form solution and the motion can be chaotic [15]. The restricted three-body problem has been studied extensively by mathematicians and engineers.

By choosing appropriate units of measurement and a rotating coordinate system, the system can be brought in a simpler, normalized form only depending on a single parameter $\mu \in (0, 0.5]$. The masses of the particles P_1 and P_2 in the new units are given by μ and $1 - \mu$, respectively. The position of the heavy particles in the rotating coordinate system remains

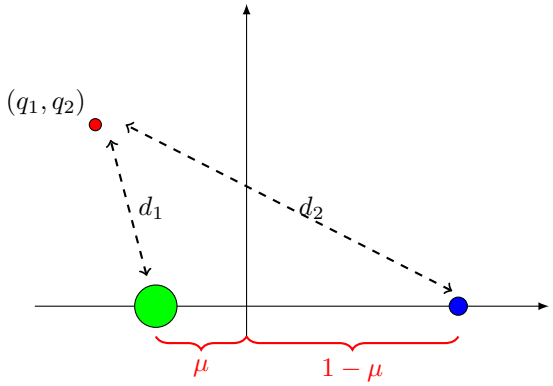


Figure 2 The planar circular restricted three-body problem in normalized form. P_1 and P_2 are fixed at position $(-\mu, 0)$ and $(1-\mu, 0)$. They have masses $1-\mu$ and μ and influence the motion of the massless particle P_3 at position (q_1, q_2).

fixed at $(-\mu, 0)$ and $(1-\mu, 0)$. We use $q = (q_1, q_2)$ to describe the coordinates of P_3 relative to that coordinate system (see Figure 2). A full derivation for the transformations as well as formulas to translate a solution for the normalized system to a solution for the non-modified system can e.g. be found in [11].

In Hamiltonian form the IVP can be written in terms of position $q \in \mathbb{R}^2$ and moment $p \in \mathbb{R}^2$ as

$$H(p,q) = \frac{1}{2}\|p\|^2 + q_2 p_1 - q_1 p_2 - \frac{\mu}{d_1} - \frac{1-\mu}{d_2} \tag{5}$$

where $d_1 := \sqrt{(q_1+\mu)^2 + q_2^2}$ and $d_2 := \sqrt{(q_1+\mu-1)^2 + q_2^2}$. This defines a dynamical system with phase space $\Gamma \subseteq \mathbb{R}^4$. We sometimes also consider the velocity of the particle given by $v(t) = (p_1(t) + q_2(t), p_2(t) - q_1(t))$ instead of the moment.

We assume that we start the simulation with initial values in the set A of points q_0, p_0 with $|q_0| \leq 1$ and $|p_0| \leq 1$. We first make the additional assumption that the energy is bounded by some constant $h > 0$, i.e., it holds $|H(q_0, p_0)| \leq h$. We denote the subset of initial values satisfying this bound by A_h.

In this problem, an α-singularity corresponds to the situation where P_3 gets close to either P_1 or P_2. Note that the absolute value of the moments tend to infinity when approaching a singularity. A bound on the volume of α-singularities in phase-space is given by the following lemma.

▶ **Lemma 8.** *For any $\alpha \in [0, 0.5]$, the Lebesgue measure of $N_{A_h}(\alpha)$ is given by $\lambda(N_{A_h}(\alpha)) \leq 8\pi^2 h \alpha^2$.*

Proof. We show that for the set $N_1(\alpha)$ of points coming close to P_1 it holds that $\lambda(N_1(\alpha)) \leq \pi^2 h \alpha^2$. Then the claim follows as $N_{A_h}(\alpha) = N_1(\alpha) \cup N_2(\alpha)$. We first change to a new coordinate system (x, v) in terms of position and velocity of the particle such that P_1 is at the origin (note that this change preserves volume). The Hamiltonian of the problem in these coordinates can be written as

$$E(x,v) = \frac{1}{2}\|v\|^2 - \frac{1}{2}\|x\|^2 + x_1\mu - \frac{\mu}{\|x\|} - \frac{1-\mu}{d_2} - 0.5\mu^2$$

Let $\Gamma \subseteq \mathbb{R}^4$ the phase-space of the problem. Γ can be parameterized in terms of the position

and the energy E by

$$\Phi : (E, r, \varphi, \psi) \;\mapsto\; \big(r\cos(\varphi), r\sin(\varphi), R(E, r, \varphi)\cos(\psi), R(E, r, \varphi)\sin(\psi)\big) \tag{6}$$

with

$$R(E, r, \varphi) := \sqrt{2E + \mu^2 - 2\mu r\cos(\varphi) + \frac{2\mu}{r} + \frac{2-2\mu}{d_2} + r^2} \tag{7}$$

It is $N_1(\alpha) \subseteq \Phi(G)$ with $G := [-h, h] \times [0, \alpha] \times [0, 2\pi) \times [0, 2\pi)$. Thus the volume can be bounded by

$$\lambda(N_1(h, \alpha)) \;\le\; \int_{\Phi(G)} dq_1\, dq_2\, dv_1\, dv_2 \;=\; \int_G |\det(\mathrm{D}\,\Phi)|\, dr\, dh\, d\varphi\, d\psi \;=\; 4\pi^2 \cdot \alpha^2 \cdot h.$$

The last equality holds since $\det(\mathrm{D}\,\Phi) = r$. ◀

It further holds that a bound on the energy implies that both the position and the velocity of the particle are bounded as long as the particle does not get close to one of the singularities.

▶ **Lemma 9.** *For any $q, v \in G(\alpha) \cap A_h$ it holds $\|q\| \le 2h + 10$ and $\|v\| \le \sqrt{(2h+11)^2 + \frac{1}{\alpha}}$.*

Proof. Assume $d_1 \ge \alpha$ and $d_2 \ge \alpha$. The Hamiltonian in terms of the velocity is given by

$$H(v, q) = \frac{1}{2}\|v\|^2 - \frac{\mu}{d_1} - \frac{1-\mu}{d_2} - \frac{1}{2}\|q\|^2.$$

Thus the bound on the Hamiltonian implies the bound

$$\|v\|^2 \le 2h + |q|^2 + \frac{2}{\alpha} \tag{8}$$

on the velocity.

Now assume $\|q\| \ge 2$ then both $d_1 \ge 1$ and $d_2 \ge 2$ and thus $\|v\|^2 \le 2h + |q|^2 + 2$ holds. Let $h' = \max(h, 2)$ then $\|v\| \le h' + \|q\|$. Now consider for $n \ge 1$ the subsets U_n where $n-1 \le \|q\| \le n$. For $n \ge 3$, $(q, v) \in U_n$ implies that $\|v\| \le h' + n$. In particular the minimum time to get from U_n to U_{n+1} is $t_n := \frac{1}{h'+n}$. As for the initial values $\|q\| \le 2$ the minimum time needed to reach a state where $\|q\| \ge N$ is bounded by

$$T_N := \sum_{i=3}^{N} \frac{1}{h' + n} = \mathcal{H}_{N+h'+3} - \mathcal{H}_{h'+3}$$

where $\mathcal{H}_N := \sum_{k=1}^{N} \frac{1}{k}$ denotes the N-th *harmonic number*. By the well known bound

$$\gamma + \log(N) \le \mathcal{H}_N \le \gamma + \log(N+1)$$

it holds

$$T_N \ge \log\left(\frac{N + h' + 3}{h' + 4}\right).$$

As the time is bounded by 1 it follows that $\|q\| \le 2h' + 6 \le 2h + 10$. Inserting this back into (8) yields

$$\|v\| \le \sqrt{2h + (2h+10)^2 + \frac{2}{\alpha}}$$

from which the claim follows. ◀

From this it can be easily followed that the right-hand side function of the ODE system is computable in time polynomial in $\lceil \alpha^{-1} \rceil$ and h and that a polynomial bound in those terms holds for complex analytic extensions of the function on $N(2\alpha) \setminus N(\alpha)$.

As for initial values it holds that the magnitude of both position and moment are bounded by 1, it follows from the Equation for the Hamiltonian (5) that high energy can only be due to particles being already close at time 0. In particular for α-good initial values the following bound on the Hamiltonian holds.

▶ **Lemma 10.** *Assume* $\|q\| \leq 1$, $\|p\| \leq 1$ *and* $q, p \notin N(\alpha)$ *then* $H(p,q) \leq 3 + \frac{2}{\alpha}$.

Thus the above polynomial bounds can all be stated only in terms of α. In particular all conditions in theorem 7 are satisfied and thus the problem is polynomial-time computable on average:

▶ **Theorem 11.** *Simulating the restricted three-body problem for time* $t \leq 1$ *for initial values* p_0, q_0 *with* $|p_0| \leq 1$, $|q_0| \leq 1$ *is possible in polynomial time on average.*

5 Conclusion

We applied a recent definition of average-case complexity in analysis to problems in classical physics. We gave some general conditions which show that a continuous-time system can be computed efficiently on average. For the important special case of Hamiltonian systems, we showed that a simpler approach based on the volume of almost singularities in phase-space usually suffices. We applied our theory to the planar circular restricted three-body problem and showed that it indeed can be computed in polynomial time on average. The same can easily be done for some other simple dynamical systems.

However, our theory does not easily generalize to some other more complicated systems like the classical N-body problem as bounding the volume of singularities in phase space is more complicated for these systems. While we think that most systems in nature can indeed be simulated efficiently and that a similar result holds for, e.g., the general N-body problem it is unlikely that our approach can be easily adapted to this case as for $N > 4$ it is not even known if the set of initial conditions leading to singularities has measure zero.

Nonetheless, we hope that we can at least extend our theory to the general three-body problem and some other interesting problems in future work.

References

1. Olivier Bournez, Daniel S. Graça, and Amaury Pouly. On the complexity of solving initial value problems. In *ISSAC 2012-Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*, pages 115–121. ACM, New York, 2012. doi:10.1145/2442829.2442849.
2. Daniel S Graça, Ning Zhong, and Jorge Buescu. Computability, noncomputability and undecidability of maximal intervals of IVPs. *Transactions of the American Mathematical Society*, 361(6):2913–2927, 2009.
3. A. Grzegorczyk. On the definitions of computable real continuous functions. *Fund. Math.*, 44:61–71, 1957.
4. Akitoshi Kawamura. Lipschitz Continuous Ordinary Differential Equations are Polynomial-Space Complete. *Computational Complexity*, 19(2):305–332, 2010.
5. Akitoshi Kawamura and Stephen Cook. Complexity theory for operators in analysis. *ACM Transactions on Computation Theory*, 4(2):Article 5, 2012.

6 Ker-I Ko. On the computational complexity of ordinary differential equations. *Information and Control*, 58(1-3):157–194, 1983.

7 Ker-I Ko. *Complexity theory of real functions*. Progress in Theoretical Computer Science. Birkhäuser Boston Inc., Boston, MA, 1991. doi:10.1007/978-1-4684-6802-1.

8 Ker-I Ko and Harvey Friedman. Computational complexity of real functions. *Theoretical Computer Science*, 20(3):323–352, 1982. doi:10.1016/S0304-3975(82)80003-0.

9 Ker-I Ko and Harvey Friedman. Computing power series in polynomial time. *Advances in Applied Mathematics*, 9(1):40–50, 1988.

10 Hikosaburo Komatsu. A characterization of real analytic functions. *Proceedings of the Japan Academy*, 36(3):90–93, 1960.

11 Wang Sang Koon, Martin W. Lo, Jerrold E. Marsden, and Shane D. Ross. *Dynamical systems, the three-body problem and space mission design*. World Scientific, 2000.

12 Daniel Lacombe. Sur les possibilités d'extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles. In *Le raisonnement en mathématiques et en sciences expérimentales*, Colloques Internationaux du Centre National de la Recherche Scientifique, LXX, pages 67–75. Editions du Centre National de la Recherche Scientifique, Paris, 1958.

13 Leonid A. Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.

14 X. Mao. *Stochastic Differential Equations and Their Applications*. Ellis Horwood series in mathematics and its applications. Horwood Pub., 1997.

15 Andrea Milani. Chaos in the three body problem. In *Predictability, stability, and chaos in N-body dynamical systems*, pages 11–33. Springer, 1991.

16 R. H. Miller. Numerical difficulties with the gravitational n-body problem. In Dale G. Bettis, editor, *Proceedings of the Conference on the Numerical Solution of Ordinary Differential Equations*, pages 260–275, Berlin, Heidelberg, 1974. Springer Berlin Heidelberg.

17 Bernd Moiske and Norbert Th. Müller. Solving initial value problems in polynomial time. In *Proc. 22 JAIIO - PANEL '93, Part 2*, pages 283–293, Buenos Aires, 1993. URL: http://www.uni-trier.de/mueller.

18 Norbert Th. Müller. Uniform Computational Complexity of Taylor Series. In *Proc. 14th International Colloquium on Automata, Languages, and Programming*, volume 267 of *LNCS*, pages 435–444. Springer, 1987.

19 Norbert Th. Müller. Polynomial time computation of Taylor series. *Proc. 22 JAIIO-PANEL'93, Part 2, Buenos Aires*, 259:281, 1993.

20 Norbert Th. Müller. Constructive Aspects of Analytic Functions. In *Proc. Workshop on Computability and Complexity in Analysis*, volume 190 of *InformatikBerichte*, pages 105–114. FernUniversität Hagen, 1995.

21 Donald G. Saari. Improbability of Collisions in Newtonian Gravitational Systems. II. *Transactions of the American Mathematical Society*, 181:351–368, 1973. doi:10.2307/1996638.

22 Donald G. Saari. A global existence theorem for the four-body problem of Newtonian mechanics. *Journal of Differential Equations*, 26(1):80–111, 1977. doi:10.1016/0022-0396(77)90100-0.

23 Matthias Schröder, Florian Steinberg, and Martin Ziegler. Average-Case Bit-Complexity Theory of Real Functions. In *Mathematical Aspects of Computer and Information Sciences*, pages 505–519. Springer, Cham, 2015. doi:10.1007/978-3-319-32859-1_43.

24 A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(1):230–265, 1936. doi:10.1112/plms/s2-42.1.230.

25 Klaus Weihrauch. *Computable Analysis*. Springer, Berlin/Heidelberg, 2000.

26 Andrew Chi-Chih Yao. Classical physics and the Church–Turing Thesis. *Journal of the ACM (JACM)*, 50(1):100–105, 2003.
27 Lei Zhao. Quasi-Periodic Almost-Collision Orbits in the Spatial Three-Body Problem. *Communications on Pure and Applied Mathematics*, 68(12):2144–2176, 2015.

A Proof of Theorem 4

The proof consists of two parts. We first show that the local solution on some small interval with length polynomial in $C(y_0)^{-2}$ can be computed in $C(y_0)$-polynomial time. We then show that iterating the algorithm for the local solution until reaching time $t = 1$ can still be done within this complexity bound.

A.1 Computing a local solution

We formulate the complexity result for the local solution as the following Lemma.

▶ **Lemma 12.** *Let $D \subseteq \mathbb{R}^d$ analytic and computable. Then there is an algorithm that given an initial $y_0 \in \mathbb{R}^d$, time $t \in \mathbb{R}$ and an integer $C \in \mathbb{N}$ such that C is a derivative bound for f on the closed ball $\overline{B_C(y_0)}$ of radius $\frac{1}{C}$ around y_0 returns the solution $y(t)$ to the IVP (2) for any t with $|t - t_0| \leq \frac{1}{2(d+1)C^2}$. Assume f is C-polynomial-time computable on $\overline{B_C(y_0)}$ then so is y for all t that satisfy the above bound.*

Proof. Cauchy's existence theorem guarantees that the solution exists and is analytic on this time-interval. The polynomial-time result has in principle already been shown, e.g., by Moiske and Müller [17]. The main idea is to first compute the coefficients of the power series of f around t_0. Müller [19] showed that with the above bounds for any $\beta \in \mathbb{N}^d$ the coefficient a_β is computable in time polynomial in $n + C + |\beta|$. Further, if the power series of f around y_0 can be computed in time $T_f(n + |\beta|)$ then the k-th coefficient of the power series $(b_k)_{k\in\mathbb{N}}$ of any of the d components $y_1, \ldots, y_d$ of solution function y around t_0 can be computed from the power series of f in time $\mathrm{O}(k^d T_f(n + k^d))$.

For any $i = 1, \ldots, d$, the solution $y_i(t)$ for small enough t can then be approximated by summing up the truncated power series $\sum_{i=0}^{N} b_i(t - t_0)^i$ for some $N \in \mathbb{N}$. It can be shown that $|b_i| \leq (d + 1)^i C^{2i}$. Therefore for any t with $|t - t_0| \leq \frac{1}{2(d+1)C^2}$ it suffices to sum up $\mathrm{O}(n)$ coefficients to make the truncation error less than $2^{-(n+1)}$. To additionally make the approximation error less than $2^{-(n+1)}$ and thereby the total error less than 2^{-n} it suffices to approximate each b_i with error less than $2^{-(2n+1)}$. ◀

A.2 Extending to a global solution

We now prove the theorem by iterating the local solution algorithm. Let us first summarize the assumptions:

1. The right-hand side function $f : D \subseteq \mathbb{R}^d \to \mathbb{R}^d$ is analytic and computable,
2. The solution $y(t) := y(t; 0, y_0)$ exists for all time $t \in [0, 1]$,
3. Restricted to the set $K := y([0, 1]; 0, y_0)$ the integer C is a derivative-bound for f, i.e., $\left|D^\beta f(x)\right| \leq C^{|\beta|+1}\beta!$ for all $x \in K$,
4. The algorithm computing f gives a 2^{-n} approximation of $f(x)$ in time $poly(n + C)$ on any $x \in K$.

Note that each $x \in K$ is bounded by $|x| \leq C$ as it is in the image of y.

We want to show that iteratively using the local solution algorithm yields a polynomial-time algorithm mapping initial values $y_0 \in K$ and time $t \in [0, 1]$ to the solution $y(t)$ at time t.

Algorithm 1 Solving Initial Value Problems.

function SOLVE_IVP(f, y_0, t, n, C)
  $t_{curr} \leftarrow 0$
  $y_{curr} \leftarrow \text{APPROX}(y_0, m)$
  $h \leftarrow \frac{1}{2(d+1)C^2}$

  while $t_{curr} + h \leq t$ **do**
    $y_{curr} \leftarrow \text{LOCALSOLUTION}(y_{curr}, h, m, C)$
    $t_{curr} \leftarrow t_{curr} + h$
  return $\text{LOCALSOLUTION}(f, y_{curr}, t - t_{curr}, m, C)$

For this it suffices to show that we can always reach time 1 in polynomially many iterations and that it suffices to approximate all intermediate values with polynomial precision. For simplicity we fix the index i of the solution function y_i and simply denote it by y. By lemma 12 we can assume that there is an algorithm `LocalSolution` that computes for inputs $y_0 \in K$, $t \in [0,1]$ and $C, n \in \mathbb{N}$ with $t \leq \frac{1}{2(d+1)C^2}$ a rational approximation $q \in \mathbb{Q}$ such that $|y(t; t_0, y_0) - q| \leq 2^{-n}$. For a suitable choice of the intermediate precision parameter m Algorithm 1 computes an approximation q to the solution at any time $t \in [0,1]$ such that $|y(t; t_0, y_0) - q| \leq 2^{-n}$. It remains to show that m can be chosen to be polynomial in $n + C$. The algorithm makes at most $2(d+1)C^2$ steps to reach any time $t \in [0,1]$.

Let us first consider the error when computing the local solution y_{i+1} from y_i. There are two types of errors. The first one is due to the local solution algorithm only returning an approximation to the real solution with precision 2^{-m}. The second kind of error arises because of the accumulated error in y_i: Instead of solving the IVP problem with initial value y_i we use an approximation z_i of y_i as initial value. To bound the second kind of error we use the following fact:

▶ **Lemma 13.** *Assume $f : \mathbb{R}^d \to \mathbb{R}^d$ fulfills the conditions above. Then each component function $f_i : \mathbb{R}^d \to \mathbb{R}$ is Lipschitz-continuous on K with Lipschitz-constant $L = \sqrt{d}C^2$.*

Proof. This simply follows from the fact that all partial derivatives of f are bounded by C^2. ◀

This can be used to get a bound on the local error.

▶ **Lemma 14.** *Assume f fulfills the conditions above. Let $y_0, z_0 \in K$ be initial condition with $\|y_0 - z_0\|_\infty \leq \varepsilon$. Then for all $t < \frac{1}{2(d+1)C^2}$ it is $|y(t; y_0) - y(t; z_0)| \leq 2\varepsilon$.*

Proof. It holds that $y(t; y_0) = y_0 + \int_0^t |f(\mathbf{y}(\tau))|$. Thus

$$
\begin{aligned}
&|y(t; y_0) - y(t; z_0)| \\
&\leq |y_0 - z_0|_\infty + \int_0^t |f(y(\tau; y_0)) - f(y(\tau; z_0))| \, d\tau \\
&\leq \varepsilon + \int_{t_0}^t L \, |y(\tau; y_0) - y(\tau; z_0)| \, d\tau
\end{aligned}
$$

By Grönwall's Lemma [14, Chapter 1, Theorem 8.1] it follows that

$$|y(t; y_0) - y(t; z_0)| \leq \varepsilon e^{Lt}.$$

For $t \leq \frac{1}{2(d+1)C^2} \leq \frac{\sqrt{d}}{2(d+1)L}$ it then holds

$$|y(t;y_0) - y(t;z_0)| \leq \varepsilon e^{\frac{\sqrt{d}}{2(d+1)}} \leq 2\varepsilon. \qquad \blacktriangleleft$$

It is now easy to compute the total error for the algorithm depending on the parameter m for the intermediate precision.

▶ **Lemma 15.** *Assume algorithm 1 needs N iterations to reach the final time t. The total error E is bounded by $E \leq 2^{N+1-m}$.*

Proof. Let E_k be the error of y_{curr} after k iterations of the while-loop in Algorithm 1. It is $E_0 \leq 2^{-m}$ and $E_{k+1} \leq 2E_k + 2^{-m}$. It follows by induction that $E_N \leq 2^{N+1-m} - 2^{-m}$. ◀

Since the maximum number of iterations is bounded by $N = 2(d+1)C^2$ to achieve precision 2^{-n} it suffices to choose $m \geq n + 2(d+1)C^2 + 1$ which proves the theorem.

Generalized Budgeted Submodular Set Function Maximization

Francesco Cellinese
Gran Sasso Science Institute, L'Aquila, Italy
francesco.cellinese@gssi.it

Gianlorenzo D'Angelo
Gran Sasso Science Institute, L'Aquila, Italy
gianlorenzo.dangelo@gssi.it

Gianpiero Monaco
University of L'Aquila, L'Aquila, Italy
gianpiero.monaco@univaq.it

Yllka Velaj
University of Chieti-Pescara, Pescara, Italy
yllka.velaj@unich.it

Abstract

In this paper we consider a generalization of the well-known budgeted maximum coverage problem. We are given a ground set of elements and a set of bins. The goal is to find a subset of elements along with an associated set of bins, such that the overall cost is at most a given budget, and the profit is maximized. Each bin has its own cost and the cost of each element depends on its associated bin. The profit is measured by a monotone submodular function over the elements.

We first present an algorithm that guarantees an approximation factor of $\frac{1}{2}\left(1-\frac{1}{e^{\alpha}}\right)$, where $\alpha \leq 1$ is the approximation factor of an algorithm for a sub-problem. We give two polynomial-time algorithms to solve this sub-problem. The first one gives us $\alpha = 1-\epsilon$ if the costs satisfies a specific condition, which is fulfilled in several relevant cases, including the unitary costs case and the problem of maximizing a monotone submodular function under a knapsack constraint. The second one guarantees $\alpha = 1-\frac{1}{e}-\epsilon$ for the general case. The gap between our approximation guarantees and the known inapproximability bounds is $\frac{1}{2}$.

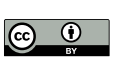
We extend our algorithm to a bi-criterion approximation algorithm in which we are allowed to spend an extra budget up to a factor $\beta \geq 1$ to guarantee a $\frac{1}{2}\left(1-\frac{1}{e^{\alpha\beta}}\right)$-approximation. If we set $\beta = \frac{1}{\alpha}\ln\left(\frac{1}{2\epsilon}\right)$, the algorithm achieves an approximation factor of $\frac{1}{2}-\epsilon$, for any arbitrarily small $\epsilon > 0$.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis, Theory of computation → Packing and covering problems

Keywords and phrases Submodular set function, Approximation algorithms, Budgeted Maximum Coverage

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.31

Related Version A full version of the paper is available at `http://arxiv.org/abs/1808.03085`.



43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018).
Editors: Igor Potapov, Paul Spirakis, and James Worrell; Article No. 31; pp. 31:1–31:14

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

1 Introduction

The Maximum Coverage (MC) is a fundamental combinatorial optimization problem which has several applications in job scheduling, facility locations and resource allocations [14, Ch. 3], as well as in influence maximization [17]. In the classical definition we are given a ground set X, a collection S of subsets of X with unit cost, and a budget k. The goal is selecting a subset $S' \subseteq S$, such that $|S'| \leq k$, and the number of elements of X covered by S' is maximized. A natural greedy algorithm starts with an empty solution and iteratively adds a set with maximum number of uncovered elements until k sets are selected. This algorithm has an approximation of $1 - \frac{1}{e}$ [20] and such result is tight given the inapproximability result due to Feige [11]. An interesting special case of the problem where this inapproximability result does not hold is when the size of the sets in S is small. In the maximum h-coverage, h denotes the maximum size of each set in S. This problem is APX-hard for any $h \geq 3$ [16] (notice that when $h = 2$ it is the maximum matching problem), while a simple polynomial local search heuristic has an approximation ratio very close to $\frac{2}{h}$ [4]. A polynomial time algorithm with approximation factor of $\frac{5}{6}$ is possible for the case when $h = 3$ [5]. In the Budgeted Maximum Coverage (BMC) problem, which is an extension of the maximum coverage, the cost of the sets in S are arbitrary, and thus a solution is feasible if the overall cost of the selected subset $S' \subseteq S$ is at most k. In [18], the authors present a polynomial time (greedy) algorithm with approximation factor of $1 - \frac{1}{e}$. In the Generalized Maximum Coverage (GMC) problem every set $s \in S$ has a cost $c(s)$, and every element $x \in X$ has a different weight and cost that depend on which set covers it. In [7], a polynomial time (greedy) algorithm with approximation factor of $1 - \frac{1}{e} - \epsilon$, for any $\epsilon > 0$, has been shown.

In all the above problems the profit of a solution is given by the sum of the weights of the covered elements. An important and studied extension is adopting a nonnegative, nondecreasing, submodular function f, which assigns a profit to each subset of elements. In the Submodular set Function subject to a Knapsack Constraint maximization (SFKC) problem we have a cost $c(x)$ for any element $x \in X$, and the goal is selecting a set $X' \subseteq X$ of elements that maximizes $f(X')$, where f is a monotone submodular function subject to the constraint that the sum of the costs of the selected elements is at most k. This problem admits a polynomial time algorithm that is $\left(1 - \frac{1}{e}\right)$-approximation [23]. Since the MC problem is a special case of SFKC problem, such result is tight. A more general setting was considered in [15], where the authors consider the following problem called Submodular Cost Submodular Knapsack (SCSK): given a set of elements $V = \{1, 2, \ldots, n\}$, two monotone non-decreasing submodular functions g and f ($f, g : 2^V \to \mathbb{R}$), and a budget b, the goal is finding a set of elements $X \subseteq V$ that maximizes the value $g(X)$ under the constraint that $f(X) \leq b$. They show that the problem cannot be approximated within any constant bound. Moreover, they give a $1/n$ approximation algorithm and mainly focus on bi-criterion approximation.

In this paper we consider the Generalized Budgeted submodular set function Maximization problem (GBSM) that is not captured by any of the above settings. We are given a ground set of elements X, a set of bins S, and a budget k. The goal is to find a subset of elements along with an associated set of bins such that the overall costs of both is at most a given budget and the profit is maximized. Each bin has its own cost, while the cost of each element depends on its associated bin. Finally, the profit is measured by a monotone submodular function over the elements.

We emphasize that the problem considered here is not a special case of the GMC problem, since we consider any monotone submodular functions for the profits. Moreover, it is possible to show that our cost function is not submodular and hence our problem is not a special

case of SCSK. Finally, our setting extends the SFKC problem, given that, the cost of an element is not fixed like in SFKC, but instead depends on the bin used for covering it.

In addition to its theoretical appeal, our setting is motivated by the adaptive seeding problem, which is an algorithmic challenge motivated by influence maximization in social networks [1, 22]. In its non-stochastic version, the problem is to select amongst certain accessible nodes in a network, and then select amongst neighbors of those nodes, in order to maximize a global objective function. In particular, given a set X and its neighbors $N(X)$ there is a monotone submodular function defined on $N(X)$, and the goal is to select $t \leq k$ elements in X connected to a set of size at most $k - t$ for which the submodular function has the largest value. Our setting is an extension of it since we consider more general costs.

1.1 Our results

In Section 3 we present an algorithm that guarantees an approximation factor of $\frac{1}{2}\left(1 - \frac{1}{e^{\alpha}}\right)$ for GBSM. Here, α is the approximation factor of an algorithm used to select a subset of elements whose ratio between marginal increment in the objective function and marginal cost is maximum. We give two polynomial-time algorithms to solve this sub-problem. In particular, in Section 4 we propose an algorithm that gives us $\alpha = 1 - \epsilon$ if the costs satisfy a specific condition. This latter is fulfilled in several relevant cases including the unitary costs case and the problem of maximizing a monotone submodular function under a knapsack constraint. In Section 5 we propose an algorithm that guarantees $\alpha = 1 - \frac{1}{e} - \epsilon$ for the general case.

The gap between our approximation guarantees and the known inapproximability bounds, i.e. the $1-\frac{1}{e}$ hardness for the MC problem [11] and the $1-\frac{1}{e^{1-\frac{1}{e}}}$ hardness for the non-stochastic adaptive seeding problem with knapsack constraint [21], is $\frac{1}{2}$, unless $P = NP$.

In Section 6, we extend our algorithm to a bi-criterion approximation algorithm in which we are allowed to spend an extra budget up to a factor β. An algorithm gives a $[\rho, \beta]$ bi-criterion approximation for GBSM if it is guaranteed to obtain a solution (S', X') such that $f(X') \geq \rho f(X^*)$ and $c(S', X') \leq \beta k$, where X^* is the optimal solution. We denote by β the extra-budget we are allowed to use in order to obtain a better approximation factor. Our algorithm guarantees a $\left[\frac{1}{2}\left(1 - \frac{1}{e^{\alpha\beta}}\right), \beta\right]$-approximation. If we set $\beta = \frac{1}{\alpha}\ln\left(\frac{1}{2\epsilon}\right)$, the algorithm achieves an approximation factor of $\frac{1}{2} - \epsilon$, for any arbitrarily small $\epsilon > 0$.

Due to space constraints, some of the proofs have been omitted and will appear in a full version of the paper.

1.2 Related work

Maximum coverage and submodular set function maximization are important problems. In the literature, besides the above mentioned ones, there are many other papers dealing with related issues. For instance, in the maximum coverage with group budgeted constraints, the set S is partitioned into groups, and the goal is to pick k sets from S to maximize the cardinality of their union with the restriction that at most one set can be picked from each group. In [6], the authors propose a $\frac{1}{2}$-approximation algorithms for this problem, and smaller constant approximation algorithm for the cost version. In the ground-set-cost budgeted maximum coverage problem, given a budget and a hypergraph, where each vertex has a non-negative cost and a non-negative profit, we want to select a set of hyperedges such that the total cost of the covered vertices is at most the budget and the total profit of all covered vertices is maximized. This problem is strictly harder than budgeted max coverage. The difference of our problem to the budgeted maximum coverage problem is that

the costs are associated with the covered vertices instead of the selected hyperedges. In [24], the authors obtain a $\frac{1}{2}\left(1-\frac{1}{\sqrt{e}}\right)$-approximation algorithm for graphs (which means having sets of size 2) and an FPTAS if the incidence graph of the hypergraph is a forest (i.e. the hypergraph is Berge-acyclic).

Maximizing submodular set function is another important research topic. The general version of the problem is: given a set of elements and a monotone submodular function, the goal is to find the subset of elements that gives the maximum value, subjected to some constraints. The case when the subset of elements must be an independent set of the matroid over the set of elements has been considered in [3], where the authors show an optimal randomized $\left(1-\frac{1}{e}\right)$-approximation algorithm. A simpler algorithm has been proposed in [13]. The case of multiple k matroid constraints has been considered in [19], where the authors give a $\frac{1}{k+\epsilon}$-approximation. An improved result appeared in [26]. Finally, unconstrained (resp. constrained) general non-monotone submodular maximization, have been considered in [2, 12] (resp. [25]).

Another related topic is the adaptive seeding problem in which the aim is to select amongst a set X of nodes of a network, called the *core*, and then adaptively selecting amongst the neighbors $N(X)$ of those nodes as they become accessible in order to maximize a submodular function of the selected nodes in $N(X)$ [1, 22]. An approximation algorithm with ratio $\left(1-\frac{1}{e}\right)^2$ has been proposed in [1]. In the adaptive seeding with knapsack constraints problem, nodes in X and in $N(X)$ are associated with a cost and the aim is to maximize the objective function while respecting a budget constraint. In this case, an $\left(1-\frac{1}{e}\right)\left(1-\frac{1}{e^{1-\frac{1}{e}}}\right)$-approximation algorithm is known [21]. In the non-stochastic version of these problems, all the nodes in $N(X)$ become accessible with probability one. Even in this case it is not possible to approximate an optimal solution within a factor greater than $\left(1-\frac{1}{e^{1-\frac{1}{e}}}\right)$, unless $P=NP$. A similar problem in which the core is made of the whole network and the network can be augmented by adding edges according to a given cost function has been shown to admit a 0.0878-approximation algorithm [9]. Finally, in [8, 10] the authors consider the problem where the core is made of a give set of nodes and the network can be augmented by adding edges incident only to the nodes in the core. In the unit-cost version of the problem where the cost of adding any edge is constant and equal to 1 the problem is NP-hard to be approximated within a constant factor greater than $1-(2e)^{-1}$. Then they provide a greedy approximation algorithm that guarantees an approximation factor of $1-\frac{1}{e}-\epsilon$, where ϵ is any positive real number. Then, they study the more general problem where the cost of edges is in $[0,1]$ and propose an algorithm that achieves an approximation guarantee of $1-\frac{1}{e}$ combining greedy and enumeration technique.

2 Preliminaries

We are given a set X of n elements and a set S of m bins. Let us denote the cost of a bin $s\in S$ by $c(s)\in\mathbb{R}_{\geq 0}$. For each bin $s\in S$ and element $x\in X$, we denote by $c(s,x)$ the cost of associating x to s. Given a budget $k\in\mathbb{R}_{\geq 0}$, and a monotone submodular function $f:2^X\to\mathbb{R}_{\geq 0}$[1], our goal is to find a subset X' of X and a subset $S'\neq\emptyset$ of S such that $c(S',X')=\sum_{s\in S'}c(s)+\sum_{x\in X'}\min_{s\in S'}c(s,x)\leq k$, and $f(X')$ is maximum. We call this problem the Generalized Budgeted submodular set function Maximization problem (GBSM).

[1] For a ground set X, a function $f:2^X\to\mathbb{R}_{\geq 0}$ is submodular if for any pair of sets $S\subseteq T\subseteq X$ and for any element $x\in X\setminus T$, $f(S\cup\{x\})-f(S)\geq f(T\cup\{x\})-f(T)$.

Our problem generalizes several well-known problems. Indeed, by setting $c(s,x) = \infty$, we do not allow the association of element x to bin s, while by setting $c(s,x) = 0$ we allow to assign element x to bin s with no additional cost. Moreover, we relax the constraints related to the association of elements to bins by setting $c(s) = 0$ for each $s \in S$, and $c(s_1,x) = c(s_2,x)$, for each $s_1, s_2 \in S$ and $x \in X$. By suitably combining these conditions we can capture the following problems: budgeted maximum coverage problem [18]; non-stochastic adaptive seeding problem [1] (also with knapsack constraints [21]); monotone submodular set function subject to a knapsack constraint maximization [23].

Let us consider a partial solution (S', X'). Given a set $T \subseteq X \setminus X'$, we denote by $c_{\min}(T)$ the minimum cost of associating the elements in T with a single bin in S, considering that the cost of bins in S' has been already paid, formally:

$$c_{\min}(T) = \min_{s \in S} \left\{ c_{S'}(s) + \sum_{x \in T} c(s,x) \right\},$$

where $c_{S'}(s) = c(s)$ if $s \notin S'$, and $c_{S'}(s) = 0$ if $s \in S'$. We call $c_{\min}(T)$ the *marginal cost* of T with respect to the partial solution (S', X'). We define $s_{\min}(T)$ as the bin $s \in S$ needed to cover T with cost $c_{\min}(T)$. Moreover, we denote by $\bar{c}(T)$ the cost of associating the elements in T to $s_{\min}(T)$, $\bar{c}(T) = c_{\min}(T) - c_{S'}(s_{\min}(T))$.

The *marginal increment* of $T \subseteq X$ with respect to the partial solution (S', X') is defined as $f(X' \cup T) - f(X')$. To simplify the notation, we use $g(T) = f(X' \cup T) - f(X')$ to denote the marginal increment.

In the algorithm in the next section, we will look for subsets of X that maximize the ratio between the marginal increment and the marginal cost with respect to some partial solution. In the following we define a family of subsets of X containing a set that approximates such maximal ratio. Given a partial solution (S', X'), we denote by $\mathcal{F}$ the family of subsets T of X that can be associated to bins in $S' \cup \{s\}$, for some single bin $s \in S$, with a cost such that $c(S' \cup \{s_{\min}(T)\}, T) \leq k$, formally $\mathcal{F} = \left\{ T \in 2^{X \setminus X'} \mid c(S' \cup \{s_{\min}(T)\}, T) \leq k \right\}$. A sub-family of $\mathcal{F}$ is an *α-list* with respect to (S', X') if it contains a subset T whose ratio between marginal increment and marginal cost is at least α times the optimal such ratio amongst all the subsets $\mathcal{F}$. Formally, $L \subseteq \mathcal{F}$ is an α-list with respect to (S', X') if

$$\max \left\{ \frac{g(T)}{c_{\min}(T)} \mid T \in L, c_{\min}(T) > 0 \right\} \geq \alpha \cdot \max \left\{ \frac{g(T)}{c_{\min}(T)} \mid T \in \mathcal{F}, c_{\min}(T) > 0 \right\}.$$

Note that the sets that maximize the above formula are not necessarily singletons due to the bin opening cost. Moreover, the algorithm given in the next section build partial solutions (S', X') in such a way that $c_{\min}(T) > 0$, for each $T \in \mathcal{F}$.

3 Greedy Algorithm

In this section we give an algorithm that guarantees a $\frac{1}{2}\left(1 - \frac{1}{e^{\alpha}}\right)$-approximation to the GBSM problem, if we assume that we can compute, in polynomial time, an α-list of polynomial size. In the next sections we will give two algorithms to compute such lists for bounded values of α.

The pseudo-code is reported in Algorithm 1. In the first step (line 3) we add all zero-cost bins to the solution. Then, the algorithm finds two candidate solutions. The first one is found at lines 4–11 with a greedy strategy as follows. The algorithm iteratively constructs a partial solution (S', X') by adding a subset $\hat{T}$ to X' and a bin $s_{\min}(\hat{T})$ to S'. In particular, at

Algorithm 1: General Algorithm.

Input : S, X
Output : S', X'

1 $S' := \emptyset$;
2 $X' := \emptyset$;
3 **foreach** $s \in S$ s.t. $c(s) = 0$ **do** $S' := S' \cup \{s\}$;
4 **repeat**
5 **foreach** $x \in X \setminus X'$ s.t. $c(s', x) = 0$ and $s' \in S'$ **do** $X' := X' \cup \{x\}$;
6 Build an α-list L w.r.t. (S', X');
7 $\hat{T} := \arg\max_{T \in L} \frac{f(X' \cup T) - f(X')}{c_{\min}(T)}$;
8 **if** $c(S' \cup \{s_{\min}(\hat{T})\}, X' \cup \hat{T}) \leq k$ **then**
9 $S' := S' \cup \{s_{\min}(\hat{T})\}$;
10 $X' := X' \cup \hat{T}$;
11 **until** $c(S' \cup \{s_{\min}(\hat{T})\}, X' \cup \hat{T}) > k$ or $X' = X$;
12 **if** $f(\hat{T}) \geq f(X')$ **then**
13 $S' := S' \cup \{s_{\min}(\hat{T})\}$;
14 $X' := \hat{T}$;
15 **return** (S', X');

each iteration, it first adds all the elements that can be associated to S' with cost 0 (line 5). Then, it selects a subset $\hat{T}$ that maximizes the ratio between the marginal increment and the marginal cost amongst the elements of an α-list L. Here, we assume that we have an algorithm to compute an α-list L w.r.t. (S', X') (see line 6). In the next sections, we will show how to compute L in polynomial time for some bounded α. The algorithm stops when adding the element with the maximum ratio would exceed the budget k or when $X' = X$. Without loss of generality, we can assume that at each iteration, the sets in the α-list L do not contain any element in X', since such elements do not increase the value of the marginal increment and possibly increase the marginal cost. This implies that at each iteration of the greedy procedure at least a new element in X is added to X' and then the number of iterations is $O(n)$.

Let (S_G, X_G) be the first candidate solution computed at the end of the greedy procedure. The second candidate solution (lines 12–14) is computed by using the set $\hat{T}$ that is discarded in the last iteration of the greedy procedure because adding $\{s_{\min}(\hat{T})\}$ and $\hat{T}$ to (S_G, X_G) would exceed the budget. Indeed, the second candidate solution is $(S_G \cup \{s_{\min}(\hat{T})\}, \hat{T})$. Note that this solution is feasible because $\hat{T}$ is contained in the α-list L computed in the last iteration of the greedy algorithm. Therefore, by definition of α-list, $c(S_G \cup \{s_{\min}(\hat{T})\}, \hat{T}) \leq k$.

The algorithm returns one of the two candidate solutions that maximizes the objective function.

The computational complexity of Algorithm 1 is $O(n \cdot (|L_{\max}| + cl))$, where $L_{\max}$ is the largest α-list computed and cl is the computational complexity of the algorithm at line 6. In the next sections we will show that our algorithms construct the α-lists in such a way that both $|L_{\max}|$ and cl are polynomially bounded in the input size.

In what follows we analyze the approximation ratio of Algorithm 1. The proof generalizes known arguments for monotone submodular maximization, see e.g. [7, 18, 23].

We give some additional definitions that will be used in the proof. We denote an optimal solution by (S^*, X^*). Let us consider the iterations executed by the greedy algorithm. Let $l + 1$ be the index of the iteration in which an element in the α-list is not added to X'

because it violates the budget constraint[2]. For $i = 1, 2, \ldots, l$, we define X'_i and S'_i as the sets X' and S' at the end of the i-th iteration of the algorithm, respectively. Moreover, let $X'_{l+1} = X'_l \cup \{\hat{T}\}$ and $S'_{l+1} = S'_l \cup \{s_{\min}(\hat{T})\}$, where $\hat{T}$ is the element selected at line 7 of iteration $l+1$ (see Algorithm 1). Let c_i be the value of $c_{\min}(\hat{T})$ as computed at iteration i of the greedy algorithm. The next lemma will be used in the proof of Theorem 2.

▶ **Lemma 1.** *After each iteration $i = 1, 2, \ldots, l+1$,*

$$f(X'_i) \geq \left(1 - \prod_{j=1}^{i}\left(1 - \alpha\frac{c_j}{k}\right)\right) f(X^*).$$

Armed with Lemma 1, we can prove Theorem 2.

▶ **Theorem 2.** *Algorithm 1 guarantees an approximation factor of $\frac{1}{2}\left(1 - \frac{1}{e^\alpha}\right)$ for GBSM.*

Proof. We observe that since (S'_{l+1}, X'_{l+1}) violates the budget, then $c(S'_{l+1}, X'_{l+1}) > k$. Moreover, for a sequence of numbers $a_1, a_2, \ldots, a_n$ such that $\sum_{\ell=1}^{n} a_\ell = A$, the function $\left[1 - \prod_{i=1}^{n}\left(1 - \frac{a_i \cdot \alpha}{A}\right)\right]$ achieves its minimum when $a_i = \frac{A}{n}$ and that $\left[1 - \prod_{i=1}^{n}\left(1 - \frac{a_i \cdot \alpha}{A}\right)\right] \geq 1 - \left(1 - \frac{\alpha}{n}\right)^n \geq 1 - e^{-\alpha}$. Therefore, by applying Lemma 1 for $i = l+1$ and observing that $\sum_{\ell=1}^{l+1} c_\ell = c(S'_{l+1}, X'_{l+1})$, we obtain:

$$f(X'_{l+1}) \geq \left[1 - \prod_{\ell=1}^{l+1}\left(1 - \frac{c_\ell \cdot \alpha}{k}\right)\right] f(X^*) \tag{1}$$

$$> \left[1 - \prod_{\ell=1}^{l+1}\left(1 - \frac{c_\ell \cdot \alpha}{c(S'_{l+1}, X'_{l+1})}\right)\right] f(X^*) \tag{2}$$

$$\geq \left[1 - \left(1 - \frac{\alpha}{(l+1)}\right)^{l+1}\right] f(X^*) \geq \left(1 - \frac{1}{e^\alpha}\right) f(X^*). \tag{3}$$

Since, by submodularity, $f(X'_{l+1}) \leq f(X'_l) + f(\hat{T})$, where $\hat{T}$ is the set selected at iteration $l+1$, we get

$$f(X'_l) + f(\hat{T}) \geq \left(1 - \frac{1}{e^\alpha}\right) f(X^*).$$

Hence, $\max\{f(X'_l), f(\hat{T})\} \geq \frac{1}{2}\left(1 - \frac{1}{e^\alpha}\right) f(X^*)$. The theorem follows by observing that $\hat{T}$ is the set selected as the second candidate solution at lines 12–14 of Algorithm 1. ◀

4 Computing an α-list for a particular case

In this section, we give a polynomial time algorithm to find a $(1-\epsilon)$-list with respect to a partial solution (S', X') for the particular case in which, for a given parameter $\epsilon \in (0, 1)$, the following condition holds:

$$\sum_{x \in T} c(s, x) \geq \frac{1}{\epsilon} c(s), \tag{4}$$

[2] We can assume that this iteration exists, as otherwise the algorithm is able to select $X' = X$, which is the optimum.

for each $s \in S$ and for each $T \subseteq X$ such that $|T| = \frac{1}{\epsilon}$. We observe that this condition is fulfilled for any $\epsilon \in (0,1)$ in the case in which $c(s) = 1$ and $c(s,x) \geq 1$, for each $s \in S$ and for each $x \in X$, which generalizes the non-stochastic adaptive seeding problem [1]. Indeed, in this case $\sum_{x \in T} c(s,x) \geq |T| = \frac{1}{\epsilon} c(s)$, for each $s \in S$ and for each $T \subseteq X$, such that $|T| = \frac{1}{\epsilon}$.

We give a simple algorithm that returns a $(1-\epsilon)$-list with respect to a partial solution (S', X'). The algorithm works as follows: build a list which contains all the subsets T of $X \setminus X'$ such that $|T| \leq \frac{1}{\epsilon}$ and $c(S' \cup \{s_{\min}(\hat{T})\}, \hat{T}) \leq k$.

Plugging this algorithm into line 6 of Algorithm 1, we can guarantee an approximation factor of $\frac{1}{2}\left(1 - \frac{1}{e}\right) - \epsilon'$, where $\epsilon' = \frac{1}{2e}\left(e^{\epsilon} - 1\right)$ for GBSM.

We observe that the case in this section contains the problem of maximizing a submodular set function under a knapsack constraint as a special case. Indeed, it is enough to set $c(s) = 0$, for each $s \in S$, and $c(s_1, x) = c(s_2, x)$, for each $s_1, s_2 \in S$ and $x \in X$. Note that in this case Condition 4 is satisfied for any $\epsilon \in (0,1)$. A special case of submodular set function maximization is the maximum coverage problem, and since this latter is NP-hard to be approximated within a factor greater than $\left(1 - \frac{1}{e}\right)$ [11], then the gap between the approximation factor of our algorithm and the best achievable one in polynomial time is $\frac{1}{2}$, unless $P = NP$.

It is easy to see that the computational complexity required by the algorithm in this section is $O(n^{\frac{1}{\epsilon}})$ and that $|L_{\max}| = O(n^{\frac{1}{\epsilon}})$.

In what follows, we assume that any set T^* that maximizes the ratio between marginal increment and marginal cost has size greater than $\frac{1}{\epsilon}$, as otherwise the α-list returned by our algorithm would contain such set. The following two technical lemmata will be used in the analysis of the algorithm.

▶ **Lemma 3.** *Given a monotone submodular set function $f : 2^X \to \mathbb{R}_{\geq 0}$, then, for any $X' \subseteq X$, the function $g(T) = f(X' \cup T) - f(X')$ is monotone and submodular.*

▶ **Lemma 4.** *Let us consider a monotone submodular set function $f : 2^X \to \mathbb{R}_{\geq 0}$ and a cost function $c : 2^X \to \mathbb{R}_{\geq 0}$ such that $c(T) = \sum_{x \in T} c(\{x\})$, for each $T \subseteq X$. For each set $T \subseteq X$, if T_y denotes the subset of T such that $\frac{f(T_y)}{c(T_y)}$ is maximum and $|T_y| = y$, then $\frac{f(T)}{c(T)} \leq \frac{f(T_y)}{c(T_y)}$, for any $y \leq |T|$.*

The next theorem shows the approximation ratio of the algorithm. The main idea is to consider the subset $\hat{T}$ that maximizes the ratio between the marginal increment and marginal cost in L and to derive a series of inequalities to lead us state that this value is greater than the ratio given by the optimal subset T^* times the factor $(1-\epsilon)$. We first compare the ratio computed for $\hat{T}$ with that for $T^*_{\frac{1}{\epsilon}}$ that is a subset of cardinality $\frac{1}{\epsilon}$ of maximal ratio, then, by rewriting the marginal cost formula according to its definition and by exploiting Lemmata 3 and 4, and Condition (4) we compare this ratio to that given by the subset T^* and this last inequality concludes the theorem.

▶ **Theorem 5.** *If for each $T \subseteq X$ such that $|T| = \frac{1}{\epsilon}$ and for each $s \in S$ we have $\sum_{x \in T} c(s,x) \geq \frac{1}{\epsilon} c(s)$, then the list L made of all the subsets of $X \setminus X'$ of size at most $\frac{1}{\epsilon}$ and cost at most k is a $(1-\epsilon)$-list.*

Proof. We recall that $g(T) = f(X' \cup T) - f(X')$. Given a subset T of $X \setminus X'$, we denote by T_y a subset of T such that $|T_y| = y$ and $\frac{f(T_y)}{c(T_y)}$ is maximum. Let T^* be the subset of $X \setminus X'$ that maximizes the ratio between the marginal increment and the marginal cost. Let $\hat{T}$ be

the element of L that maximizes $\frac{g(\hat{T})}{c_{\min}(\hat{T})}$. Since $|\hat{T}| \leq \frac{1}{\epsilon}$, then

$$\frac{g(\hat{T})}{c_{\min}(\hat{T})} \geq \frac{g\left(T^*_{\frac{1}{\epsilon}}\right)}{c_{\min}\left(T^*_{\frac{1}{\epsilon}}\right)} = \frac{g\left(T^*_{\frac{1}{\epsilon}}\right)}{c_{S'}\left(s_{\min}\left(T^*_{\frac{1}{\epsilon}}\right)\right) + \bar{c}\left(T^*_{\frac{1}{\epsilon}}\right)}.$$

By the hypothesis of the theorem, $\bar{c}\left(T^*_{\frac{1}{\epsilon}}\right) \geq \frac{1}{\epsilon} c\left(s_{\min}\left(T^*_{\frac{1}{\epsilon}}\right)\right)$, moreover, $c\left(s_{\min}\left(T^*_{\frac{1}{\epsilon}}\right)\right) \geq c_{S'}\left(s_{\min}\left(T^*_{\frac{1}{\epsilon}}\right)\right)$ and then $c_{S'}\left(s_{\min}\left(T^*_{\frac{1}{\epsilon}}\right)\right) \leq \epsilon \bar{c}\left(T^*_{\frac{1}{\epsilon}}\right)$. Therefore,

$$\frac{g\left(T^*_{\frac{1}{\epsilon}}\right)}{c_{S'}\left(s_{\min}\left(T^*_{\frac{1}{\epsilon}}\right)\right) + \bar{c}\left(T^*_{\frac{1}{\epsilon}}\right)} \geq \frac{g\left(T^*_{\frac{1}{\epsilon}}\right)}{\epsilon\bar{c}\left(T^*_{\frac{1}{\epsilon}}\right) + \bar{c}\left(T^*_{\frac{1}{\epsilon}}\right)} = \frac{g\left(T^*_{\frac{1}{\epsilon}}\right)}{(\epsilon+1)\bar{c}\left(T^*_{\frac{1}{\epsilon}}\right)}.$$

Since f is monotone and submodular, then, by Lemma 3, also $g\left(T^*_{\frac{1}{\epsilon}}\right)$ is submodular. By Lemma 4 follows that

$$\frac{g\left(T^*_{\frac{1}{\epsilon}}\right)}{(\epsilon+1)\bar{c}\left(T^*_{\frac{1}{\epsilon}}\right)} \geq \frac{g(T^*)}{(\epsilon+1)\bar{c}(T^*)}.$$

We now focus on the denominator, and we obtain that:

$$\frac{1}{(\epsilon+1)\bar{c}(T^*)} = \frac{1+\epsilon-\epsilon}{(\epsilon+1)\bar{c}(T^*)} = \frac{1}{\bar{c}(T^*)} - \frac{\epsilon}{(\epsilon+1)\bar{c}(T^*)} \geq \frac{1}{\bar{c}(T^*) + c_{S'}(s_{\min}(T^*))} - \frac{\epsilon}{\epsilon\bar{c}(T^*) + \bar{c}(T^*)}.$$

By applying the hypothesis $\bar{c}(T^*) \geq \frac{1}{\epsilon} c(s_{\min}(T^*))$, it follows that:

$$\frac{1}{\bar{c}(T^*) + c_{S'}(s_{\min}(T^*))} - \frac{\epsilon}{\epsilon\bar{c}(T^*) + \bar{c}(T^*)} \geq \frac{1}{\bar{c}(T^*) + c_{S'}(s_{\min}(T^*))} - \frac{\epsilon}{c(s_{\min}(T^*)) + \bar{c}(T^*)} \geq \frac{1}{\bar{c}(T^*) + c_{S'}(s_{\min}(T^*))} - \frac{\epsilon}{\bar{c}(T^*) + c_{S'}(s_{\min}(T^*))} = \frac{1-\epsilon}{c_{\min}(T^*)}.$$

To conclude:

$$\frac{g(\hat{T})}{c_{\min}(\hat{T})} \geq (1-\epsilon)\frac{g(T^*)}{c_{\min}(T^*)}.$$ ◀

5 Computing an α-list for the general case

In this section we give a polynomial time algorithm that builds a $\left(1-\frac{1}{e}\right)(1-\epsilon)$-list with respect to a partial solution (S', X'), for any $\epsilon \in (0,1)$. Using this algorithm as routine at line 6 of Algorithm 1, we can guarantee an approximation factor of

$$\frac{1}{2}\left(1 - \frac{1}{e^{\left(1-\frac{1}{e}\right)(1-\epsilon)}}\right)$$

for GBSM. We observe that this case generalizes the non-stochastic adaptive seeding with knapsack constraints problem, which cannot be approximated within a factor greater than

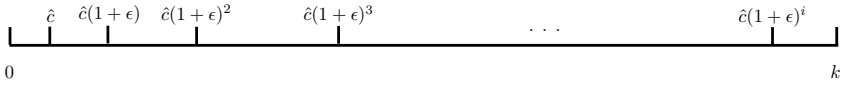


Figure 1 Growth of the budget B_i in the inner cycle of the algorithm.

$\left(1-\frac{1}{e^{1-\frac{1}{e}}}\right)$, unless $P=NP$ [21]. Then, the gap between the approximation factor of our algorithm and the best achievable one in polynomial time is $\frac{1}{2}$, unless $P=NP$.

In the algorithm of this section we make use of a procedure called `GreedyMaxCover` to maximize the value of a monotone submodular function $g: 2^X \to \mathbb{R}_{\geq 0}$, given a certain budget and costs associated to the elements of X. It is well-known that there exists a polynomial-time procedure that guarantees a $\left(1-\frac{1}{e}\right)$-approximation for this problem [23].

Let us denote by $\hat{c}$ the minimum possible positive value of functions $c(s)$ and $c(s,x)$, amongst all elements x and bins s, i.e. $\hat{c}=\min\{\min\{c(s): s\in S, c(s)>0\}, \min\{c(s,x): s\in S, x\in X, c(s,x)>0\}\}$.

The main idea is to build an α-list L which contains approximate solutions to the problem of maximizing a monotone submodular set function subject to a knapsack constraint in which the budget increases by a factor $1+\epsilon$ starting from $\hat{c}$, and the cost of the elements are given by the cost of associating them to a single bin. In particular, we consider $q=\left\lfloor \log_{1+\epsilon}\left(\frac{k}{\hat{c}}\right)\right\rfloor+1$ different budgets B_i that iteratively increase by a factor $1+\epsilon$, i.e. $B_0=\hat{c}$ and $B_i=(1+\epsilon)B_{i-1}$, for $i=1,\ldots,q$. Moreover we define $B_{q+1}=k$. For each $i=0,\ldots,q+1$ and for each bin $s\in S$, we apply procedure `GreedyMaxCover` with ground set X, budget B_i, and the cost of associating the elements to bin s as cost function. Then, we add the set returned by `GreedyMaxCover` to L. In this way we consider a budget that is at most a factor $1+\epsilon$ greater than the cost of an optimal solution and the solution returned by `GreedyMaxCover` for this budget has a value that is at most $1-\frac{1}{e}$ times smaller than that of the optimal solution.

The pseudo-code of the algorithm is reported in Algorithm 2. The outer cycle at lines 2–11 iteratively selects a bin s in S and finds a list of sets of elements assigned to bin s. The inner cycle at lines 4–8, at each iteration i, calls procedure `GreedyMaxCover` which uses g as function to maximize, $\hat{c}(1+\epsilon)^i - c_{S'}(s)$ as budget and the cost of associating the elements to bin s as cost function (to compute this costs, we only pass s as a parameter to `GreedyMaxCover`). The budget is increased by a factor $(1+\epsilon)$ until $\hat{c}(1+\epsilon)^i \geq k$. Finally the algorithm runs `GreedyMaxCover` with the full budget k. See Figure 1 for an illustration.

We call q the value of i at the end of the last iteration in the inner cycle of the algorithm. Let T_j be the set in L that maximizes the ratio between $g(T_j)$ and its assigned budget, that is:

$$T_j = \arg\max\left\{\frac{g(T_i(s))}{B_i} : s\in S, i=0,1,\ldots,q+1\right\}. \tag{5}$$

In order to bound the approximation ratio, we consider $\bar{X}^*$ as the set with the optimal ratio $\frac{g(\bar{X}^*)}{c_{\min}(\bar{X}^*)}$ amongst any possible subset of items. Let B_l be the smallest value of B_i, for $i\in\{0,1,\ldots,q+1\}$, that is greater than or equal to the cost of an optimal solution, that is the smallest B_l such that $B_l \geq c_{\min}(\bar{X}^*)$. See figure 2 for an illustration. We call T_l^* the set in L that has the highest ratio $\frac{g(T_l)}{B_l}$ amongst those computed by `GreedyMaxCover` with budget B_l, i.e. $T_l^* = \max\left\{\frac{g(T_l(s))}{B_l} : s\in S\right\}$. We also denote the set that maximizes $g(X_l^*)$ with budget B_l by X_l^*.

The idea of the approximation analysis is that an optimal solution $\bar{X}^*$ has a value of g that is at most $g(X_l^*)$ and a cost that is at most $1+\epsilon$ times smaller than B_l, while the number of iterations remains polynomial since the size of the intervals grows exponentially.

Algorithm 2: Exponential Budget Greedy.

Input : $S, X, S', X', k, \epsilon$
Output : L

$L := \emptyset$;
foreach $s \in S$ **do**
  $i := 0$;
  while $\hat{c}(1+\epsilon)^i < k$ **do**
    $B_i := \hat{c}(1+\epsilon)^i$;
    $T_i(s) := \texttt{GreedyMaxCover}(X, s, B_i - c_{S'}(s))$;
    $L := L \cup \{T_i(s)\}$;
    $i := i + 1$
  $B_i := k$;
  $T_i(s) := \texttt{GreedyMaxCover}(X, s, B_i - c_{S'}(s))$;
  $L := L \cup \{T_i(s)\}$;
return L;

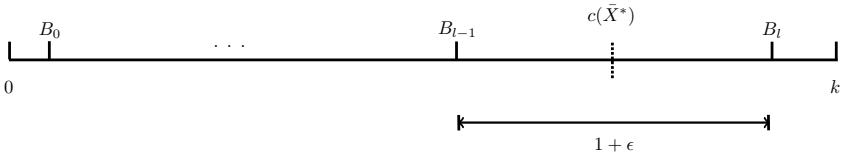


Figure 2 Notation used in Theorem 6.

The next two theorems show the bounds on approximation ratio, computational complexity and size of L.

▶ **Theorem 6.** *The list L built by Algorithm 2, is a $\left(1-\frac{1}{e}\right)(1-\epsilon)$-list.*

Proof. Since, by construction, $c_{\min}(T_j) \leq B_j$, and, by Equation 5, $\frac{g(T_j)}{B_j}$ is maximum, then

$$\frac{g(T_j)}{c_{\min}(T_j)} \geq \frac{g(T_j)}{B_j} \geq \frac{g(T_l^*)}{B_l}.$$

Procedure `GreedyMaxCover` guarantees a $\left(1-\frac{1}{e}\right)$-approximation, then

$$\frac{g(T_l^*)}{B_l} \geq \left(1-\frac{1}{e}\right)\frac{g(X_l^*)}{B_l}.$$

Moreover, since function g is monotone and $c_{\min}(\bar{X}^*) \leq B_l$, then $g(X_l^*) \geq g(\bar{X}^*)$, and therefore:

$$\left(1-\frac{1}{e}\right)\frac{g(X_l^*)}{B_l} \geq \left(1-\frac{1}{e}\right)\frac{g(\bar{X}^*)}{B_l}.$$

We defined B_l as the smallest value of B_i that is at least $c_{\min}(\bar{X}^*)$, this implies that $B_{l-1} \leq c_{\min}(\bar{X}^*)$. Moreover the ratio between B_l and B_{l-1} is $1+\epsilon$. It follows that $B_l \leq (1+\epsilon)c_{\min}(\bar{X}^*)$, which implies:

$$\left(1-\frac{1}{e}\right)\frac{g(\bar{X}^*)}{B_l} \geq \left(1-\frac{1}{e}\right)\left(\frac{1}{1+\epsilon}\right)\frac{g(\bar{X}^*)}{c_{\min}(\bar{X}^*)} \geq \left(1-\frac{1}{e}\right)(1-\epsilon)\frac{g(\bar{X}^*)}{c_{\min}(\bar{X}^*)}$$

Algorithm 3: Bi-criterion Algorithm.

Input : S, X
Output : S', X'

1 $S' := \emptyset$;
2 $X' := \emptyset$;
3 **foreach** $s \in S$ s.t. $c(s) = 0$ **do** $S' := S' \cup \{s\}$;
4 **repeat**
5 **foreach** $x \in X \setminus X'$ s.t. $c(s', x) = 0$ and $s' \in S'$ **do** $X' := X' \cup \{x\}$;
6 Build an α-list L w.r.t. (S', X');
7 $\hat{T} := \arg\max_{T \in L} \frac{f(X' \cup T) - f(X')}{c_{\min}(T)}$;
8 **if** $c(S' \cup \{s_{\min}(\hat{T})\}, X' \cup \hat{T}) \leq \beta k$ **then**
9 $S' := S' \cup \{s_{\min}(\hat{T})\}$;
10 $X' := X' \cup \hat{T}$;
11 **until** $c(S' \cup \{s_{\min}(\hat{T})\}, X' \cup \hat{T}) > \beta k$ or $X' = X$;
12 **if** $f(\hat{T}) \geq f(X')$ **then**
13 $S' := S' \cup \{s_{\min}(\hat{T})\}$;
14 $X' := \hat{T}$;
15 **return** (S', X');

The last inequality holds since $\frac{1}{1+\epsilon} = 1 - \frac{\epsilon}{1+\epsilon} \geq 1 - \epsilon$, for any $\epsilon > 0$, and this concludes the proof. ◀

▶ **Theorem 7.** *Algorithm 2 requires* $O\left(\frac{1}{\epsilon} m \cdot gr(n) \cdot \log \frac{k}{\hat{c}}\right)$ *computational time, where* $gr(n)$ *is the computational time of* `GreedyMaxCover`*, and* $|L_{\max}| = O\left(\frac{1}{\epsilon} m \log \frac{k}{\hat{c}}\right)$.

Proof. The outer for cycle requires m iterations. We now bound the number q of iteration of the inner cycle of the algorithm. By the exit condition of the cycle, we have: $\hat{c} \cdot (1+\epsilon)^q < k$, which is equivalent to: $q < \log_{1+\epsilon}\left(\frac{k}{\hat{c}}\right)$. Since for $\epsilon < 1$, $\log_{1+\epsilon}\left(\frac{k}{\hat{c}}\right) = O\left(\frac{1}{\epsilon} \log \frac{k}{\hat{c}}\right)$, the statement follows. ◀

We observe that $O(\log \frac{k}{\hat{c}})$ is polynomially bounded in the size of the input.

6 Bi-criterion approximation algorithm

In this section we extend the results given in Section 3 providing a bi-criterion approximation algorithm that guarantees a $\frac{1}{2}\left(1 - \frac{1}{e^{\alpha\beta}}\right)$-approximation to the GBSM problem, if we allow an extra budget up to a factor $\beta \geq 1$. We notice that, if $\beta = 1$, i.e. we do not increase the budget, the approximation factor is $\frac{1}{2}\left(1 - \frac{1}{e^{\alpha}}\right)$, while if $\beta = \frac{1}{\alpha} \ln\left(\frac{1}{2\epsilon}\right)$ the algorithm achieves an approximation factor of $\frac{1}{2} - \epsilon$, for any arbitrarily small $\epsilon > 0$.

The algorithm is slightly different from Algorithm 1 and it is reported in Algorithm 3. In this algorithm, we allow to exceed the given budget k by a factor β. In particular we modify lines 8 and 11, admitting a greater budget respect to Algorithm 1.

In the next theorem we show the approximation ratio of this algorithm.

▶ **Theorem 8.** *There exists an algorithm that guarantees a* $\left[\frac{1}{2}\left(1 - \frac{1}{e^{\alpha\beta}}\right), \beta\right]$ *bi-criterion approximation for GBSM, for any* $\beta \geq 1$.

It is possible to prove the theorem by using the same argument as in Theorem 2, by taking into account the new budget βk. We also exploit the fact that Lemma 1 also holds when considering the budget βk.

7 Conclusion

In this paper we defined a new challenging problem which leads to many open problems and new research questions, we referred to it as the generalized budgeted submodular set function maximization problem.

The main open problem is to close the gap between the known hardness result of $1 - \frac{1}{e^\alpha}$, where $\alpha = 1$ for the MC problem [11] and $\alpha = 1 - \frac{1}{e}$ for the non-stochastic adaptive seeding problem with knapsack constraint problem [21], and our approximation bound of $\frac{1}{2}\left(1 - \frac{1}{e^\alpha}\right)$. One possibility to get rid of the $\frac{1}{2}$ factor could be to use the partial enumeration technique exploited in specific subproblems (e.g. budgeted maximum coverage problem [18] and monotone submodular set function subject to a knapsack constraint maximization problem [23]). However, this requires that each greedy step selects a single element of X, to be added to a partial solution X', while our greedy algorithm selects a subset of $X \setminus X'$ that maximizes the ratio between its marginal increment in the objective function and its marginal cost. Note that this set can contain more than one element in order to ensure that the ratio is non-increasing at each iteration of the greedy algorithm, which is needed to apply the analysis in [18] and [23].

Other research directions, that deserve further investigation, include the study of the GBSM considering different cost functions and also different objective functions where the profit given by an element x depends on the bin s which it is associated with. It would be interesting also to analyse GBSM in the case that each bin $s \in S$ has its own budget k to use in order to maximize the objective function.

References

1 Ashwinkumar Badanidiyuru, Christos H. Papadimitriou, Aviad Rubinstein, Lior Seeman, and Yaron Singer. Locally adaptive optimization: Adaptive seeding for monotone submodular functions. In *27th ACM-SIAM Symp. on Disc. Alg., (SODA)*, pages 414–429, 2016.

2 Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. In *53rd IEEE Symp. on Foundations of Computer Science, FOCS*, pages 649–658, 2012.

3 Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011.

4 Ioannis Caragiannis. Wavelength management in WDM rings to maximize the number of connections. *SIAM J. Discrete Math.*, 23(2):959–978, 2009.

5 Ioannis Caragiannis and Gianpiero Monaco. A 6/5-approximation algorithm for the maximum 3-cover problem. *J. Comb. Optim.*, 25(1):60–77, 2013.

6 Chandra Chekuri and Amit Kumar. Maximum coverage problem with group budget constraints and applications. In *7th Intl. Work. on Approximation Algorithms for Combinatorial Optimization Problems, APPROX*, pages 72–83, 2004.

7 Reuven Cohen and Liran Katzir. The generalized maximum coverage problem. *Information Processing Letters*, 108(1):15–22, 2008.

8 Gianlorenzo D'Angelo, Lorenzo Severini, and Yllka Velaj. Influence maximization in the independent cascade model. In *Proceedings of the 17th Italian Conference on Theoretical Computer Science (ICTCS2016)*, volume 1720, pages 269–274. CEUR-WS.org, 2016.

9 Gianlorenzo D'Angelo, Lorenzo Severini, and Yllka Velaj. Selecting nodes and buying links to maximize the information diffusion in a network. In *42st Intl. Symp. on Mathematical Foundations of Computer Science, MFCS*, volume 83 of *LIPIcs*, pages 75:1–75:14, 2017.

10 Gianlorenzo D'Angelo, Lorenzo Severini, and Yllka Velaj. Recommending links through influence maximization. *Theoretical Computer Science*, 2018. doi:10.1016/j.tcs.2018.01.017.

11 Uriel Feige. A threshold of ln n for approximating set cover. *Journal of ACM*, 45(4):634–652, 1998.

12 Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. In *48th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 461–471, 2007.

13 Yuval Filmus and Justin Ward. Monotone submodular maximization over a matroid via non-oblivious local search. *SIAM J. Comput.*, 43(2):514–542, 2014.

14 D.S. Hochbaum. *Approximation Algorithms for NPHard Problems.* PWS Publishing Company,, Boston, MA, USA, 1997.

15 Rishabh K. Iyer and Jeff A. Bilmes. Submodular optimization with submodular cover and submodular knapsack constraints. In *27th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 2436–2444, 2013.

16 V Kann. Maximum bounded 3-dimensional matching is max snp-comple. *Information Processing Letters*, 37:27–35, 1991.

17 David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. *Theory of Computing*, 11:105–147, 2015.

18 Samir Khuller, Anna Moss, and Joseph Seffi Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

19 Jon Lee, Maxim Sviridenko, and Jan Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. *Math. Oper. Res.*, 35(4):795–806, 2010.

20 G.L. Nemhauser, L.A. Wolsey, and M.L. Fisher. An analysis of approximations for maximizing submodular set functions–I. *Mathematical Programming*, 14(1):265–294, 1978.

21 Aviad Rubinstein, Lior Seeman, and Yaron Singer. Approximability of adaptive seeding under knapsack constraints. In *16th ACM Conf. on Economics and Computation*, pages 797–814. ACM, 2015.

22 Lior Seeman and Yaron Singer. Adaptive seeding in social networks. In *IEEE 54th Symp. on Foundations of Computer Science (FOCS)*, pages 459–468. IEEE, 2013.

23 Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operation Research Letters*, 32(1):41–43, 2004.

24 Irving van Heuven van Staereling, Bart de Keijzer, and Guido Schäfer. The Ground-Set-Cost Budgeted Maximum Coverage Problem. In *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58 of *LIPIcs*, pages 50:1–50:13, 2016.

25 Jan Vondrák, Chandra Chekuri, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. In *43rd ACM Symp. on Theory of Computing, STOC*, pages 783–792, 2011.

26 Justin Ward. A (k+3)/2-approximation algorithm for monotone submodular k-set packing and general k-exchange systems. In *29th Intl. Symp.on Theoretical Aspects of Computer Science, STACS*, pages 42–53, 2012.

Complexity of Preimage Problems for Deterministic Finite Automata

Mikhail V. Berlinkov[1]
Institute of Natural Sciences and Mathematics, Ural Federal University, Ekaterinburg, Russia
berlm@mail.ru

Robert Ferens[2]
Institute of Computer Science, University of Wrocław, Wrocław, Poland
robert.ferens@interia.pl

Marek Szykuła[3]
Institute of Computer Science, University of Wrocław, Wrocław, Poland
msz@cs.uni.wroc.pl

Abstract

Given a subset of states S of a deterministic finite automaton and a word w, the preimage is the subset of all states that are mapped to a state from S by the action of w. We study the computational complexity of three problems related to the existence of words yielding certain preimages, which are especially motivated by the theory of synchronizing automata. The first problem is whether, for a given subset, there exists a word extending the subset (giving a larger preimage). The second problem is whether there exists a word totally extending the subset (giving the whole set of states) – it is equivalent to the problem whether there exists an avoiding word for the complementary subset. The third problem is whether there exists a word resizing the subset (giving a preimage of a different size). We also consider the variants of the problem where an upper bound on the length of the word is given in the input. Because in most cases our problems are computationally hard, we additionally consider parametrized complexity by the size of the given subset. We focus on the most interesting cases that are the subclasses of strongly connected, synchronizing, and binary automata.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases avoiding word, extending word, extensible subset, reset word, synchronizing automaton

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.32

Related Version `https://arxiv.org/abs/1704.08233`

1 Introduction

A deterministic finite complete (semi)automaton $\mathscr{A}$ is a triple (Q, Σ, δ), where Q is the set of *states*, Σ is the input *alphabet*, and $\delta\colon Q \times \Sigma \to Q$ is the *transition function*. We extend δ to a function $Q \times \Sigma^* \to Q$ in the usual way. Throughout the paper, by n we always denote the number of states $|Q|$.

[1] Supported by Russian Foundation for Basic Research, grant no. 16-01-00795, and the Competitiveness Enhancement Program of Ural Federal University.
[2] Supported in part by the National Science Centre, Poland under project number 2014/15/B/ST6/00615.
[3] Supported in part by the National Science Centre, Poland under project number 2017/25/B/ST6/01920.

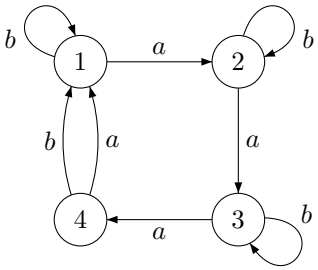


Figure 1 The Černý automaton with 4 states.

When the context automaton is clear, given a state $q \in Q$ and a word $w \in \Sigma^*$, we write shortly $q \cdot w$ for $\delta(q, w)$. Given a subset $S \subseteq Q$, the *image* of S under the action of a word $w \in \Sigma^*$ is $S \cdot w = \delta(S, w) = \{q \cdot w \mid q \in S\}$. The *preimage* is $S \cdot w^{-1} = \delta^{-1}(S, w) = \{q \in Q \mid q \cdot w \in S\}$. If $S = \{q\}$, then we usually simply write $q \cdot w^{-1}$.

We say that a word w *compresses* a subset S if $|S \cdot w| < |S|$, avoids S if $(Q \cdot w) \cap S = \emptyset$, *extends* S if $|S \cdot w^{-1}| > |S|$, and *totally extends* S if $S \cdot w^{-1} = Q$. A subset S is *compressible*, *avoidable*, *extensible*, and *totally extensible*, if there is a word that respectively compresses, avoids, extends and totally extends it.

▶ Remark. A word $w \in \Sigma^*$ is avoiding for $S \subseteq Q$ if and only if w is totally extending for $Q \setminus S$.

Fig. 1 shows an example automaton. For $S = \{2, 3\}$, the shortest compressing word is aab, and we have $\{2, 3\} \cdot aab = \{1\}$, while the shortest extending word is ba, and we have $\{2, 3\} \cdot (ba)^{-1} = \{1, 2\} \cdot b^{-1} = \{1, 2, 4\}$.

In fact, the preimage of a subset under the action of a word can be smaller than the subset. In this case, we say that a word *shrinks* the subset (not to be confused with compressing when the image is considered). For example, in Fig. 1, subset $\{1, 4\}$ is shrank by b to subset $\{4\}$.

Note that shrinking a subset is equivalent to extending its complement. Similarly, a word totally extending a subset also shrinks its complement to the empty set.

▶ Remark. $|S \cdot w^{-1}| > |S|$ if and only if $|(Q \setminus S) \cdot w^{-1}| < |Q \setminus S|$, and $S \cdot w^{-1} = Q$ if and only if $(Q \setminus S) \cdot w^{-1} = \emptyset$.

Therefore, avoiding a subset is equivalent to shrinking it to the empty set.

The *rank* of a word w is the cardinality of the image $Q \cdot w$. A word of rank 1 is called *reset* or *synchronizing*, and an automaton that admits a reset word is called *synchronizing*. Also, for a subset $S \subseteq Q$, we say that a word $w \in \Sigma^*$ such that $|S \cdot w| = 1$ *synchronizes* S.

Synchronizing automata serve as transparent and natural models of various systems in many applications in different fields, such as coding theory, DNA-computing, robotics, testing of reactive systems, and theory of information sources. They also reveal interesting connections with symbolic dynamics, language theory, group theory, and many other parts of mathematics. For a detailed introduction to the theory of synchronizing automata we refer the reader to the survey [31], and for a review of relations with coding theory to [16] and [8].

The famous Černý conjecture [11], which was formally stated in 1969 during a conference ([31]), is one of the most longstanding open problems in automata theory, and is the central problem in the theory of synchronizing automata. It states that a synchronizing automaton has a reset word of length at most $(n-1)^2$. Besides the conjecture, algorithmic issues are also important. Unfortunately, the problem of finding a *shortest* reset word is computationally hard [12, 21], and also its length approximation remains hard [13]. We also refer to surveys [24, 31] about algorithmic issues and the Černý conjecture.

Our general motivation comes from the fact that words compressing and extending subsets play a crucial role in synchronization automata. In fact, all known algorithms finding a reset word as intermediate steps use finding words that either compresses or extends a subset (e.g. [1, 7, 12, 18, 22]). Moreover, probably all proofs of upper bounds on the length of the shortest reset words use bounding the length of words that compress (e.g. [2, 7, 9, 12, 14, 27, 29, 32]) or extend (e.g. [3, 4, 7, 17, 26, 27]) some subsets.

In this paper, we study several natural problems related to preimages. Our goal is to provide a systematic view of their computational complexity and solve several open problems.

1.1 Compressing a Subset

The complexities of problems related to compressing a subset have been well studied.

It is known that given an automaton $\mathscr{A}$ and a subset $S \subseteq Q$, determining whether there is a word that synchronizes it is PSPACE-complete [23]. The same holds even for strongly connected binary automata [33].

On the other hand, checking whether the automaton is synchronizing (whether there is a word that synchronizes Q) can be solved in $\mathcal{O}(|\Sigma|n^2)$ time and space [11, 12, 31] and in $\mathcal{O}(n)$ average time and space for the random binary case [6]. To this end, we just verify whether all pairs of states are compressible. Using the same algorithm, we can determine whether a given subset is compressible.

Deciding whether there exists a synchronizing word of a given length is NP-complete [12] (cf. [21] for the complexity of the corresponding functional problems), even if the given automaton is binary. There exist stronger results, such as NP-completeness of this problem when the automaton is Eulerian and binary [34], which immediately implies that for the class of strongly connected automata the complexity is the same.

However, deciding whether there exists a word that only compresses a subset still can be solved in $\mathcal{O}(|\Sigma|n^2)$ time, as for every pair of states we can compute a shortest word that compresses the pair.

The problems have been also studied in other settings than DFAs. We refer to [20, 23] for the cases of NFA and PDFA (*partial deterministic finite automata*), and to [15] for the partial observability setting. Finally, in [10] the problem of reachability of a given subset in a DFA has been studied.

1.2 Extending a Subset and Our Contributions

In contrast to the problems related to images (compression), the complexity of the problems related to preimages has not been well studied. In the paper, we fill this gap. We study three families of problems. As we noted before, extending is equivalent to shrinking the complement, hence we deal only with the extending word problems.

Extending words: Our first family of problems is the question whether there exists an extending word (Problems 1, 7, 9, 13, 16, 22).

This is motivated by the fact that finding such a word is the basic step of the so-called *extension method* of finding a reset word that is used in many proofs and also some algorithms. The extension method of finding a reset word is to start from some singleton $S_0 = \{q\}$, and iteratively find extending words $w_1, \ldots, w_k$ such that $|S_0 \cdot w_1^{-1} \cdots w_i^{-1}| > |S_0 \cdot w_1^{-1} \cdots w_{i-1}^{-1}|$ for $1 \leq i \leq k$, and where final $S_0 \cdot w_1^{-1} \cdots w_k^{-1} = Q$. For finding a short reset word one needs to bound the lengths of the extending words. For instance, by showing that in the case of Eulerian automata there are always extending words of length at most n, which implies the upper bound $(n-2)(n-1)+1$ on the length of the shortest reset words for this class [17]. In this case, a polynomial algorithm for finding extending words has been proposed in [7].

Table 1 Computational complexity of decision problems in classes of automata. Given an automaton $\mathscr{A} = (Q, \Sigma, \delta)$ with n states and a subset $S \subseteq Q$, is there a word $w \in \Sigma^*$ such that:

Subclass of automata	All automata	Strongly connected	Synchronizing	Str. con. and synch.
$\lvert S \cdot w\rvert = 1$ (reset word)	PSPACE-c [23, 33]		$\mathcal{O}(1)$	$\mathcal{O}(1)$
$\lvert S \cdot w\rvert < \lvert S\rvert$ (compressing word)	$\mathcal{O}(\lvert\Sigma\rvert n^2)$ [11, 31]		$\mathcal{O}(1)$	$\mathcal{O}(1)$
$\lvert S \cdot w^{-1}\rvert > \lvert S\rvert$ (Problem 1)	PSPACE-c (Thm. 3)		PSPACE-c (Prop. 5)	$\mathcal{O}(1)$
$S \cdot w^{-1} = Q$ (Problem 2)	PSPACE-c (Thm. 3)		$\mathcal{O}(\lvert\Sigma\rvert n)$ (Thm. 6)	$\mathcal{O}(1)$
$\lvert S \cdot w^{-1}\rvert > \lvert S\rvert$, $\lvert S\rvert \leq k$ (Problem 9)	$\mathcal{O}(\lvert\Sigma\rvert n^k)$ (Prop. 10)		$\mathcal{O}(\lvert\Sigma\rvert n^k)$ (Prop. 10)	$\mathcal{O}(1)$
$S \cdot w^{-1} = Q$, $\lvert S\rvert \leq k$ (Problem 11)	$\mathcal{O}(\lvert\Sigma\rvert(n^3 + n^k))$ (Prop. 12)		$\mathcal{O}(\lvert\Sigma\rvert n)$ (Thm. 6)	$\mathcal{O}(1)$
$\lvert S \cdot w^{-1}\rvert > \lvert S\rvert$, $\lvert S\rvert \geq n - k$ (Problem 16, $k \geq 2$)	PSPACE-c (Thm. 19)	Open	PSPACE-c (Thm. 19)	$\mathcal{O}(1)$
$S \cdot w^{-1} = Q$, $\lvert S\rvert \geq n - k$ (Problem 17, $k \geq 2$)	$\mathcal{O}(n^3 + \lvert\Sigma\rvert n^k)$ (Thm. 21)		$\mathcal{O}(\lvert\Sigma\rvert n)$ (Thm. 6)	$\mathcal{O}(1)$
$S \cdot w^{-1} = Q$, $\lvert S\rvert = n - 1$ (Problem 18)	$\mathcal{O}(\lvert\Sigma\rvert n^2)$ (Thm. 20)		$\mathcal{O}(\lvert\Sigma\rvert)$	$\mathcal{O}(1)$
$\lvert S \cdot w^{-1}\rvert \neq \lvert S\rvert$ (Problem 27)	$\mathcal{O}(\lvert\Sigma\rvert n^3)$ (Thm. 29)		$\mathcal{O}(1)$	$\mathcal{O}(1)$

Totally extending words and avoiding: We study the problem whether there exists a totally extending word (Problems 2, 8, 11, 14, 17, 23). The question about the existence of a totally extending word is equivalent to the question about the existence of an avoiding word for the complementary subset.

Totally extending words themselves can be viewed as a generalization of reset words: a word totally extending a singleton to the whole set of states Q is a reset word. If we are not interested in bringing the automaton into one particular state but want it to be in any of the states from a specified subset, then it is exactly the question about totally extending word for our subset. In view of applications of synchronization, this can be particularly useful when we deal with non-synchronizing automata, where reset words cannot be applied.

Avoiding word problem is a recent concept that is dual to synchronization: instead of being in some states, we want to not be in them. A quadratic upper bound on the length of the shortest avoiding words of a single state have been established in [27], where avoiding words were also used to improve the best known upper bound on the length of the shortest reset words. The computational complexity of the problems related to avoiding, both a single state or a subset, have not been established, which is another motivation to study

Table 2 Computational complexity of decision problems in classes of automata. Given an automaton $\mathscr{A} = (Q, \Sigma, \delta)$ with n states, a subset $S \subseteq Q$, and an integer ℓ given in binary form, is there are a word $w \in \Sigma^*$ of length $\leq \ell$ such that:

Subclass of automata	All automata	Strongly connected	Synchronizing	Str. con. and synch.
$\lvert S \cdot w\rvert = 1$ (reset word)	PSPACE-c [23, 33]		NP-c [12]	NP-c [34]
$\lvert S \cdot w\rvert < \lvert S\rvert$ (compressing word)	$\mathcal{O}(\lvert\Sigma\rvert n^2)$ [12]		$\mathcal{O}(\lvert\Sigma\rvert n^2)$ [12]	$\mathcal{O}(\lvert\Sigma\rvert n^2)$ [12]
$\lvert S \cdot w^{-1}\rvert > \lvert S\rvert$ (Problem 7)	PSPACE-c (Subsec. 2.1)		PSPACE-c (Subsec. 2.1)	NP-c (Thm. 25)
$S \cdot w^{-1} = Q$ (Problem 8)	PSPACE-c (Subsec. 2.1)		NP-c (Cor. 26)	NP-c (Cor. 26)
$\lvert S \cdot w^{-1}\rvert > \lvert S\rvert$, $\lvert S\rvert \leq k$ (Problem 13)	$\mathcal{O}(\lvert\Sigma\rvert n^k)$ (Prop. 10)		$\mathcal{O}(\lvert\Sigma\rvert n^k)$ (Prop. 10)	$\mathcal{O}(\lvert\Sigma\rvert n^k)$ (Prop. 10)
$S \cdot w^{-1} = Q$, $\lvert S\rvert \leq k$ (Problem 14)	NP-c (Prop. 15)		NP-c (Prop. 15)	NP-c (Prop. 15)
$\lvert S \cdot w^{-1}\rvert > \lvert S\rvert$, $\lvert S\rvert \geq n - k$ (Problem 22, $k \geq 2$)	PSPACE-c (Thm. 19)	Open	PSPACE-c (Thm. 19)	NP-c (Cor. 26)
$S \cdot w^{-1} = Q$, $\lvert S\rvert \geq n - k$ (Problem 23, $k \geq 2$)	NP-c (Cor. 26)		NP-c (Cor. 26)	NP-c (Cor. 26)
$S \cdot w^{-1} = Q$, $\lvert S\rvert = n - 1$ (Problem 24)	NP-c (Thm. 25)		NP-c (Thm. 25)	NP-c (Thm. 25)
$\lvert S \cdot w^{-1}\rvert \neq \lvert S\rvert$ (Problem 28)	$\mathcal{O}(\lvert\Sigma\rvert n^3)$ (Thm. 29)		$\mathcal{O}(\lvert\Sigma\rvert n^3)$ (Thm. 29)	$\mathcal{O}(\lvert\Sigma\rvert n^3)$ (Thm. 29)

totally extending words. We give a special attention to the problem of avoiding one state and a small subset of states (totally extending a large subset), since they seem to be most important in view of their applications (and as we show, the complexity grows with the size of the subset to avoid).

Resizing: Shrinking a subset is dual to extending, i.e. shrinking a subset means extending its complement. Therefore, the complexity immediately transfers from the previous results. However, in Section 5 we consider the problem of determining whether there is a word whose inverse action results in a subset having a different size, that is, either extends the subset or shrinks it (Problems 27, 28).

Interestingly, in contrast with the computationally difficult problems of finding a word that extends the subset and finding a word that shrinks the subset, for this variant there exists a polynomial algorithm finding a shortest resizing word in all cases.

We can mention that in some cases extending and shrinking words are related, and it may be enough to find either one. For instance, this is used in the so-called *averaging trick*, which appears in several proofs (e.g. [7, 17, 25]).

Summary: For all the problems we consider the subclasses of strongly connected, synchronizing, and binary automata. Also, we consider the problems where an upper bound on the length of the word is additionally given in binary form in the input. Since in most cases, the problems are computationally hard, in Section 3 and Section 4 we consider parameterized complexity by the size of the given subset.

Table 1 and Table 2 summarize our results together with known results about compressing words. For the cases where a polynomial algorithm exists, we put the time complexity of the best one known. All the hardness results hold also in the case of a binary alphabet.

2 Extending a Subset in General

We deal with the following problems:

▶ **Problem 1** (Extensible subset). *Given $\mathscr{A} = (Q, \Sigma, \delta)$ and a subset $S \subseteq Q$, is S extensible?*

▶ **Problem 2** (Totally extensible subset). *Given $\mathscr{A} = (Q, \Sigma, \delta)$ and a subset $S \subseteq Q$, is there a word $w \in \Sigma^*$ such that $S \cdot w^{-1} = Q$?*

▶ **Theorem 3.** *Problem 1 and Problem 2 are PSPACE-complete even if $\mathscr{A}$ is strongly connected.*

Proof idea. To solve both problems in NPSPACE, we just guess the length of a (totally) extending word and then subsequently its letters, storing only the current subset all the time.

For PSPACE-hardness, we perform a reduction from the problem of determining whether an intersection of regular languages given as DFAs is non-empty [19]. We create one instance for both problems that consists of a strongly connected automaton and a subset S extensible if and only if it is also totally extensible, which is simultaneously equivalent to the non-emptiness of the intersection of the given regular languages. ◀

We ensure that both problems remain PSPACE-complete in the case of a binary alphabet, which follows from the following theorem.

▶ **Theorem 4.** *Given an automaton $\mathscr{A} = (Q, \Sigma, \delta)$ and a subset $S \subseteq Q$, we can construct in polynomial time a binary automaton $\mathscr{A}' = (Q', \{a', b'\}, \delta')$ and a subset $S' \subseteq Q'$ such that:*
(1) *$\mathscr{A}$ is strongly connected if and only $\mathscr{A}'$ is strongly connected;*
(2) *S' is extensible in $\mathscr{A}'$ if and only if S is extensible in $\mathscr{A}$;*
(3) *S' is totally extensible in $\mathscr{A}'$ if and only if S is totally extensible in $\mathscr{A}$.*

Proof idea. Let $\Sigma = \{a_1, \ldots, a_k\}$. We reduce $\mathscr{A}$ to a binary automaton $\mathscr{A}'$ that consists of k copies of $\mathscr{A}$. The first letter a acts in an i-th copy as the letter a_i in $\mathscr{A}$. The second letter b acts cyclically on these copies. Then we define S' to contain the states from S in the first copy and all states from the other copies. ◀

Now we consider the subclass of synchronizing automata.

▶ **Proposition 5.** *When the automaton is binary and synchronizing, Problem 1 remains PSPACE-complete.*

Proof idea. We just add a sink state z and a letter which synchronizes $\mathscr{A} = (Q, \Sigma, \delta)$ to z. Additionally, a standard tree-like binarization is suitably used to obtain a binary automaton that preserves extensibility of the subset. ◀

▶ **Theorem 6.** *When the automaton is synchronizing, Problem 2 can be solved in $\mathcal{O}(|\Sigma|n)$ time and is NL-complete.*

Proof idea. Since $\mathscr{A}$ is synchronizing, the problem reduces to checking whether there is a state $q \in S$ reachable from every state: It is well known that a synchronizing automaton has precisely one strongly connected *sink* component that is reachable from every state. If S does not contain a state from the sink component, then every preimage of S also does not contain these states. The problem can be solved in $\mathcal{O}(|\Sigma|n)$ time, since the states of the sink component can be determined in linear time by Tarjan's algorithm [28]. ◀

Note that in the case of strongly connected synchronizing automaton, both problems have a trivial solution, since every non-empty proper subset of Q is totally extensible (by a suitable reset word); thus they can be solved in constant time, assuming that we can check the size of the given subset and the number of states in constant time.

2.1 Bounded Length of the Word

We turn our attention to the variants in which an upper bound on the length of word w is also given.

▶ **Problem 7** (Extensible subset by short word). *Given $\mathscr{A} = (Q, \Sigma, \delta)$, a subset $S \subseteq Q$, and an integer ℓ given in binary form, is S extensible by a word of length at most ℓ?*

▶ **Problem 8** (Totally extensible subset by short word). *Given $\mathscr{A} = (Q, \Sigma, \delta)$, a subset $S \subseteq Q$, and an integer ℓ given in binary form, is there a word $w \in \Sigma^*$ such that $S \cdot w^{-1} = Q$ of length at most ℓ?*

Obviously, these problems remain PSPACE-complete (also when the automaton is strongly connected and binary), as we can set $\ell = 2^n$, which bounds the number of different subsets of Q. In this case, both the problems are reduced respectively to Problem 1 and Problem 2.

When the automaton is synchronizing, Problem 8 is NP-complete, which will be shown in Corollary 26. Of course, Problem 7 remains PSPACE-complete for a synchronizing automaton by the same argument as in the general case.

3 Extending Small Subsets

The complexity of extending problems rely on the size of the given subset. Variable subset size is essential for hardness. In the proof of PSPACE-hardness in Theorem 3 the used subsets and simultaneously their complements may grow with an instance of the reduced problem, and it is known that the problem of the emptiness of intersection can be solved in polynomial time if the number of given DFAs is fixed. Here we study the computational complexity of the extending problems when the size of the subset is not larger than a fixed k.

▶ **Problem 9** (Extensible small subset). *For a fixed $k \in \mathbb{N}$, given $\mathscr{A} = (Q, \Sigma, \delta)$ and a subset $S \subseteq Q$ with $|S| \leq k$, is there a word extending S?*

▶ **Proposition 10.** *Problem 9 can be solved in $\mathcal{O}(|\Sigma|n^k)$ time.*

Proof. We build the k-subsets automaton $\mathscr{A}^{\leq k} = (Q^{\leq k}, \Sigma, \delta^{\leq k}, S_0, F)$, where $Q^{\leq k} = \{A \subseteq Q \colon |A| \leq k\}$ and $\delta^{\leq k}$ is naturally defined by the image of δ on a subset. Let the set of initial states be $I = \{A \in Q^{\leq k} \colon |A \cdot a^{-1}| > |S| \text{ for some } a \in \Sigma\}$, and the set of final states be the set of all subsets of S. A final state can be reached from an initial state if and only if S is extensible in $\mathscr{A}$. We can simply check this condition by a BFS. The size (number of states and edges) of this automaton is bounded by $\mathcal{O}(|\Sigma|n^k)$, so the procedure takes this time. ◀

▶ **Problem 11** (Totally extensible small subset). *For a fixed $k \in \mathbb{N}$, given $\mathscr{A} = (Q, \Sigma, \delta)$ and a subset $S \subseteq Q$ with $|S| \leq k$, is there a word $w \in \Sigma^*$ such that $S \cdot w^{-1} = Q$?*

For $k = 1$ Problem 2 is equivalent to checking if the automaton is synchronizing to the given state, thus can be solved in $\mathcal{O}(|\Sigma|n^2)$ time. For larger k we have the following:

▶ **Proposition 12.** *Problem 11 can be solved in $\mathcal{O}(|\Sigma|(n^3 + n^k))$ time.*

Proof. Let u be a word of the minimal rank in $\mathscr{A}$. We can find such a word and compute the image $Q \cdot u$ in $\mathcal{O}(|\Sigma|n^3)$ time, using e.g. the algorithm from [12].

For each $w \in \Sigma^*$ we have $S \cdot w^{-1} = Q$ if and only if $Q \cdot w \subseteq S$. We can meet the required condition for w if and only if $(Q \cdot u) \cdot w \subseteq S$. Surely $|(Q \cdot u) \cdot w| = |(Q \cdot u)|$. The desired word does not exist if the minimal rank is larger than $|S| = k$. Otherwise, we can build the subset automaton $\mathscr{A}^{\leq |Q \cdot u|}$ (similarly as in the proof of Proposition 10). The initial subset is $Q \cdot u$. If some subset of S is reachable by a word w, then the word uw totally extends S in $\mathscr{A}$. Otherwise, S is not totally extensible. Reachability can be checked in at most $\mathcal{O}(|\Sigma|n^k)$ time. However, if the rank r of u is less than k, the algorithm takes only $\mathcal{O}(|\Sigma|n^r)$ time. ◀

3.1 Bounded Length of the Word

We also have the two variants of the above problems when an upper bound on the length of the word is additionally given.

▶ **Problem 13** (Extensible small subset by short word). *For a fixed $k \in \mathbb{N}$, given $\mathscr{A} = (Q, \Sigma, \delta)$, a subset $S \subseteq Q$ with $|S| \leq k$, and an integer ℓ given in binary form, is there a word extending S of length at most ℓ?*

Problem 13 can be solved by the same algorithm in a Proposition 10, since the procedure can find a shortest extending word.

▶ **Problem 14** (Totally extensible small subset by short word). *For a fixed $k \in \mathbb{N}$, given $\mathscr{A} = (Q, \Sigma, \delta)$, a subset $S \subseteq Q$ with $|S| \leq k$, and an integer ℓ given in binary form, is there a word $w \in \Sigma^*$ such that $S \cdot w^{-1} = Q$ of length at most ℓ?*

▶ **Proposition 15.** *For every k, Problem 14 is NP-complete, even if the automaton is simultaneously strongly connected, synchronizing, and binary.*

Proof. The problem is in NP, as the shortest extending words have length at most $\mathcal{O}(n^3 + n^k)$ (since words of this length can be found by the procedure from Proposition 12).

When we choose S of size 1, the problem is equivalent to finding a reset word that maps every state to the state in S. In [34] it has been shown that for Eulerian automata that are simultaneously strongly connected, synchronizing, and binary, deciding whether there is a reset word of length at most ℓ is NP-complete. Moreover, in this construction, if there exists a reset word of this length, then it maps every state to one particular state s_2 (see [34, Lemma 2.4]). Therefore, we can set $S = \{s_2\}$, and thus Problem 14 is NP-complete. ◀

4 Extending Large Subsets

We consider here the case when the subset S contains all except at most a fixed number of states k.

▶ **Problem 16** (Extensible large subset). *For a fixed $k \in \mathbb{N}$, given $\mathscr{A} = (Q, \Sigma, \delta)$ and a subset $S \subseteq Q$ with $|Q \setminus S| \leq k$, is there a word extending S?*

▶ **Problem 17** (Totally extensible large subset). *For a fixed $k \in \mathbb{N}$, given $\mathscr{A} = (Q, \Sigma, \delta)$ and a subset $S \subseteq Q$ with $|Q \setminus S| \leq k$, is there a word $w \in \Sigma^*$ such that $S \cdot w^{-1} = Q$?*

Problem 17 is equivalent to deciding the existence of an avoiding word for a subset S of size $\leq k$. Note that both problems are equivalent for $k = 1$, which is the problem of avoiding a single given state. Their properties will also turn out to be different than in the case of $k \geq 2$. We give a special attention to this problem and study it separately.

▶ **Problem 18** (Avoidable state). *Given $\mathscr{A} = (Q, \Sigma, \delta)$ and a state $q \in Q$, is there a word $w \in \Sigma^*$ such that $q \notin Q \cdot w$?*

The following result may be a bit surprising, in view of that it is the only case where the general problem remains equally hard when the subset size is bounded. We state that the first problem remains PSPACE-complete for all $k \geq 2$, although the problem remains open for strongly connected automata.

▶ **Theorem 19.** *Problem 16 is PSPACE-hard for every $k \geq 2$ and $|\Sigma| \geq 2$ even if the given automaton is synchronizing.*

Proof idea. We show a reduction from PSPACE-complete Problem 2 (Thm. 3). Our obtained automaton is ternary and synchronizing with a sink state e'.

We reduce the alphabet to two letters by an application of the Theorem 4 to Problem 16. The reduction in the proof keeps the size of complement set the same, so we can apply it. Furthermore, we identify all copies of the sink state e', which ensures that it remains synchronizing. ◀

Now, we focus on totally extending words for large subsets, which we study in terms of avoiding small subsets. First we provide a complete characterization of single states that are avoidable:

▶ **Theorem 20.** *Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a strongly connected automaton. For every $q \in Q$, state q is avoidable if and only if there exists $p \in Q \setminus \{q\}$ and $w \in \Sigma^*$ such that $q \cdot w = p \cdot w$.*

Proof. Let p and w be the state and the word from the theorem for a given state q. Since the automaton is strongly connected, there is a word w' such that such that $(p \cdot w) \cdot w' = (q \cdot w) \cdot w' = p$. For each subset $S \subseteq Q$ such that $p \in S$ we have $p \in S \cdot ww'$. Moreover, if $q \in S$ then $|S \cdot ww'| < |S|$, because $\{q, p\} \cdot ww' = \{p\}$. If q is not avoidable, then all subsets $Q \cdot (ww'), Q \cdot (ww')^2, \ldots$ contain q and they form an infinite sequence of subsets of decreasing cardinality, which is a contradiction.

Now consider the other direction. Suppose for a contradiction that q is avoidable, but there is no state $p \in Q \setminus \{q\}$ such that $\{q, p\}$ can be compressed. Let u be a word of the minimal rank in $\mathscr{A}$, and v be a word that avoids q. Then $w = uv$ has the same rank and also avoids q. Let $\sim$ be the equivalence relation defined by

$$p_1 \sim p_2 \iff p_1 \cdot w = p_2 \cdot w.$$

The equivalence class $[p]_\sim$ for $p \in Q$ is $(p \cdot w) \cdot w^{-1}$. There are $|Q/\!\sim| = |Q \cdot w|$ equivalence classes and one of them is $\{q\}$, since q does not belong to a compressible pair of states. For every state $p \in Q$, we know that $|(Q \cdot w) \cap [p]_\sim| \leq 1$, because $[p]_\sim$ is compressed by w to a singleton and $Q \cdot w$ cannot be compressed by any word. Note that every state $r \in Q \cdot w$ belongs to some class $[p]_\sim$. From the equality $|Q/\!\sim| = |Q \cdot w|$ we conclude that for every class $[p]_\sim$ there is a state $r \in (Q \cdot w) \cap [p]_\sim$, thus $|(Q \cdot w) \cap [p]_\sim| = 1$. In particular, $1 = |(Q \cdot w) \cap [q]_\sim| = |(Q \cdot w) \cap \{q\}|$. This contradicts that w avoids q. ◀

Note that if $\mathscr{A}$ is not strongly connected, then every state from a strongly connected component that is not a sink can be avoided. If a state belongs to a sink component, then we can consider the sub-automaton of this sink component, and by Theorem 20 we know that given $q \in Q$, it is sufficient to check whether q belongs to a compressible pair of states. Hence, Problem 18 can be solved using the well-known algorithm [12] computing the pair automaton and performing a breadth-first search with inverse edges on the pairs of states. It works in $\mathcal{O}(|\Sigma|n^2)$ time and $\mathcal{O}(n^2 + |\Sigma|n)$ space.

We note that in a synchronizing automaton all states are avoidable except a *sink state*, which is a state q such that $q \cdot a = q$ for all $a \in \Sigma$. We can check this condition and hence verify if a state is avoidable in a synchronizing automaton in $\mathcal{O}(|\Sigma|)$ time.

The above algorithm does not find an avoiding word but checks avoidability indirectly. For larger subsets than singletons, we construct another algorithm finding a word avoiding the subset, which also generalizes the idea from Theorem 20. From the following theorem, it follows that Problem 17 for $k \geq 2$ can be solved in polynomial time.

▶ **Theorem 21.** *Let $\mathscr{A} = (Q, \Sigma, \delta)$, let r be the minimum rank in $\mathscr{A}$ over all words, and let $S \subseteq Q$ be a subset of size $\leq k$. We can find a word w such that $(Q \cdot w) \cap S = \emptyset$ or verify that it does not exist in $\mathcal{O}(n^3 + |\Sigma|(n^2 + n^{\min(r,k)}))$ time and $\mathcal{O}(n^2 + n^{\min(r,k)} + |\Sigma|n)$ space. Moreover the length of w is bounded by $\mathcal{O}(n^3 + n^{\min(r,k)}))$.*

Proof. Similarly to the proof of Theorem 20, let u be a word of the minimal rank r in $\mathscr{A}$ and let $\sim$ be the equivalence relation on Q defined by

$$p_1 \sim p_2 \iff p_1 \cdot u = p_2 \cdot u.$$

The equivalence class $[p]_\sim$ for $p \in Q$ is the set $(p \cdot u) \cdot u^{-1}$. There are $|Q/\!\sim| = |Q \cdot u|$ equivalence classes.

Now, we are going to show the following characterization: S is avoidable if and only if there exist a subset $Q' \subseteq Q \cdot u$ of size $|S/\!\sim|$ and a word w' such that $(Q' \cdot w') \cap ([s]_\sim \setminus S) \neq \emptyset$ for each $s \in S$.

Suppose that S is avoidable, and let w' be an avoiding word for S. Then the word $w = uw'$ also avoids S. Observe that w has rank r as u has. For every state $p \in Q$, we know that $|(Q \cdot w) \cap [p]_\sim| \leq 1$, because $[p]_\sim$ is compressed by u to a singleton and $Q \cdot w$ cannot be compressed by any word. Note that every state $q \in Q \cdot w$ belongs to some class $[p]_\sim$. From the equality $|Q/\sim| = |Q \cdot u| = |Q \cdot w|$ we conclude that for every class $[p]_\sim$ there is a unique state $q_{[p]_\sim} \in (Q \cdot w) \cap [p]_\sim$.

Then for every state $s \in S$, we have $q_{[s]_\sim} \in [s]_\sim \setminus S$, because w avoids S and $q_{[s]_\sim} \in Q \cdot w$. Notice that $[s]_\sim \cap S$ can contain more than one state, so the set $\{q_{[s]_\sim} \mid s \in S\}$ has size $|S/\!\sim|$, which is not always equal to $|S|$. Therefore, there exists a subset $Q' \subseteq Q \cdot u$ of size $|S/\!\sim|$ such that $Q' \cdot w' = \{q_{[s]_\sim} \mid s \in S\}$. Now, we know that for every $s \in S$ we have $q_{[s]_\sim} \in Q' \cdot w'$ and $q_{[s]_\sim} \in [s]_\sim \setminus S$. We conclude that, if S is avoidable, then there exist a subset $Q' \subseteq Q \cdot u$ of size $|S/\!\sim|$ and a word w' such that $(Q' \cdot w') \cap ([s]_\sim \setminus S) \neq \emptyset$ for every $s \in S$.

Conversely, suppose that there is a subset $Q' \subseteq Q \cdot u$ of size $|S/\!\sim|$ and a word w' such that $(Q' \cdot w') \cap ([s]_\sim \setminus S) \neq \emptyset$ for every $s \in S$. Since in the image $Q \cdot uw'$ there is exactly one state in each equivalence class, we have $((Q \cdot u) \setminus Q') \cdot w' \subseteq Q \setminus \bigcup_{s \in S}([s]_\sim) \subseteq Q \setminus S$, and by the assumption, $(Q' \cdot w') \cap S = \emptyset$. Therefore, we get that uw' is an avoiding word for S.

This characterization gives us Alg. 1 to find w or verify that S cannot be avoided.

Alg. 1 first finds a word u of the minimal rank. This can be done by iterative compressing the subset as long as possible by the algorithm from [12], which works in $\mathcal{O}(n^3 + |\Sigma|n^2)$ time and $\mathcal{O}(n^2 + |\Sigma|n)$ space. For every subset $Q' \subseteq Q \cdot u$ of size $z = |S/\!\sim|$ the algorithm checks

Algorithm 1 Avoiding a subset.

Require: Automaton $\mathscr{A}(Q, \Sigma, \delta)$ and a subset $S \subseteq Q$.
1: Find a word u of the minimal rank.
2: Compute $|S/\!\sim|$.
3: **for all** $Q' \subseteq Q \cdot u$ of size $|S/\!\sim|$ **do**
4: **if** there is a word w' such that $(Q' \cdot w') \cap ([s]_\sim \setminus S) \neq \emptyset$ for each $s \in S$ **then**
5: **return** uw'.
6: **end if**
7: **end for**
8: **return** "S is unavoidable".

whether there is a word w' mapping Q' to avoid S, but using its $\sim$-classes. This can be done by constructing the automaton $\mathscr{A}^z(Q^z, \Sigma, \delta^z)$, where δ^z is δ naturally extended to z-tuples of states, and checking whether there is a path from Q' to a subset containing a state from each class $[s]_\sim$ but avoiding the states from S. Note that since Q' cannot be compressed, every reachable subset from Q' has also size $|Q'|$. The number of states in this automaton is $\binom{n}{z} \in \mathcal{O}(n^z)$. Also, note that we have to visit every z-tuple only once during a run of the algorithm, and we can store it in $\mathcal{O}(n^z + |\Sigma|n)$ space. Therefore, the algorithm works in $\mathcal{O}(n^3 + |\Sigma|(n^2 + n^z))$ time and $\mathcal{O}(n^2 + n^z + |\Sigma|n)$ space.

The length of u is bounded by $\mathcal{O}(n^3)$, and the length of w' is at most $\mathcal{O}(n^z)$. Note that $z = |S/\!\sim| \leq \min(r, |S|)$, where r is the minimal rank in the automaton. ◀

4.1 Bounded Length of the Word

We now turn our attention to the variants of the problems where an upper bound on the length of the word is given.

▶ **Problem 22** (Extensible large subset by short word). *For a fixed $k \in \mathbb{N}$, given $\mathscr{A} = (Q, \Sigma, \delta)$, a subset $S \subseteq Q$ with $|Q \setminus S| \leq k$, and an integer ℓ given in binary form, is there a word extending S of length at most ℓ?*

▶ **Problem 23** (Totally extensible large subset by short word). *For a fixed $k \in \mathbb{N}$, given $\mathscr{A} = (Q, \Sigma, \delta)$, a subset $S \subseteq Q$ with $|Q \setminus S| \leq k$, and an integer ℓ given in binary form, is there a word $w \in \Sigma^*$ such that $S \cdot w^{-1} = Q$ of length at most ℓ?*

As before, both problems for $k = 1$ are equivalent to the following:

▶ **Problem 24** (Avoidable state by short word). *Given $\mathscr{A} = (Q, \Sigma, \delta)$, a state $q \in Q$, and an integer ℓ given in binary form, is there a word $w \in \Sigma^*$ such that $q \notin Q \cdot w$ of length at most ℓ?*

Problem 22 for $k \geq 2$ obviously remains PSPACE-complete. By the following theorem, we show that Problem 24 is NP-complete, which then implies NP-completeness of Problem 23 for every $k \geq 1$ (by Corollary 26).

▶ **Theorem 25.** *Problem 24 is NP-complete, even if the automaton is simultaneously strongly connected, synchronizing, and binary.*

Proof idea. The problem is in NP, because we can non-deterministically guess a word w as a certificate, and verify $q \notin Q \cdot w$ in $\mathcal{O}(|\Sigma|n)$ time. If the state q is avoidable, then the length of the shortest avoiding words is at most $\mathcal{O}(n^2)$ [27]. Then we can guess an avoiding word w of at most quadratic length and compute $Q \cdot w$ in $\mathcal{O}(n^3)$ time.

To prove NP-hardness, we show a reduction from the problem of determining the reset threshold in the specific subclass of automata constructed in the Eppstein's proof of [12, Theorem 8], which is known to be NP-complete. The reduction has two steps. First, we construct a strongly connected synchronizing ternary automaton for which deciding about the length of an avoiding word is equivalent to determining the existence of a reset word in the original automaton. Then, based on the ideas from [5] we turn the automaton into a binary one that still has the desired properties. ◀

As a corollary from Theorem 25 and Theorem 21, we complete the results.

▶ **Corollary 26.** *Problem 23 is NP-complete, Problem 8 is NP-complete when the automaton is synchronizing, and Problem 22 is NP-complete when the automaton is strongly connected and synchronizing. They remain NP-complete when the automaton is simultaneously strongly connected, synchronizing, and binary.*

5 Resizing a Subset

▶ **Problem 27** (Resizable subset)**.** *Given an automaton $\mathscr{A} = (Q, \Sigma, \delta)$ and a subset $S \subseteq Q$, is there a word $w \in \Sigma^*$ such that $|S \cdot w^{-1}| \neq |S|$?*

▶ **Problem 28** (Resizable subset by short word)**.** *Given an automaton $\mathscr{A} = (Q, \Sigma, \delta)$, a subset $S \subseteq Q$, and an integer ℓ given in binary form, is there a word $w \in \Sigma^*$ such that $|S \cdot w^{-1}| \neq |S|$ of length at most ℓ?*

In contrast to the cases $|S \cdot w^{-1}| > |S|$ and $|S \cdot w^{-1}| < |S|$, there exists a polynomial time algorithm for both these problems.

▶ **Theorem 29.** *Given an automaton $\mathscr{A} = (Q, \Sigma, \delta)$, a subset $S \subseteq Q$, there exists an algorithm working in $\mathcal{O}(|\Sigma|n^3)$ time (assuming constant time arithmetic of integers whose values are bounded by $\mathcal{O}(2^n)$) that computes a shortest word w such that $|S \cdot w^{-1}| \neq |S|$ or verifies that there is no such word. Moreover, the length of the shortest such words is a at most $n - 1$.*

Proof idea. We construct a reduction to the problem of multiplicity equivalence of NFAs and apply the algorithm from [30] with an improvement to achieve the desired complexity[4]. ◀

The running time $\mathcal{O}(|\Sigma|n^3)$ of the algorithm is quite large (and may require large arithmetic), and it is an interesting open question whether there is a faster algorithm for Problems 27 and 28.

We note that Problem 27 becomes trivial when the automaton is synchronizing: A word resizing the subset exists if and only if $S \neq \emptyset$ and $S \neq Q$, because if w is a reset word and $\{q\} = Q \cdot w$, then $S \cdot w^{-1}$ is either Q when $q \in S$ or $\emptyset$ when $q \notin S$. This implies that there exists a faster algorithm in the sense of expected running time when the automaton over an at least binary alphabet is drawn uniformly at random:

▶ Remark. The algorithm from [6] checks in expected $\mathcal{O}(n)$ time (regardless of the alphabet size, which is not fixed) whether a random automaton is synchronizing, and it is synchronizing with probability $1 - \Theta(1/n^{0.5|\Sigma|})$ (for $|\Sigma| \geq 2$). Then only if it is not synchronizing we

[4] In a previous version of our proof we presented our own algorithm having $\mathcal{O}(|\Sigma|n^3)$ time complexity under the assumption of performing arithmetic computations in constant time. We thank one of the anonymous reviewers that suggested a shortcut by reducing to multiplicity equivalence of NFAs.

have to use the algorithm from Theorem 29. Thus, Problem 28 can be solved for a random automaton in the expected time

$$\mathcal{O}(|\Sigma|n^3) \cdot \Theta(1/n^{0.5|\Sigma|}) + \mathcal{O}(n) = \mathcal{O}(|\Sigma|n^{3-0.5|\Sigma|}) \leq \mathcal{O}(n^2).$$

Note that the bound is independent on the alphabet size, and this is because a random automaton with a growing alphabet is more likely to be synchronizing, so less likely we need to use Theorem 29.

References

1. D. S. Ananichev and V. V. Gusev. Approximation of Reset Thresholds with Greedy Algorithms. *Fundamenta Informaticae*, 145(3):221–227, 2016.
2. D. S. Ananichev and M. V. Volkov. Synchronizing generalized monotonic automata. *Theoretical Computer Science*, 330(1):3–13, 2005.
3. M.-P. Béal, M. Berlinkov, and D. Perrin. A quadratic upper bound on the size of a synchronizing word in one-cluster automata. *International Journal of Foundations of Computer Science*, 22(2):277–288, 2011.
4. M. Berlinkov. Synchronizing Quasi-Eulerian and Quasi-one-cluster Automata. *International Journal of Foundations of Computer Science*, 24(6):729–745, 2013.
5. M. Berlinkov. On Two Algorithmic Problems about Synchronizing Automata. In *Developments in Language Theory*, LNCS, pages 61–67. Springer, 2014.
6. M. Berlinkov. On the probability of being synchronizable. In *CALDAM*, volume 9602 of *LNCS*, pages 73–84. Springer, 2016.
7. M. Berlinkov and M. Szykuła. Algebraic synchronization criterion and computing reset words. *Information Sciences*, 369:718–730, 2016.
8. J. Berstel, D. Perrin, and C. Reutenauer. *Codes and Automata*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2009.
9. M. T. Biskup and W. Plandowski. Shortest synchronizing strings for Huffman codes. *Theoretical Computer Science*, 410(38-40):3925–3941, 2009.
10. E. A. Bondar and M. V. Volkov. Completely reachable automata. In C. Câmpeanu, F. Manea, and J. Shallit, editors, *DCFS*, LNCS, pages 1–17. Springer, 2016.
11. J. Černý. Poznámka k homogénnym eksperimentom s konečnými automatami. *Matematicko-fyzikálny Časopis Slovenskej Akadémie Vied*, 14(3):208–216, 1964. In Slovak.
12. D. Eppstein. Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19:500–510, 1990.
13. P. Gawrychowski and D. Straszak. Strong inapproximability of the shortest reset word. In *Mathematical Foundations of Computer Science*, volume 9234 of *LNCS*, pages 243–255. Springer, 2015.
14. M. Grech and A. Kisielewicz. The Černý conjecture for automata respecting intervals of a directed graph. *Discrete Mathematics and Theoretical Computer Science*, 15(3):61–72, 2013.
15. K. Guldstrand Larsen, S. Laursen, and J. Srba. Synchronizing Strategies under Partial Observability. In P. Baldan and D. Gorla, editors, *CONCUR 2014*, pages 188–202. Springer, 2014.
16. H. Jürgensen. Synchronization. *Information and Computation*, 206(9-10):1033–1044, 2008.
17. J. Kari. Synchronizing finite automata on Eulerian digraphs. *Theoretical Computer Science*, 295(1-3):223–232, 2003.
18. A. Kisielewicz, J. Kowalski, and M. Szykuła. Computing the shortest reset words of synchronizing automata. *Journal of Combinatorial Optimization*, 29(1):88–124, 2015.

19 D. Kozen. Lower Bounds for Natural Proof Systems. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, FOCS, pages 254–266, 1977.

20 P. Martyugin. Computational Complexity of Certain Problems Related to Carefully Synchronizing Words for Partial Automata and Directing Words for Nondeterministic Automata. *Theory of Computing Systems*, 54(2):293–304, 2014.

21 J. Olschewski and M. Ummels. The complexity of finding reset words in finite automata. In *Mathematical Foundations of Computer Science*, volume 6281 of *LNCS*, pages 568–579. Springer, 2010.

22 A. Roman and M. Szykuła. Forward and backward synchronizing algorithms. *Expert Systems with Applications*, 42(24):9512–9527, 2015.

23 I. K. Rystsov. Polynomial complete problems in automata theory. *Information Processing Letters*, 16(3):147–151, 1983.

24 S. Sandberg. Homing and synchronizing sequences. In *Model-Based Testing of Reactive Systems*, volume 3472 of *LNCS*, pages 5–33. Springer, 2005.

25 B. Steinberg. The averaging trick and the Černý conjecture. *International Journal of Foundations of Computer Science*, 22(7):1697–1706, 2011.

26 B. Steinberg. The Černý conjecture for one-cluster automata with prime length cycle. *Theoretical Computer Science*, 412(39):5487–5491, 2011.

27 M. Szykuła. Improving the Upper Bound on the Length of the Shortest Reset Word. In *STACS 2018*, LIPIcs, pages 56:1–56:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

28 R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

29 A. N. Trahtman. The Černý conjecture for aperiodic automata. *Discrete Mathematics and Theoretical Computer Science*, 9(2):3–10, 2007.

30 W.-G. Tzeng. The Equivalence and Learning of Probabilistic Automata. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, FOCS, pages 268–273. IEEE Computer Society, 1989.

31 M. V. Volkov. Synchronizing automata and the Černý conjecture. In *Language and Automata Theory and Applications*, volume 5196 of *LNCS*, pages 11–27. Springer, 2008.

32 M. V. Volkov. Synchronizing automata preserving a chain of partial orders. *Theoretical Computer Science*, 410(37):3513–3519, 2009.

33 V. Vorel. Subset Synchronization of Transitive Automata. In *Proceedings 14th International Conference on Automata and Formal Languages (AFL 2014)*, pages 370–381, 2014.

34 V. Vorel. Complexity of a problem concerning reset words for Eulerian binary automata. *Information and Computation*, 253(Part 3):497–509, 2017.

The Complexity of Disjunctive Linear Diophantine Constraints

Manuel Bodirsky[1]
Institut für Algebra, TU Dresden, Germany
manuel.bodirsky@tu-dresden.de

Barnaby Martin
Department of Computer Science, Durham University, U.K.
barnabymartin@gmail.com

Marcello Mamino[2]
Dipartimento di Matematica, largo Pontecorvo 5, 56127 Pisa, Italy
marcello.mamino@dm.unipi.it

Antoine Mottet[3]
Institut für Algebra, TU Dresden, Germany
antoine.mottet@tu-dresden.de

Abstract

We study the Constraint Satisfaction Problem CSP($\mathbb{A}$), where $\mathbb{A}$ is first-order definable in $(\mathbb{Z}; +, 1)$ and contains $+$. We prove such problems are either in P or NP-complete.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic

Keywords and phrases Constraint Satisfaction, Presburger Arithmetic, Computational Complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.33

1 Introduction

A *constraint satisfaction problem* (CSP) is a computational problem where the input consists of a finite set of variables and a finite set of constraints, and where the question is whether there exists a mapping from the variables to some fixed domain such that all the constraints are satisfied. When the domain is finite, and arbitrary constraints are permitted in the input, the CSP is NP-complete. However, when only constraints for a restricted set of relations are allowed in the input, it might be possible to solve the CSP in polynomial time. The set of relations that is allowed to formulate the constraints in the input is often called the *constraint language*. The question as to which constraint languages give rise to polynomial-time solvable CSPs has been the topic of intensive research over the past years. It was conjectured by Feder and Vardi [14] that CSPs for constraint languages over finite domains have a complexity dichotomy: they are in P or are NP-complete. This conjecture has recently been proved [12, 25].

[1] Manuel Bodirsky has received funding from the ERC under the European Community's Seventh Framework Programme (Grant Agreement no. 681988, CSP-Infinity), and the DFG-funded project 'Homogene Strukturen, Bedingungserfüllungsprobleme, und topologische Klone' (Project number 622397)
[2] Marcello Mamino has received funding from the ERC under the European Community's Seventh Framework Programme (Grant Agreement no. 681988, CSP-Infinity).
[3] Supported by the DFG Gratuiertenkolleg 1763 (QuantLA).

A famous CSP over an infinite domain is the feasibility question for Integer Programs. It is of great importance in practice and theory of computing, and NP-complete. In order to obtain a systematic understanding of polynomial-time solvable restrictions and variations of this problem, Jonsson and Lööw [15] proposed to study the class of CSPs where the constraint language $\mathbb{A}$ is definable in *Presburger arithmetic*; that is, consists of relations that have a first-order definition over $(\mathbb{Z}; <, +, 1)$. Equivalently, each relation $R(x_1, \ldots, x_n)$ in $\mathbb{A}$ can be defined by a disjunction of conjunctions of the atomic formulas of the form $p \leq 0$ where p is a linear polynomial with integer coefficients and variables from $\{x_1, \ldots, x_n\}$. The constraint satisfaction problem for $\mathbb{A}$, denoted by $\mathrm{CSP}(\mathbb{A})$, is the problem of deciding whether a given conjunction of formulas of the form $R(y_1, \ldots, y_n)$, for some n-ary R from $\mathbb{A}$, is satisfiable in $\mathbb{A}$. By appropriately choosing such a constraint language $\mathbb{A}$, a great variety of problems over the integers can be formulated as $\mathrm{CSP}(\mathbb{A})$. Several constraint languages $\mathbb{A}$ over the integers are known where the CSP can be solved in polynomial time. Among the most famous of these is Linear Diophantine Equations, namely $\mathrm{CSP}(\mathbb{Z}; +, 1)$. The first polynomial-time algorithms for the satisfiability of linear Diophantine equation systems have been discovered by Frumkin and, independently, Sieveking and von zur Gathen. Kannan and Bachem [17] presented a method based on first computing the Hermite Normal Form of the matrix given by the linear system (see discussion in the text-book of Schrijver [23]). Further improvements have been made in [13, 24, 19]. In the present parlance, $\mathrm{CSP}(\mathbb{Z}; <, +, 1)$ is Integer Program feasibility itself. However, a complete complexity classification for the CSPs of Jonsson-Lööw languages appears to be a very ambitious goal.

Among the classes of constraint language that fall into the framework of Jonsson and Lööw are the *distance CSPs* of [5, 10] and the *temporal CSPs* of [8]. Temporal CSPs are those whose constraint language is first-order definable in $(\mathbb{Q}; <)$ and *discrete temporal CSPs* are those whose constraint languages is first-order definable in $(\mathbb{Z}; <)$. The classification for discrete temporal CSPs represents the join of the work on temporal CSPs and distance CSPs, and has only recently been accomplished [4].

Moving away from the discrete and non-dense, $(\mathbb{Q}; <)$ is not the only structure for which constraint languages that are first-order expansions have had their CSPs classified. The situation for such expansions of the language of linear programming, $(\mathbb{Q}; <, +, 1)$ was settled in [6]. Perhaps, more interesting for us is the simplified situation in which only first-order expansions of $(\mathbb{Q}; +)$ are considered, in [7]. Most recently, the work [16] delivers a classification for all first-order definitions in $(\mathbb{Q}; <, +, 1)$ that contain $+$, thus properly extending the result from [6]. In these works, the class of relations quantifier-free definable in Horn CNF plays a key role. In this context, the atomic relations are inequalities and equalities, and each clause may have no more than one equality or inequality. That is, additional disjuncts in clauses must be disequalities. For first-order expansions of $(\mathbb{Q}; +)$, the tractable constraint languages are precisely those that are quantifier-free Horn definable on $(\mathbb{Q}; +)$ [7].

However, the integers behave very differently from the rationals or reals and even simple types of Horn definitions engender intractable constraint languages, as documented in [15]. This article shows, depending on one's perspective, [un]surprisingly, that the tractability frontier for first-order definitions of $(\mathbb{Z}; +, 1)$, containing $+$, coincides with that for first-order expansions of $(\mathbb{Q}; +)$. Under a mild technical assumption on $\mathbb{A}$, either all of its relations are quantifier-free Horn definable, in the expansion of $(\mathbb{Z}; +, 1)$ associated with its quantifier elimination, and $\mathrm{CSP}(\mathbb{A})$ is solvable in P; or $\mathrm{CSP}(\mathbb{A})$ is NP-complete. From this we obtain the following dichotomy result.

▶ **Theorem 1.** *Let $\mathbb{A}$ be an expansion of $(\mathbb{Z}; +)$ by finitely many relations with a first-order definition in $(\mathbb{Z}; +, 1)$. Then $\mathrm{CSP}(\mathbb{A})$ is in P or NP-complete.*

Other related work

This work forms part of a growing body addressing infinite-domain CSPs. One line of that work concerns ω-categorical and finitely-bounded constraint languages and the other line considers constraint languages over ordinary structures of arithmetic. The two lines overlap in the foundational work on temporal CSPs [8]. The outstanding other result in the first line is [11] and recent progress can be seen in [3, 1]. The importance of the latter line is discussed in the survey [9].

The CSP for certain finite groups were studied already in the seminal [14]. $(\mathbb{Z}; +, 0)$ is a group *par excellence* and our work takes inspiration from that paper. One of our hardness results uses its Theorem 34 and our tractable cases include the situation when all relations are subgroups, or cosets of subgroups, of powers of $\mathbb{Z}$ (cf. [14], Theorem 33). However, not all first-order expansions of $(\mathbb{Z}; +, 1)$ are related to groups, and we have other sources of tractability too.

2 Preliminaries

We say a relational structure $\mathbb{A}$ is *first-order definable in* $(\mathbb{Z}; +, 1)$ (or a *first-order reduct of*$(\mathbb{Z}; +, 1)$ if it is over domain $\mathbb{Z}$ with relations specified by first-order formulas over $(\mathbb{Z}; +, 1)$. An *endomorphism* of $\mathbb{A}$ is a map $h\colon \mathbb{Z} \to \mathbb{Z}$ such that for every relation R of $\mathbb{A}$ and every tuple $(a_1, \dots, a_k) \in \mathbb{Z}^k$, we have $\mathbf{a} \in R \Rightarrow h(\mathbf{a}) \in R$. We say that h is a *self-embedding* if the implication is an equivalence.

A formula over a relational signature σ is *primitive positive (pp)* if it is of the form $\exists x_1, \dots, x_k(\psi_1 \wedge \dots \wedge \psi_m)$ where each ψ_i is an atomic relation built from σ. Note that 0 is pp-definable in $(\mathbb{Z}; +)$. A *sentence* is a formula without free variables.

The *constraint satisfaction problem* for a structure $\mathbb{A}$ with finite relational signature σ, denoted CSP($\mathbb{A}$), is the following computational problem.

Input: A primitive positive σ-sentence Φ.
Question: $\mathbb{A} \models \Phi$?

All CSPs will be defined over strictly relational signatures, thus in this context $+$ must be considered a ternary relation and 1 a constant or singleton unary relation, depending on taste. Since we also use $+$ with its common meaning of binary operation, we concede guilt for overloading. However, the two uses will never conflict in meaning, so we will not dwell further on the matter. If $\mathbb{A}$ is first-order definable in $(\mathbb{Z}; <, +, 1)$ then CSP($\mathbb{A}$) is in NP (this is noted e.g. in [15]).

A *linear equation* is a formula of the form $\sum_{i=1}^{n} a_i x_i = b$ with $a_1, \dots, a_n, b \in \mathbb{Z}$, whose free variables are $\{x_1, \dots, x_n\}$. A *modular linear equation* is a formula of the form $\sum_{i=1}^{n} a_i x_i = b \bmod c$ with $a_1, \dots, a_n, b, c \in \mathbb{Z}$. Let $\mathcal{L}_{(\mathbb{Z};+,1)}$ be the infinite relational language containing a relation symbol for each linear equation and modular linear equation. For convenience, we consider first-order logic to have native symbols for $\top$ (true) and $\bot$ (false). It is well-known that $(\mathbb{Z}; +, 1)$ admits quantifier elimination in the language $\mathcal{L}_{(\mathbb{Z};+,1)}$ (see [21], or [18, Corollary 3.1.21] for a more modern treatment). Call an $\mathcal{L}_{(\mathbb{Z};+,1)}$-formula *standard* if it does not contain a negated modular linear equation. Every $\mathcal{L}_{(\mathbb{Z};+,1)}$-formula is equivalent to a standard $\mathcal{L}_{(\mathbb{Z};+,1)}$-formula, since a negated modular linear equation is equivalent to a disjunction of modular linear equations (i.e., $k \neq b \bmod c \Leftrightarrow \bigvee_{0 \leq a \leq c, a \neq b} k = a \bmod c$). We say that an equation *appears* in a formula if it is a positive or negative literal in that formula.

Any subgroup G of $\mathbb{Z}^k$ can be given by a finite set of *generators*, i.e., k-tuples $\mathbf{g}^1, \dots, \mathbf{g}^m$, such that for every $\mathbf{g} \in G$, there are $\lambda_1, \dots, \lambda_m \in \mathbb{Z}$ such that $\mathbf{g} = \sum_i \lambda_i \mathbf{g}^i$, where we

write $\lambda \cdot \mathbf{g}$ for $(\lambda g_1, \dots, \lambda g_k)$. A *coset* of a subgroup G of $\mathbb{Z}^k$ is any set of the form $\mathbf{a} + G := \{\mathbf{a} + \mathbf{g} \mid \mathbf{g} \in G\}$, where $\mathbf{a} \in \mathbb{Z}^k$. By moving to a standard formula, we are in a position to deduce the following.

▶ **Proposition 2.** *Suppose R is a unary relation first-order definable in $(\mathbb{Z}; +, 1)$. Then R has the form $(R^\circ \cup R^+) \setminus R^-$, where R° is a finite union of cosets of nontrivial subgroups of $\mathbb{Z}$, and R^+ and R^- are finite disjoint sets of integers.*

Proof. Consider a disjunction φ of equations (possibly negated and modular equations). If this disjunction contains a negated equation $ax \neq c$, then φ defines a relation that contains $\mathbb{Z} \setminus \{c/a\}$ and is therefore as in the statement. Otherwise, φ contains only positive linear equation and modular equations, and the relation that φ defines is clearly of the form $R^\circ \cup R^+$ for some finite set R^+ and some union R° of nontrivial subgroups of $\mathbb{Z}$.

Consider a quantifier-free formula φ in conjunctive normal form defining R. Each conjunct defines a relation of the right form, per the previous paragraph. It is easily checked that a conjunction of relations of this form is again a relation of the form $(R^\circ \cup R^+) \setminus R^-$, so that we have proved that every quantifier-free formula with one free variables defines a relation of the right form. The proposition then follows from quantifier-elimination. ◀

Note that if $R^+ \cap R^\circ = \emptyset$ and $R^- \subset R^\circ$, then R^+, R^-, and R° are unique. We use the terminology with this convention for all unary relations R that are first-order definable in $(\mathbb{Z}; +, 1)$ throughout the article.

▶ **Definition 3.** Let φ be an $\mathcal{L}_{(\mathbb{Z};+,1)}$-formula. We say that φ is *Horn* if it is a conjunction of clauses of the form

$$\bigvee_{i=1}^{n} \neg\varphi_i \vee \varphi_0$$

where $\varphi_1, \dots, \varphi_n$ are linear equations and φ_0 is a linear or a modular linear equation.

▶ **Example 4.** Singletons, cofinite unary relations, and cosets of subgroups of $\mathbb{Z}^n$ are examples of Horn-definable relations.

3 Cores

If $\mathbb{A}$ is a first-order expansion of $(\mathbb{Z}; +)$, note that its endomorphisms are precisely of the form $x \mapsto \lambda x$ for some $\lambda \in \mathbb{Z}$. Therefore, we view in the following $\mathrm{End}(\mathbb{A})$ as a subset of $\mathbb{Z}$, where the monoid structure on $\mathrm{End}(\mathbb{A})$ implies that as a subset of $\mathbb{Z}$, it is closed under multiplication and contains 1. We say that $\mathbb{A}$ is a core if all its endomorphisms are self-embeddings, and that $\mathbb{B}$ is a core of $\mathbb{A}$ if $\mathbb{A}$ and $\mathbb{B}$ are homomorphically equivalent and $\mathbb{B}$ is a core.

▶ **Lemma 5.** *Let $\mathbb{A}$ be first-order definable in $(\mathbb{Z}; +, 1)$, and suppose that $\mathbb{A}$ contains $+$. There exists a structure which is a core of $\mathbb{A}$, and which is either a 1-element structure or first-order definable in $(\mathbb{Z}; +, 1)$ and containing $+$.*

Proof. If $0 \in \mathrm{End}(\mathbb{A})$ then the lemma is clearly true ($\mathbb{A}$ being homomorphically equivalent to the substructure of $\mathbb{A}$ induced by $\{0\}$), so let us assume that $0 \notin \mathrm{End}(\mathbb{A})$. Similarly we can assume that $\mathrm{End}(\mathbb{A}) \not\subseteq \{-1, 1\}$, otherwise $\mathbb{A}$ is already a core. For a quantifier-free formula ψ and an integer λ, define ψ/λ by induction on ψ as follows:

- if ψ is $\sum \lambda_i x_i = c$ and λ divides c, then ψ/λ is $\sum \lambda_i x_i = c/\lambda$,
- if ψ is $\sum \lambda_i x_i = c$ and λ does not divide c, then ψ/λ is $\bot$,

- if ψ is $\sum \lambda_i x_i = c \bmod d$ and $\ell := \gcd(\lambda, d)$ divides c, then ψ/λ is $\sum \lambda_i x_i = ec/\ell \bmod d/\ell$ where e is the inverse of λ/ℓ modulo d/ℓ,
- if ψ is $\sum \lambda_i x_i = c \bmod d$ and $\ell := \gcd(\lambda, d)$ does not divide c, then ψ/λ is $\bot$,
- extend to boolean combinations in the obvious fashion.

Note that for every tuple $\mathbf{a}$, we have that $\mathbf{a}$ satisfies ψ/λ iff $\lambda \cdot \mathbf{a}$ satisfies ψ. Indeed, if ψ is a linear equation then this is clear. Similarly, it is clear if ψ is a modular equation and $\ell := \gcd(\lambda, d)$ does not divide c. Suppose that ψ is a modular equation and $\ell := \gcd(\lambda, d)$ divides c. If $\sum \lambda\lambda_i x_i = c \bmod d$ then $\lambda/\ell \cdot (\sum \lambda_i x_i) = qd/\ell + c/\ell$ so that $e\lambda/\ell \cdot (\sum \lambda_i x_i) = (eq) \cdot d/\ell + ec/\ell$, where e is the inverse of λ/ℓ modulo d/ℓ and $q \in \mathbb{Z}$. We therefore obtain $\sum \lambda_i x_i = ec/\ell \bmod d/\ell$. Conversely if $\sum \lambda_i x_i = ec/\ell \bmod d/\ell$ then $\sum \lambda\lambda_i x_i = (\lambda e)c/\ell + (\frac{\lambda}{\ell} q)d = c \bmod d$.

Let ψ be any quantifier-free $\mathcal{L}_{(\mathbb{Z};+,1)}$-formula and suppose that $|\lambda| > 1$. The only cases where some magnitudes of the integers on the right-hand sides of terms in the formula ψ do not decrease by forming ψ/λ is when ψ only contains literals either of the form $\sum \lambda_i x_i = 0$ or of the form $\sum \lambda_i x_i = c \bmod d$ with λ and d coprime. Therefore, the sequence $\psi_0, \psi_1, \psi_2, \ldots$ where ψ_0 is ψ and where ψ_{i+1} is ψ_i/λ for some $\lambda \in \mathrm{End}(\mathbb{A})$ with $|\lambda| > 1$ reaches in a finite number of steps a fixpoint where all the literals are either of the form $\sum \lambda_i x_i = 0$ or are modular equations whose modulus d is such that λ and d are coprime. Let $n \geq 1$ be such that for every ψ defining a relation of $\mathbb{A}$, the formula ψ_n is a fixpoint. Let $\mathbb{B}$ be the structure whose domain is $\mathbb{Z}$ and whose relations are $+$ and the relations defined by ψ_n for each ψ defining a relation of $\mathbb{A}$.

We claim that $\mathbb{B}$ is homomorphically equivalent to $\mathbb{A}$ and is a core. The first claim is clear, since $\mathbb{B}$ is isomorphic to the structure obtained from $\mathbb{A}$ by successive applications of endomorphisms $x \mapsto \lambda \cdot x$ (in particular $\mathbb{B}$ embeds into $\mathbb{A}$). Let now $x \mapsto \lambda \cdot x$ be an endomorphism of $\mathbb{B}$, and suppose that $\mathbf{a}$ is a tuple in a relation R of $\mathbb{B}$. Then we have that $\lambda \cdot \mathbf{a}$ in R since $x \mapsto \lambda \cdot x$ is an endomorphism. Conversely, note that λ is coprime to d or else we would not have reached a fixed point in the previous stage. Thus, $\lambda^{\varphi(d)} = 1 \bmod d$, where $\varphi(d)$ here is the totient of d. It follows then that $\lambda^{\varphi(d)}\mathbf{a} = \mathbf{a} \bmod d$. Suppose $\lambda\mathbf{a} \in R$, then by applying $\varphi(d) - 1$ times an endomorphism, we derive $\lambda^{\varphi(d)}\mathbf{a} \in R$. It follows that $\mathbf{a} \in R$, for both the cases that atoms are of the form $\sum \lambda_i x_i = 0$ or are modular equations whose modulus d is such that λ and d are coprime. Hence, $x \mapsto \lambda \cdot x$ is an embedding of $\mathbb{A}$. ◀

We order the standard formulas lexicographically with respect to (in this order)

1. the number of non-Horn clauses,
2. the number of literals in clauses with at least two literals,
3. the number of all literals, and
4. the sum of the absolute values of all numbers appearing in an equation.

This order is used in a number of statements and proofs throughout the text, e.g., in Proposition 6, Lemma 17, and Theorem 18. A standard formula is *minimal* if no smaller formula is equivalent to it.

The following properties follow from the construction of cores in the previous proof.

▶ **Proposition 6.** *Let $\mathbb{A}$ be first-order definable in $(\mathbb{Z}; +, 1)$, and suppose that $\mathbb{A}$ contains $+$ and is a core. Let $\lambda \in \mathrm{End}(\mathbb{A})$. Let R be a relation of $\mathbb{A}$ and let φ be a minimal standard formula defining R.*

- *If $\sum \lambda_i x_i = c$ is a linear equation appearing in φ, then $c = 0$ or $|\lambda| = 1$.*
- *If $\sum \lambda_i x_i = c \bmod d$ is a modular linear equation in φ, then λ and d are coprime.*

Moreover, if $\mathrm{End}(\mathbb{A}) = 1 + d\mathbb{Z}$ for some $d \geq 2$, then every relation of $\mathbb{A}$ can be expressed with a minimal formula in which all modular linear equations are modulo a divisor of d.

Proof. The two items are clear from the proof of Lemma 5. For the last statement, let d' be a modulus appearing in a minimal definition of a relation of $\mathbb{A}$. By the second item, we have that d' and $1 + kd$ are coprime, for all $k \in \mathbb{Z}$. Let ℓ be such that $\ell d = -1 \bmod \frac{d'}{\gcd(d,d')}$. If d' and $1 + \ell d$ are coprime, there exist $u, v \in \mathbb{Z}$ such that $ud' + v(1 + \ell d) = 1$. Taking this equation modulo $\frac{d'}{\gcd(d,d')}$ we obtain $0 = 1 \bmod \frac{d'}{\gcd(d,d')}$, so that $\gcd(d, d') = d'$ and d' divides d. ◀

4 Hardness

Our sources of hardness come from *pp-interpretations*, that we define now. A structure $\mathbb{B}$ is said to be *one-dimensional pp-interpretable* in $\mathbb{A}$ if there exists a partial surjective map $h\colon A \to B$, called the *coordinate map*, such that the inverse image of every relation of $\mathbb{B}$ (including the equality relation and the unary relation B) under h has a pp-definition in A. Formally, we require that for every k-ary relation R of $\mathbb{B}$, there exists a pp-formula $\varphi_R(x_1, \dots, x_k)$ in the language of $\mathbb{A}$ such that

$$\mathbb{A} \models \varphi_R(a_1, \dots, a_k) \Leftrightarrow \mathbb{B} \models R(h(a_1), \dots, h(a_k))$$

holds for all $a_1, \dots, a_k \in A$. This requirement for the equality relation of $\mathbb{B}$ and the unary relation B implies that the kernel of h and its domain have a pp-definition in $\mathbb{A}$. It is well-known that if $\mathbb{B}$ is pp-interpretable in $\mathbb{A}$, then $\mathrm{CSP}(\mathbb{B})$ reduces in polynomial time to $\mathrm{CSP}(\mathbb{A})$.

4.1 The fully modular case

One of the sources of hardness for our problems are expansions of the *general subgroup problem* from [14]. The general subgroup problem of a finite abelian group G is the CSP of $(G; +)$ expanded with a k-ary relation for every coset $\mathbf{a} + H$, where H is a subgroup of G^k. It is known that this problem is solvable in polynomial time (under some reasonable encoding of the input); in modern parlance, this follows from the fact that the operation $(x, y, z) \mapsto x - y + z$ is a Maltsev polymorphism of the template. Feder and Vardi [14, Theorem 34] proved that the problem becomes NP-hard if the template is further expanded by any other relation.

The general subgroup problem of $\mathbb{Z}/d\mathbb{Z}$ can be viewed as a CSP of a first-order reduct of $(\mathbb{Z}; +, 1)$ whose relations are defined by quantifier-free formulas only containing modular linear equations. This motivates the following definition.

▶ **Definition 7.** A relation $R \subseteq \mathbb{Z}^k$ is called *fully modular* if it is definable by a conjunction of disjunctions of modular linear equations, in which case we can even assume that all the modular linear equations involved in such a definition of R have the same modulus $d \geq 1$.

▶ **Proposition 8.** *Let $\mathbb{A}$ be a finite-signature core which is first-order definable in $(\mathbb{Z}; +, 1)$ and contains $+$. Suppose that $\mathbb{A}$ has a fully modular relation that is not Horn-definable. Then $\mathrm{CSP}(\mathbb{A})$ is NP-complete.*

Proof. Let R be a relation of $\mathbb{A}$ that is not Horn-definable and fully modular, and let $d \geq 1$ be such that R can be defined with only linear equalities modulo d. Let $\mathbb{A}/d\mathbb{A}$ be the structure with domain $\mathbb{Z}/d\mathbb{Z}$ containing the ternary relation $+$ as well as a relation S' for every relation S of arity k of $\mathbb{A}$, defined by

$$S' = \{(a_1, \dots, a_k) \mid \exists q \in \mathbb{Z} : (qd + a_1, \dots, qd + a_k) \in S\}.$$

Note that $\mathbb{A}/d\mathbb{A}$ is pp-interpretable in $\mathbb{A}$: the coordinate map is the canonical projection $x \mapsto x \bmod d$, whose kernel is pp-definable by the formula $\varphi_=(x,y) := \exists z(x - y = dz)$. As a consequence, $\mathrm{CSP}(\mathbb{A}/d\mathbb{A})$ reduces in logarithmic space to $\mathrm{CSP}(\mathbb{A})$. Moreover, if $\mathbb{A}$ is a core then $\mathbb{A}/d\mathbb{A}$ is also a core. It follows from general principles [2, Proposition 3.3] that $\mathrm{CSP}(\mathbb{A}/d\mathbb{A}, 1)$ reduces to $\mathrm{CSP}(\mathbb{A}/d\mathbb{A})$ and so to $\mathrm{CSP}(\mathbb{A})$. Note that every coset of a subgroup of $(\mathbb{Z}/d\mathbb{Z})^k$ is pp-definable in $(\mathbb{A}/d\mathbb{A}, 1)$ and that if R is not Horn-definable then R' is not a coset of a subgroup. It follows from Theorem 34 in the bible [14] that $\mathrm{CSP}(\mathbb{A})$ is NP-complete. ◀

4.2 The unary case

In order to prove Theorem 1, we now focus on the case of *parametrised unary relations*.

▶ **Definition 9** (Compatibility). Let $\Lambda \subseteq \mathbb{Z} \setminus \{0\}$ be a set containing 1. We say that a set $\{S_\lambda\}_{\lambda\in\Lambda}$ of subsets of $\mathbb{Z}$ that are definable in $(\mathbb{Z}; +, 1)$ is *compatible* if there exist disjoint finite sets $A, B \subseteq \mathbb{Z}$ such that

- $S_\lambda = (S_\lambda^\circ \cup \lambda \cdot A) \setminus \lambda \cdot B$ for all $\lambda \in \Lambda$ and
- for all $d \geq 1$ and $c \in \{0, \dots, d-1\}$, we have $c + d\mathbb{Z} \subseteq S_1^\circ \Leftrightarrow \lambda c + d\mathbb{Z} \subseteq S_\lambda^\circ$.

▶ **Definition 10** (Uniform pp-definability). Let $\mathbb{A}$ be a first-order reduct of $(\mathbb{Z}; +, 1)$. We say that $\{S_\lambda\}_{\lambda\in\Lambda}$ is *uniformly pp-definable in* $\mathbb{A}$ if there exists a pp-formula $\theta(x, y)$ such that $a \in S_\lambda$ if, and only if, $\mathbb{A} \models \theta(\lambda, a)$.

Note that the definition of being uniformly pp-definable implies that Λ has a pp-definition in $\mathbb{A}$, for $\exists y.\, \theta(x, y)$ is a pp-definition. Let $S \subseteq \mathbb{Z}^2$ be a binary relation that is pp-definable in $\mathbb{A}$. Then the family $\{S_\lambda\}_{\lambda\in\Lambda}$ where $\Lambda := \{a \in \mathbb{Z} \mid (a, b) \in S \text{ for some } b \in \mathbb{Z}\} \subseteq \mathbb{Z} \setminus \{0\}$ and $S_\lambda := \{a \in \mathbb{Z} \mid (\lambda, a) \in S\}$ is uniformly pp-definable in $\mathbb{A}$. But even if S contains a tuple of the form $(1, b)$ and no tuple of the form $(0, b)$, it might not necessarily satisfy the compatibility condition, as illustrated in the following example.

▶ **Example 11.** Let $S = \{(a, b) \in \mathbb{Z}^2 \mid a \neq 0 \wedge (a = b \vee a = 2b)\}$. Then $\Lambda = \mathbb{Z} \setminus \{0\}$, and for $\lambda \in \Lambda$ we have $S_\lambda = \{\lambda\}$ if $\lambda = 1 \bmod 2$ and $S_\lambda = \{\lambda, \frac{\lambda}{2}\}$ if $\lambda = 0 \bmod 2$. Therefore, the compatibility condition is not satisfied by $\{S_\lambda\}_{\lambda\in\Lambda}$.

In the following proof, we write 1-in-3-SAT for $\mathrm{CSP}(\{0, 1\}; \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\})$. It is well-known that this problem is NP-complete ([22]; for a proof see [20]).

▶ **Lemma 12.** *Let $\mathbb{A}$ be a finite-signature first-order reduct of $(\mathbb{Z}; +, 1)$ containing $+$. If $\{S_\lambda\}_{\lambda\in\Lambda}$ is a compatible set of unary relations that is uniformly pp-definable in $\mathbb{A}$ and if $1 < |S_\lambda| < \infty$ for all $\lambda \in \Lambda$, then $\mathrm{CSP}(\mathbb{A})$ is NP-hard.*

Proof. Since every S_λ is finite, one sees that $S_\lambda = \lambda \cdot A$ for the finite set A coming from the compatibility condition. Let $m_1 := \min(A)$ and $m_2 := \min(A \setminus \{m_1\})$. The formula

$$\exists \lambda(x + y + z = (m_2 - m_1)\lambda \wedge x + m_1\lambda \in S_\lambda \wedge y + m_1\lambda \in S_\lambda \wedge z + m_1\lambda \in S_\lambda \wedge \lambda \in \Lambda)$$

defines the ternary relation consisting of $(a, b, c) \in \mathbb{Z}^3$ such that $a, b, c \in \{0, m_2 - m_1\}$ and exactly one of a, b, c is equal to $m_2 - m_1$. Note that this formula is in the language of $\mathbb{A}$, since $\{S_\lambda\}_{\lambda\in\Lambda}$ is uniformly pp-definable and in particular Λ is pp-definable in $\mathbb{A}$. This gives an interpretation of 1-in-3-SAT in $\mathbb{A}$, using the map $h\colon \{0, m_2 - m_1\} \to \{0, 1\}$ such that $h(0) = 0$ and $h(m_2 - m_1) = 1$. Therefore, $\mathrm{CSP}(\mathbb{A})$ is NP-hard. ◀

▶ **Proposition 13.** *Let $\mathbb{A}$ be a finite-signature first-order reduct of $(\mathbb{Z}; +, 1)$ that contains $+$ and is a core. Let $\{S_\lambda\}_{\lambda\in\Lambda}$ be a compatible family that is uniformly pp-definable in $\mathbb{A}$ such that for every $\lambda \in \Lambda$ the set S_λ is not Horn-definable. Then* CSP($\mathbb{A}$) *is NP-hard.*

Proof. Let $A, B \subset \mathbb{Z}$ be finite such that $S_\lambda = (S_\lambda^\circ \cup \lambda \cdot A) \setminus (\lambda \cdot B)$ for all $\lambda \in \Lambda$. Since S_λ is not Horn-definable, we have $|S_\lambda| > 1$ for all $\lambda \in \Lambda$. If S_λ is finite for every $\lambda \in \Lambda$, then CSP($\mathbb{A}$) is NP-hard by Lemma 12. Therefore, we can assume that $S_\lambda^\circ \neq \emptyset$ for some $\lambda \in \Lambda$, and the second compatibility condition implies that S_λ° is infinite for all $\lambda \in \Lambda$. Let $d \geq 1$ be such that S_λ° is a union of cosets of $d\mathbb{Z}$ for all $\lambda \in \Lambda$. Write $S_1^\circ = \bigcup_{i=1}^n c_i + d\mathbb{Z}$, with $c_i \in \{0, \dots, d-1\}$.

If $n \in \{2, \dots, d-1\}$, we claim that we can pp-define a fully modular relation that is not Horn-definable. Indeed, let $\theta(x, y)$ be a formula that defines $\{S_\lambda\}_{\lambda\in\Lambda}$. Note that

$$\chi(x, y) := \theta(x, y) \wedge \theta(x, y + dx) \wedge \cdots \wedge \theta(x, y + \max(A \cup B)dx)$$

holds precisely on the pairs (λ, a) such that $a \in S_\lambda^\circ$: since x is forced to be in Λ by θ, a satisfying assignment gives a nonzero value λ to x. Thus, if all of $y, y + d\lambda, \dots, y + \max(A \cup B)d\lambda$ are in S_λ, then they all must be in the modular part S_λ°. The relation T that χ defines is fully modular and is such that $T_\lambda = S_\lambda^\circ$ and in particular T is not Horn-definable. It follows from Proposition 8 that CSP($\mathbb{A}$) is NP-hard.

Otherwise, the set S_λ° consists of a single coset of $d\mathbb{Z}$ for all $\lambda \in \Lambda$, and this coset is $\lambda c_1 + d\mathbb{Z}$ by the compatibility condition on $\{S_\lambda\}_{\lambda\in\Lambda}$. Since S_λ° is assumed to not be Horn-definable, A must contain an element a. We claim that we can define another family of unary relations where the unary relations are finite and not singletons. Indeed, consider the formula

$$\psi(x, y) := \exists z\, (\theta(x, y) \wedge \theta(x, z) \wedge y + z = (c_1 + a)x)$$

and let $T \subseteq \mathbb{Z}^2$ be the relation that it defines. First, note that $\psi(\lambda, c_1)$ and $\psi(\lambda, a)$ hold for all $\lambda \in \Lambda$, so that $|T_\lambda| > 1$. We claim that T_λ is finite. Since $A \cap S_1^\circ = \emptyset$, one has $a \neq c_1 \bmod d$. Consequently, $c_1 + a \neq 2c_1 \bmod d$ and $(c_1 + a)\lambda \neq 2c_1\lambda \bmod d$. The equation $y + z = (c_1 + a)\lambda$ therefore forces that one of y and z is in $\lambda \cdot A$. Since A is finite, there are only finitely many pairs satisfying this condition, thus showing that $1 < |T_\lambda| < \infty$. It follows from Lemma 12 that CSP($\mathbb{A}$) is NP-hard. ◀

As a corollary we obtain a simple-to-state condition implying that CSP($\mathbb{A}$) is NP-hard (Corollary 16). The corollary relies on the fact that End($\mathbb{A}$), being identified with a subset of $\mathbb{Z}$, can be pp-defined in $\mathbb{A}$. We prove this in the next lemma.

▶ **Lemma 14.** *Let $\mathbb{A}$ be a finite-signature first-order reduct of $(\mathbb{Z}; +, 1)$ that contains $+$. Then the set* End($\mathbb{A}$) *has a pp-definition in $\mathbb{A}$ that is additionally quantifier-free.*

Proof. Let E be the set of all the formulas $R(a_1 \cdot x, \dots, a_r \cdot x)$ for R in the language of $\mathbb{A}$ and $(a_1, \dots, a_r) \in R$. We then have that $\mathbb{A} \models E(\lambda)$ iff $\lambda \in$ End($\mathbb{A}$). We now show that there exists a finite subset $F \subseteq E$ that defines the same set of integers.

For each relation R of $\mathbb{A}$, fix a standard definition φ_R in conjunctive normal form of R in $(\mathbb{Z}; +, 1)$. Let M be the largest absolute value of a constant appearing in φ_R. Consider the finite family $\mathcal{F}$ of equations $\sum \mu_i x_i = m$, where $\sum \mu_i x_i = m'$ is some equation appearing in φ_R and $|m| \leq M$, together with all the equations $\sum \mu_i x_i = c \bmod d$ where $\sum \mu_i x_i = c' \bmod d$ is a modular equation appearing in φ_R and $c \in \{0, \dots, d-1\}$. For each subset of $\mathcal{F}$ that is satisfiable by a tuple in R, pick a tuple $\mathbf{b} \in R$ satisfying the formulas in this subset and add this tuple to a set $\mathcal{S}$. Repeat this operation for every relation of $\mathbb{A}$, and let $\mathcal{S}$ be the finite set of tuples (of possibly different arities) that we obtain. Finally, let F be the subset of E where only the formulas associated with tuples from $\mathcal{S}$ are kept.

We claim that F defines $\mathrm{End}(\mathbb{A})$. Since $F \subseteq E$, it suffices to show that every λ satisfying F is an endomorphism of $\mathbb{A}$. Let $\lambda \in \mathbb{Z}$ satisfy F, and let $\mathbf{a} \in R$ be a tuple in some relation of $\mathbb{A}$. Let $\mathbf{b} \in \mathcal{S}$ be such that $\mathbf{b}$ satisfies exactly the same equations in $\mathcal{F}$ as $\mathbf{a}$. By construction, $\lambda\mathbf{b} \in R$ so that in each clause of φ_R, some equation is satisfied by $\lambda\mathbf{b}$. We show that $\lambda\mathbf{a}$ satisfies the same equations, so that $\lambda\mathbf{a} \in R$. If $\lambda = 0$, then $\lambda\mathbf{b} = \lambda\mathbf{a}$ so that $\lambda\mathbf{a} \in R$. Suppose now that $\lambda \neq 0$. Let $\sum \mu_i x_i = c$ be a linear equation that is satisfied by $\lambda\mathbf{b}$. Then necessarily λ divides c, so that $\mathbf{b}$ satisfies $\sum \mu_i x_i = \frac{c}{\lambda}$ and $|\frac{c}{\lambda}| \leq |c| \leq M$, so that $\sum \mu_i x_i = \frac{c}{\lambda}$ is an equation in $\mathcal{F}$. Consequently, $\mathbf{a}$ also satisfies this equation and $\lambda\mathbf{a}$ satisfies $\sum \mu_i x_i = c$. The proof for modular linear equations is similar. This proves that λ is an endomorphism of $\mathbb{A}$ and concludes the proof. ◀

▶ **Lemma 15.** *Let $R \subseteq \mathbb{Z}$ be first-order definable over $(\mathbb{Z}; +, 1)$ such that $(\mathbb{Z}; +, R)$ is a core.*
- *If $R^+ \neq \emptyset$, then $\{1\}$ or $\{1, -1\}$ is pp-definable in $(\mathbb{Z}; +, R)$.*
- *If $R^+ = \emptyset$, then $R^- = \emptyset$ or $R = \mathbb{Z} \setminus \{0\}$.*

Proof. Let n be such that R° is a union of n cosets of $d\mathbb{Z}$, i.e.,

$$R^\circ = \bigcup_{i=1}^{n} c_i + d\mathbb{Z}.$$

Let us prove the first item. By Lemma 14, it suffices to prove that the only possible endomorphisms of the structure $(\mathbb{Z}; +, R)$ are $x \mapsto \lambda \cdot x$ with $\lambda \in \{1, -1\}$. Suppose that $x \mapsto \lambda \cdot x$ is an endomorphism. Then $\lambda \neq 0$ since the structure is a core, so suppose that $|\lambda| > 1$. Let a be the maximal element of R^+, and note that in particular $a + d \notin R$ (it cannot be in R^+ because of the maximality assumption, and cannot be equal to any c_i modulo d). Then $a \in R$, so $\lambda^q a \in R$ for all $q \in \mathbb{N}$. In particular, if q is such that $\lambda^q > \max(R^+ \cup R^-)$ we obtain $\lambda^q a \in R^\circ$. This means that $\lambda^q a = c_i \bmod d$ for some $i \in \{1, \ldots, n\}$. Finally, $\lambda^q(a + d) = \lambda^q a + \lambda^q d = c_i \bmod d$, so that $\lambda^q(a + d) \in R$. This implies that $x \mapsto \lambda^q \cdot x$ is not an embedding, contradicting the core assumption on $(\mathbb{Z}; +, R)$.

Let us now prove the second item. Let b be some element of R^-. We must have $b = c_i \bmod d$ for some $i \in \{1, \ldots, n\}$ since $R^- \subset R^\circ$. Note that the map $x \mapsto (d + 1)x$ is an endomorphism of $(\mathbb{Z}; +, R)$, so it has to be an embedding. It follows that $(d + 1)^m \cdot b \notin R$ for any m. Suppose that b is not 0. Choose m so that $(d + 1)^m \cdot |b| > \max_{e \in R^-} |e|$ so that $(d + 1)^m \cdot b \notin R^-$. But $(d + 1)^m b = c_i \bmod d$, a contradiction. It follows that $R^- \subseteq \{0\}$, which concludes the proof. ◀

▶ **Corollary 16.** *Let $\mathbb{A}$ be a finite-signature first-order reduct of $(\mathbb{Z}; +, 1)$ which contains $+$ and is a core. If $\mathrm{End}(\mathbb{A})$ is not Horn-definable, then $\mathrm{CSP}(\mathbb{A})$ is NP-hard. Moreover, if $\mathrm{End}(\mathbb{A})$ is Horn-definable, then it is either $\{1\}$, $\mathbb{Z} \setminus \{0\}$, or $1 + d\mathbb{Z}$ for some $d \geq 2$.*

Proof. Lemma 14 implies that $R := \mathrm{End}(\mathbb{A})$ has a quantifier-free pp-definition in $\mathbb{A}$. We first prove that $(\mathbb{Z}; +, R)$ is a core. Indeed, let λ be an endomorphism of $(\mathbb{Z}; +, R)$. Since $1 \in R$, we obtain that $\lambda \in R$, so that $x \mapsto \lambda x$ is a self-embedding of $\mathbb{A}$ by the fact that $\mathbb{A}$ is a core. Since R is has a quantifier-free definition over $\mathbb{A}$, it follows that $x \mapsto \lambda x$ is also a self-embedding of $(\mathbb{Z}; +, R)$.

First consider the case that R is not Horn-definable. If $R^+ \neq \emptyset$ then Lemma 15 implies that $\{1\}$ or $\{1, -1\}$ are pp-definable in $(\mathbb{Z}; +, R)$. All endomorphisms of $\mathbb{A}$ must preserve this set, so $\mathrm{End}(\mathbb{A}) = R = \{1, -1\}$ or $\mathrm{End}(\mathbb{A}) = R = \{1\}$; since R is no Horn-definable, we must even have $R = \{1, -1\}$. But then the family $\{S_\lambda\}_{\lambda \in \{-1,1\}}$ with $S_{-1} = S_1 = \{1, -1\}$ is uniformly definable and compatible, the conditions being satisfied for $A = \{-1, 1\}$ and $B = \emptyset$:

- $S_1 = A = S_{-1}$;
- $S_1^\circ = S_{-1}^\circ = \emptyset$.

Then Proposition 13 applied to S_λ implies that CSP($\mathbb{A}$) is NP-hard.

If $R^+ = \emptyset$ then Lemma 15 implies that $R^- = \emptyset$ or that $R = \mathbb{Z} \setminus \{0\}$. In the latter case, R would be Horn, contrary to the assumptions, so $R^- = \emptyset$. In this case, R is fully modular, but not Horn definable, so NP-hardness of CSP($\mathbb{A}$) follows from Proposition 8. This shows the first part of the statement.

Finally, consider the case that R is Horn-definable. If $R^+ = \emptyset$ then Lemma 15 implies that $R = \mathbb{Z} \setminus \{0\}$, and we are done, or $R^- = \emptyset$, in which case $R = 1 + d\mathbb{Z}$ for some $d \geq 2$ and we are also done. Otherwise, $R^+ \neq \emptyset$ and Lemma 15 implies that $\{1\}$ or $\{1, -1\}$ is pp-definable in $(\mathbb{Z}; +, R)$. ◀

4.3 Arbitrary arities

We finally present the hardness proof in the general case where the structure contains a relation that is not Horn-definable. The strategy is to cut from a non-Horn relation R a uniformly definable family $\{S_\lambda\}_{\lambda \in \Lambda}$ of lines for which each S_λ is not Horn-definable. In a second step, we ensure that we get a family satisfying the compatibility condition, and we conclude using Proposition 13. Call a formula φ in conjunctive normal form *reduced* if removing any literal or clause from φ yields a formula that is not equivalent to φ. Note that minimal formulas are necessarily reduced.

▶ **Lemma 17.** *Let $\mathbb{A}$ be a finite-signature first-order reduct of $(\mathbb{Z}; +, 1)$ which contains $+$ and is a core. Suppose that $\mathbb{A}$ contains a relation R that is not Horn-definable. Then* CSP($\mathbb{A}$) *is NP-hard, or $\mathbb{A}$ pp-defines a relation that is not Horn-definable and that has a minimal definition containing a non-Horn clause ψ such that:*

- *no negated linear equation is in ψ,*
- *at least one linear equation is in ψ.*

Proof. Let φ be a standard minimal definition of R in conjunctive normal form, and let ψ be a clause of φ that is not Horn. From Corollary 16, we can suppose that End($\mathbb{A}$) is $\{1\}$, $\mathbb{Z} \setminus \{0\}$, or $1 + d\mathbb{Z}$ for $d \geq 2$. This implies that either $\{1\}$ is pp-definable or, by Proposition 6, all the linear equations appearing in φ are homogeneous.

We can assume that ψ does not contain any negative literal, per the assumption that φ is minimal: indeed, consider the relation R' defined by the formula

$$\varphi' := \varphi \wedge \sum \lambda_i x_i = c \tag{†}$$

where $\sum \lambda_i x_i \neq c$ is in ψ. Either $c = 0$, in which case the relation R' defined by (†) is pp-definable in $\mathbb{A}$, or $\{1\}$ is pp-definable in $\mathbb{A}$ and R' is pp-definable in $\mathbb{A}$, too. The relation R' is not Horn-definable, and when we reduce the definition φ' of R' we obtain a formula that has fewer literals in clauses that contain more than one literal, in contradiction to the minimality of φ.

If ψ contains a linear equation then we are done. Otherwise, ψ only contains modular linear equations. If End($\mathbb{A}$) $= \mathbb{Z} \setminus \{0\}$ then by Proposition 6 any modulus of a modular linear equation appearing in ψ would have to be coprime with every nonzero integer, which is impossible. Therefore, End($\mathbb{A}$) is $\{1\}$ or $1 + d\mathbb{Z}$ for $d \geq 2$. In the latter case, we can assume by Proposition 6 that all the modular linear equations in ψ are modulo a divisor of d. In the former case, let d be a common multiple of all the moduli appearing in a modular linear equation in ψ. Consider the structure $\mathbb{A}/d\mathbb{A}$ defined in Proposition 8. The relation T

obtained from R in this structure is not a coset of a subgroup H of $(\mathbb{Z}/d\mathbb{Z})^k$ (where k is the arity of R): otherwise this coset is definable by a conjunction θ of modular linear equations modulo a divisor of d. Replacing ψ by θ in φ would produce a smaller definition of R, a contradiction to the minimality of φ. Moreover, $(\mathbb{A}/d\mathbb{A}, 1)$ is pp-interpretable in $\mathbb{A}$: in the two cases that $\mathrm{End}(\mathbb{A}) = \{1\}$ and $\mathrm{End}(\mathbb{A}) = 1 + d\mathbb{Z}$, the preimage of $\{1\}$ under the canonical projection $x \mapsto x \bmod d$ is pp-definable in $\mathbb{A}$. We conclude as in Proposition 8 that $\mathrm{CSP}(\mathbb{A})$ is NP-hard. ◀

▶ **Theorem 18.** *Let $\mathbb{A}$ be a finite-signature first-order reduct of $(\mathbb{Z}; +, 1)$ which contains $+$ and is a core. Suppose that $\mathbb{A}$ contains a relation that is not Horn-definable. Then* $\mathrm{CSP}(\mathbb{A})$ *is NP-hard.*

Proof. From Lemma 17, we can suppose that $\mathbb{A}$ pp-defines a relation R that is not Horn-definable, that has a reduced standard definition φ containing a non-Horn clause ψ with at least one linear equation (L) and no negated linear equation. Since ψ is not Horn, it contains at least another equation (L'), possibly modular. Let $(a_1, \ldots, a_n)$ satisfy φ and only (L) in ψ. Such a tuple exists by the assumption that φ is reduced. Similarly, let $(b_1, \ldots, b_n)$ satisfy φ and only (L') in ψ. Let S be the binary relation such that $(\lambda, t) \in S$ if, and only if, $\lambda \in \mathrm{End}(\mathbb{A})$ and $t(\mathbf{a} - \mathbf{b}) + \lambda\mathbf{b}$ is in R. Note that $\mathbf{a}$ and $\mathbf{b}$ being fixed, S is pp-definable over $\mathbb{A}$. Therefore, we obtain a family $\{S_\lambda\}_{\lambda \in \Lambda}$ that is uniformly definable in $\mathbb{A}$, where $\Lambda = \mathrm{End}(\mathbb{A})$. Clearly, $1 \in \Lambda$, and $\Lambda \subseteq \mathbb{Z} \setminus \{0\}$ and $0, \lambda \in S_\lambda$. Moreover, note that

$$S_\lambda \cap \lambda \cdot \mathbb{Z} = \lambda \cdot S_1 \tag{‡}$$

holds for all $\lambda \in \mathrm{End}(\mathbb{A})$. Indeed:

$$\begin{aligned} t \in S_1 &\Leftrightarrow t(\mathbf{a} - \mathbf{b}) + \mathbf{b} \in R \\ &\Leftrightarrow \lambda t(\mathbf{a} - \mathbf{b}) + \lambda\mathbf{b} \in R && \text{because } \mathbb{A} \text{ is a core} \\ &\Leftrightarrow \lambda t \in S_\lambda. \end{aligned}$$

We prove that for all $\lambda \in \mathrm{End}(\mathbb{A})$ the relation S_λ is not Horn-definable. Since $0, \lambda \in S_\lambda$, it suffices to prove that S_λ omits infinitely many multiples of λ, and by (‡) it suffices to prove that $\ell \notin S_1$ for infinitely many ℓ. Let ℓ be such that $\ell = 1 \bmod d'$, for every modulus d' appearing in ψ. We claim that $\ell(\mathbf{a} - \mathbf{b}) + \mathbf{b}$ does not satisfy any modular linear equation in ψ. Indeed, let $\sum_i \sigma_i x_i = c \bmod d'$ be such a modular linear equation. Then we have

$$\sum_i \sigma_i(\ell(a_i - b_i) + b_i) = c \bmod d' \Leftrightarrow \sum_i \sigma_i a_i = c \bmod d',$$

which is a contradiction to the choice of $\mathbf{a}$ since $\mathbf{a}$ only satisfies (L) in ψ and (L) is assumed to be non-modular. Consider now a linear equation $\sum \sigma_i x_i = c$ in ψ. This equation is satisfied by $\ell(\mathbf{a} - \mathbf{b}) + \mathbf{b}$ if, and only if

$$\ell \cdot \sum_i \sigma_i(a_i - b_i) = c - \sum_i \sigma_i b_i\,. \tag{⋆}$$

Suppose first that $\sum \sigma_i(a_i - b_i) = 0$. Then (⋆) is satisfied if, and only if, we have $\sum \sigma_i b_i = c = \sum \sigma_i a_i$. This implies that both $\mathbf{a}$ and $\mathbf{b}$ satisfy the equation; this is a contradiction to our choice of the vectors $\mathbf{a}$ and $\mathbf{b}$, so that $\ell(\mathbf{a} - \mathbf{b}) + \mathbf{b}$ does not satisfy (⋆). Suppose now that $\sum \sigma_i(a_i - b_i) \neq 0$. If $\ell > |c - \sum \sigma_i b_i|$, it is then clear that (⋆) is not satisfied. Therefore, for infinitely many ℓ, the tuple $\ell(\mathbf{a} - \mathbf{b}) + \mathbf{b}$ does not satisfy any literal in ψ and $\ell \notin S_1$.

Let $\theta(x,y)$ be a minimal reduced standard definition of S. By inspection of the formula θ, one finds that S^+_λ and S^-_λ consist of points of the form $-\frac{a\lambda}{b}$, where $ax+by=0$ is an equation in θ. Note that since a and b are taken coprime by the minimality of θ, if b divides $a\lambda$ then b divides λ. Let $m := \mathrm{lcm}\{|b| : ax+by=0 \text{ is an equation in } \theta\}$, and note that $m\mathbb{Z}\cap\Lambda$ is not empty by the previous remark. Let $T \neq \emptyset$ be the binary relation defined by $\theta(m\cdot x, y)$. For all $\lambda \in \mathrm{End}(\mathbb{A})$, the set T_λ is not Horn-definable and of the form $(T^\circ_\lambda \cup (\lambda\cdot P)) \setminus (\lambda \cdot Q)$, where P and Q are finite sets that are independent of λ and $T^\circ_\lambda = S^\circ_{m\lambda}$. For the family of relations $\{T_\lambda\}_{\lambda\in\mathrm{End}(\mathbb{A})}$ to satisfy the compatibility condition, it remains to prove that $c+d\mathbb{Z}\subseteq T^\circ_1$ if, and only if, $\lambda c+d\mathbb{Z}\subseteq T^\circ_\lambda$, for all $d\geq 1$ and $c\in\{0,\dots,d-1\}$. Suppose that $c+d\mathbb{Z}\subseteq T^\circ_1$ for some $d\geq 1$ and suppose that the cosets of T°_λ are cosets of $d'\mathbb{Z}$. By Proposition 6, λ and d' are coprime. Therefore, there exists $\mu\in\mathbb{Z}$ such that $\lambda\mu = 1 \bmod d'$. We have $c+d\mu\mathbb{Z}\subseteq T^\circ_1$, because d divides $d\mu$. It follows that $\lambda c+\lambda d\mu\mathbb{Z}\subseteq T^\circ_\lambda$. Now, let $x\in\lambda c+d\mathbb{Z}$, say $x=\lambda c+qd$. Then we have $x=\lambda c+q\lambda\mu d \bmod d'$. Note that $\lambda c+q\lambda\mu d\in\lambda c+\lambda d\mu\mathbb{Z}\subseteq T^\circ_\lambda$. Since the cosets in T°_λ are cosets of $d'\mathbb{Z}$, we obtain that $x\in T^\circ_\lambda$ and consequently that $\lambda c+d\mathbb{Z}\subseteq T^\circ_\lambda$. Conversely, if $\lambda c+d\mathbb{Z}\subseteq T^\circ_\lambda$ then $\lambda c+d\lambda\mathbb{Z}\subseteq T^\circ_\lambda$, because d divides $d\lambda$. Since $\mathbb{A}$ is a core, $x\mapsto\lambda\cdot x$ is a self-embedding of $\mathbb{A}$, so that $c+d\mathbb{Z}\subseteq T^\circ_1$.

To conclude, the family of compatible relations $\{T_\lambda\}_{\lambda\in\mathrm{End}(\mathbb{A})}$ is uniformly pp-definable in $\mathbb{A}$ and consists of non-Horn relations. By Proposition 13, we obtain that $\mathrm{CSP}(\mathbb{A})$ is NP-hard. ◀

We illustrate our proofs in some examples below.

▶ **Example 19.** Consider the binary relation

$$S = \{(\lambda,t) \mid (\lambda = 1 \bmod 4 \wedge t = 1 \bmod 4) \vee (\lambda = 3 \bmod 4 \wedge t = 3 \bmod 4) \vee t = 0\}.$$

One sees that the set of endomorphisms of $\mathbb{A} := (\mathbb{Z};+,S)$ is equal to $\mathrm{End}(\mathbb{A}) := 1+2\mathbb{Z}$. Moreover, S is not Horn-definable. For every $\lambda\in\mathrm{End}(\mathbb{A})$, one has $S_\lambda = \{0\}\cup(\lambda+4\mathbb{Z})$. When λ is fixed, one can define a finite set by $\exists y(x\in S_\lambda \wedge y\in S_\lambda \wedge x+y=\lambda)$, which defines $\{0,\lambda\}$. One then obtains a reduction from 1-in-3-SAT by $\exists x,y,z\in\{0,\lambda\} : x+y+z\in\{0,\lambda\}$. Finally, by existentially quantifying over $\lambda\in\mathrm{End}(\mathbb{A})$ we obtain a reduction from 1-in-3-SAT to $\mathrm{CSP}(\mathbb{Z};+,S)$.

▶ **Example 20.** Let $R := \{0\}\cup(1+3\mathbb{Z})\cup(2+3\mathbb{Z})$ and $K = 1+3\mathbb{Z}$. Note that Proposition 13 does not apply to $\mathrm{CSP}(\mathbb{Z};+,R)$ since $(\mathbb{Z};+,R)$ is not a core (we have $0\in\mathrm{End}(\mathbb{Z};+,R)$). Neither does Corollary 16 apply to $\mathrm{CSP}(\mathbb{Z};+,R,K)$ since $\mathrm{End}(\mathbb{Z};+,R,K)=K$, which is clearly Horn-definable in $(\mathbb{Z};+,R,K)$. But one obtains hardness of $\mathrm{CSP}(\mathbb{Z};+,R,K)$ by Theorem 18. Indeed, pick $a=0$ (satisfying the linear equation $x=0$ in the definition of R) and $b=1$ (satisfying the modular linear equation $x=1 \bmod 3$ in the definition of R), and define the relation $S=\{(\lambda,t)\mid\lambda\in K\wedge\lambda-t\in S\}$. Note that for all $\lambda\in K$, we have $S_\lambda=\{\lambda\}\cup 3\mathbb{Z}\cup(2+3\mathbb{Z})$. The formula $\exists w(w\in K\wedge S(\lambda,t)\wedge S(\lambda,t+3w))$ defines the relation $T=\{(\lambda,t)\mid\lambda=1 \bmod 3\wedge(t=0 \bmod 3\vee t=2 \bmod 3)\}$, which is fully modular and not Horn-definable. Proposition 8 implies that $\mathrm{CSP}(\mathbb{Z};+,R,S)$ is NP-hard.

5 Tractability

In this section we show the following.

▶ **Proposition 21.** *Let $\mathbb{A}$ be a structure with finite relational signature, domain $\mathbb{Z}$, and whose relations have quantifier-free Horn definitions over $(\mathbb{Z};+,1)$. Then there is an algorithm that solves CSP($\mathbb{A}$) in polynomial time.*

This result follows from the following more general result.

▶ **Theorem 22.** *Let φ be a quantifier-free Horn formula over $(\mathbb{Z}; +)$, allowing parameters from $\mathbb{Z}$ represented in binary. Then there exists a polynomial-time algorithm to decide whether φ is satisfiable over $(\mathbb{Z}; +)$.*

The proof of Theorem 22 can be found at the end of this section. We first show how to derive Proposition 21.

Proof of Proposition 21. The input of CSP($\mathbb{A}$) consists of a primitive positive sentence whose atomic formulas are of the form $R(x_1, \ldots, x_k)$ where R is quantifier-free Horn definable over $\mathcal{L}_{(\mathbb{Z};+,1)}$. Since $\sum_{i=1}^{n} a_i x_i = b \bmod c$ is equivalent to $\sum_{i=1}^{n} a_i x_i = b + ck$, where k is a new integer variable, we can as well assume that the input to our problem consists of a set of Horn clauses over $(\mathbb{Z}; +, 1)$. This is tacitly the process of quantifier introduction, the converse of quantifier elimination. Then apply Theorem 22. ◀

Our algorithm for the proof of Theorem 22 uses two other well-known algorithms:

1. a polynomial-time algorithm for satisfiability of linear diophantine equations, i.e., the subproblem of the computational problem from Theorem 22 where the input only contains atomic formulas (see, e.g., [23]).
2. a polynomial-time algorithm to compute the rank of a matrix over $\mathbb{Q}$; this allows us in particular to decide whether a given linear system of equalities implies another equality over the rationals (this is standard, using Gaussian elimination; again, see [23] for a discussion of the complexity).

These two algorithms can be combined to obtain the following.

▶ **Lemma 23.** *There is a polynomial-time algorithm that decides whether a given system Φ of linear diophantine equations implies another given diophantine equation ψ over $\mathbb{Z}$.*

Proof. First, use the first algorithm above to test whether Φ has a solution over $\mathbb{Z}$. If no, return yes (false implies everything). If yes, we claim that Φ implies ψ over $\mathbb{Q}$ (which can be tested by the second algorithm above) if and only if Φ implies ψ over the integers. Clearly, if every rational solution of Φ satisfies ψ, then so does every integer solution. Suppose now that there exists a rational solution α to Φ which does not satisfy ψ. Also take an integer solution β to Φ. Then on the line L that goes through α and β there are infinitely many integer points. If infinitely many points on a line satisfy ψ, then all points of the line must satisfy ψ. Since $\alpha \in L$ does not satisfy ψ it follows that an integer point on L does not satisfy ψ, i.e., Φ does not imply ψ over the integers. ◀

Given the two mentioned algorithms, our procedure for the proof of Theorem 22 is basically an implementation of positive unit clause resolution. It takes the same form as the algorithm presented in [7] for satisfiability over the rationals.

Proof. We follow the proof of Proposition 3.1 from [7]. We first discuss the correctness of the algorithm.

When $\mathcal{U}$ logically implies φ (which can be tested with the algorithm from Lemma 23) then the negative literal $\neg\varphi$ is never satisfied and can be deleted from all clauses without affecting the set of solutions. Since this is the only way in which literals can be deleted from clauses, it is clear that if one clause becomes empty the instance is unsatisfiable.

If the algorithm terminates with *satisfiable*, then no negation of an inequality is implied by $\mathcal{U}$. If r is the rank of the linear equation system defined by $\mathcal{U}$, we can use Gaussian

```
// Input: a set of Horn-clauses C over (ℤ; +) with parameters.
// Output: satisfiable if C is satisfiable in (ℤ; +), unsatisfiable otherwise
Let U be clauses from C that only contain a single positive literal.
If U is unsatisfiable then return unsatisfiable.
Do
    For all negative literals ¬φ in clauses from C
        If U implies φ, then delete the negative literal ¬φ from all clauses in C.
    If C contains an empty clause, then return unsatisfiable.
    If C contains a clause with a single positive literal ψ, then add {ψ} to U.
Loop until no literal has been deleted
Return satisfiable.
```

Figure 1 An algorithm for satisfiability of Horn formulas with parameters over $(\mathbb{Z}; +)$.

elimination to eliminate r of the variables from all literals in the remaining clauses. For each of the remaining inequalities, consider the sum of absolute values of all coefficients. Let S be one plus the maximum of this sum over all the remaining inequalities. Then setting the i-th variable to S^i satisfies all clauses. To see this, take any inequality, and assume that i is the highest variable index in this inequality. Order the inequality in such a way that the variable with highest index is on one side and all other variables on the other side of the $\neq$ sign. The absolute value on the side with the i-th variable is at least S^i. The absolute value on the other side is less than $S^i - S$, since all variables have absolute value less than S^{i-1} and the sum of all coefficients is less than $S - 1$ in absolute value. Hence, both sides of the inequality have different absolute value, and the inequality is satisfied. Since all remaining clauses have at least one inequality, all constraints are satisfied.

Now let us address the complexity of the algorithm. With appropriate data structures, the time needed for removing negated literals $\neg\varphi$ from all clauses when φ is implied by $\mathcal{U}$ is linearly bounded in the input size since each literal can be removed at most once. ◀

6 Conclusion

We are finally in position to prove the main result.

Proof of Theorem 1. Let $\mathbb{A}$ be a finite-signature first-order reduct of $(\mathbb{Z}; +, 1)$ that $\mathbb{A}$ contains $+$. By Lemma 5 there exists a core $\mathbb{B}$ of $\mathbb{A}$. If $\mathbb{B}$ has only one element then CSP($\mathbb{B}$) and CSP($\mathbb{A}$) are trivially in P. Otherwise, $\mathbb{B}$ is itself first-order definable in $(\mathbb{Z}; +, 1)$ and contains $+$ by Lemma 5, and the statement follows from Theorem 18 and Proposition 21. ◀

References

1 Libor Barto, Michael Kompatscher, Miroslav Olsák, Trung Van Pham, and Michael Pinsker. The equivalence of two dichotomy conjectures for infinite domain constraint satisfaction problems. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005128.

2 Libor Barto, Jakub Opršal, and Michael Pinsker. The wonderland of reflections. *Israel Journal of Mathematics*, 2017. To appear. Preprint arXiv:1510.04521.

3 Libor Barto and Michael Pinsker. The algebraic dichotomy conjecture for infinite domain constraint satisfaction problems. In *Proceedings of the 31st Annual ACM/IEEE Symposium*

on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016, pages 615–622, 2016. doi:10.1145/2933575.2934544.

4 M. Bodirsky, B. Martin, and A. Mottet. Discrete temporal constraint satisfaction problems. *Journal of the ACM*, 65(2), 2018. preprint available at https://arxiv.org/abs/1503.08572. doi:10.1145/3154832.

5 Manuel Bodirsky, Víctor Dalmau, Barnaby Martin, Antoine Mottet, and Michael Pinsker. Distance constraint satisfaction problems. *Information and Computation*, 247:87–105, 2016.

6 Manuel Bodirsky, Peter Jonsson, and Timo von Oertzen. Essential convexity and complexity of semi-algebraic constraints. *Logical Methods in Computer Science*, 8(4), 2012. An extended abstract about a subset of the results has been published under the title *Semilinear Program Feasibility* at ICALP'10.

7 Manuel Bodirsky, Peter Jonsson, and Timo von Oertzen. Horn versus full first-order: Complexity dichotomies in algebraic constraint satisfaction. *J. Log. Comput.*, 22(3):643–660, 2012. doi:10.1093/logcom/exr011.

8 Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. *J. ACM*, 57(2), 2010. doi:10.1145/1667053.1667058.

9 Manuel Bodirsky and Marcello Mamino. Constraint Satisfaction Problems over Numeric Domains. In Andrei Krokhin and Stanislav Zivny, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 79–111. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017. doi:10.4230/DFU.Vol7.15301.79.

10 Manuel Bodirsky, Barnaby Martin, and Antoine Mottet. Constraint satisfaction problems over the integers with successor. In *Proceedings of ICALP'15*, 2015.

11 Manuel Bodirsky and Michael Pinsker. Schaefer's theorem for graphs. *Journal of the ACM*, 62(3):Article no. 19, 1–52, 2015. A conference version appeared in the Proceedings of STOC 2011, pages 655–664.

12 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *Proceedings of FOCS'17*, 2017. arXiv:1703.03021.

13 T.-W. J. Chou and G. E. Collins. Algorithms for the solution of systems of linear diophantine equations. *SIAM J. Computing*, 11:687–708, 1982.

14 T. Feder and M. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal on Computing*, 28:57–104, 1998.

15 Peter Jonsson and Tomas Lööw. Computational complexity of linear constraints over the integers. *Artificial Intelligence*, 195:44–62, 2013. An extended abstract appeared at IJCAI 2011.

16 Peter Jonsson and Johan Thapper. Constraint satisfaction and semilinear expansions of addition over the rationals and the reals. *J. Comput. Syst. Sci.*, 82(5):912–928, 2016. doi:10.1016/j.jcss.2016.03.002.

17 Ravindran Kannan and Achim Bachem. Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix. *SIAM J. Comput.*, 8(4):499–507, 1979.

18 David Marker. *Model Theory: An Introduction*. Springer, 2002.

19 Daniele Micciancio and Bogdan Warinschi. *A Linear Space Algorithm for Computing the Hermite Normal Form*, pages 231–236. Association for Computing Machinery (ACM), United States, 2001.

20 Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

21 M. Presburger. über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. *Comptes Rendus du I congres de Mathématiciens des Pays Slaves*, pages 92–101, 1929.

22 T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of STOC'78*, pages 216–226, 1978.

23 Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley - Interscience Series in Discrete Mathematics and Optimization, 1998.

24 Arne Storjohann. Computing hermite and smith normal forms of triangular integer matrices. *Linear Algebra and its Applications*, 282:25–45, 1998.

25 Dmitriy Zhuk. The Proof of CSP Dichotomy Conjecture. In *Proceedings of FOCS'17*, 2017. arXiv:1704.01914.

Give Me Some Slack: Efficient Network Measurements

Ran Ben Basat
Department of Computer Science, Technion
sran@cs.technion.ac.il

Gil Einziger
Nokia Bell Labs
gil.einziger@nokia.com

Roy Friedman
Department of Computer Science, Technion
roy@cs.technion.ac.il

Abstract

Many networking applications require timely access to recent network measurements, which can be captured using a sliding window model. Maintaining such measurements is a challenging task due to the fast line speed and scarcity of fast memory in routers. In this work, we study the impact of allowing *slack* in the window size on the asymptotic requirements of sliding window problems. That is, the algorithm can dynamically adjust the window size between W and $W(1+\tau)$ where τ is a small positive parameter. We demonstrate this model's attractiveness by showing that it enables efficient algorithms to problems such as Maximum and General-Summing that require $\Omega(W)$ bits even for constant factor approximations in the exact sliding window model. Additionally, for problems that admit sub-linear approximation algorithms such as Basic-Summing and Count-Distinct, the slack model enables a further asymptotic improvement.

The main focus of the paper is on the widely studied Basic-Summing problem of computing the sum of the last W integers from $\{0, 1 \ldots, R\}$ in a stream. While it is known that $\Omega(W \log R)$ bits are needed in the exact window model, we show that approximate windows allow an *exponential* space reduction for constant τ.

Specifically, for $\tau = \Theta(1)$, we present a space lower bound of $\Omega(\log(RW))$ bits. Additionally, we show an $\Omega(\log(W/\epsilon))$ lower bound for $RW\epsilon$ additive approximations and a $\Omega(\log(W/\epsilon) + \log\log R)$ bits lower bound for $(1+\epsilon)$ multiplicative approximations. Our work is the first to study this problem in the exact and additive approximation settings. For all settings, we provide memory optimal algorithms that operate in *worst case* constant time. This strictly improves on the work of [14] for $(1+\epsilon)$-multiplicative approximation that requires $O(\epsilon^{-1}\log(RW)\log\log(RW))$ space and performs updates in $O(\log(RW))$ worst case time. Finally, we show asymptotic improvements for the Count-Distinct, General-Summing and Maximum problems.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Streaming, Network Measurements, Statistics, Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.34

Related Version A full version of the paper is avalable at [5], `https://arxiv.org/abs/1703.01166`.

43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018).
Editors: Igor Potapov, Paul Spirakis, and James Worrell; Article No. 34; pp. 34:1–34:16

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

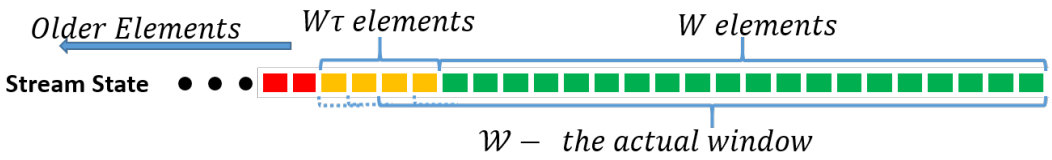


Figure 1 We need to answer each query with respect to a τ-slack window that must include the last W items, but may or may not consider a suffix of the previous $W\tau$ elements.

1 Introduction

Network algorithms in diverse areas such as traffic engineering, load balancing and quality of service [2, 9, 21, 24, 31] rely on timely link measurements. In such applications recent data is often more relevant than older data, motivating the notions of *aging* and *sliding window* [6, 11, 15, 25, 27]. For example, a sudden decrease in the average packet size on a link may indicate a SYN attack [26]. Additionally, a load balancer may benefit from knowing the current utilization of a link to avoid congestion [2].

While conceptually simple, conveying the necessary information to network algorithms is a difficult challenge due to current memory technology limitations. Specifically, DRAM memory is abundant but too slow to cope with the line rate while SRAM memory is fast enough but has a limited capacity [10, 13, 29]. Online decisions are therefore realized through space efficient data structures [7, 8, 16, 17, 4, 23, 28, 30] that store measurement statistics in a concise manner. For example, [16, 28] utilize probabilistic counters that only require $O(\log\log N)$ bits to approximately represent numbers up to N. Others conserve space using variable sized counter encoding [17, 23] and monitoring only the frequent elements [6].

Basic-Summing is one of the most basic textbook examples of such approximated sliding window stream processing problems [14]. In this problem, one is required to keep track of the sum of the last W elements, when all elements are non-negative integers in the range $\{0, 1, \ldots, R\}$. The work in [14] provides a $(1+\epsilon)$-multiplicative approximation of this problem using $O\left(\frac{1}{\epsilon}\cdot\left(\log^2 W + \log R\cdot(\log W + \log\log R)\right)\right)$ bits. The amortized time complexity is $O(\frac{\log R}{\log W})$ and the worst case is $O(\log W + \log R)$. In contrast, we previously showed an $RW\epsilon$-additive approximation with $\Theta\left(\frac{1}{\epsilon} + \log W\epsilon\right)$ bits [3].

Sliding window counters (approximated or accurate) require asymptotically more space than plain stream counters. Such window counters are prohibitively large for networking devices which already optimize the space consumption of plain counters.

This paper explores the concept of *slack*, or approximated sliding window, bridging this gap. Figure 1 illustrates a "window" in this model. Here, each query may select a *τ-slack window* whose size is between W (the green elements) and $W(1+\tau)$ (the green plus yellow elements). The goal is to compute the sum with respect to this chosen window.

Slack windows were also considered in previous works [14, 27] and we call the problem of maintaining the sum over a slack window Slack Summing. Datar et al. [14] showed that constant slack reduces the required memory from $O(\frac{1}{\epsilon}\cdot\left(\log^2 W + \log R\cdot(\log W + \log\log R)\right))$ to $O(\epsilon^{-1}\log(RW)\log\log(RW))$. For τ-slack windows they provide a $(1+\epsilon)$-multiplicative approximation using $O(\epsilon^{-1}\log(RW)(\log\log(RW) + \log\tau^{-1}))$ bits.

Table 1 Comparison of Basic-Summing algorithms. Our contributions are in bold. All algorithms process elements in constant time except for the rightmost column where both update in $O(\log(RW))$ time. We present matching lower bounds to all our algorithms.

	Exact Sum	Additive Error	Multiplicative Error	
$\tau = \Theta(1)$	$\mathbf{\Theta(\log(RW))}$	$\mathbf{\Theta(\log(W/\epsilon))}$	$\mathbf{\Theta(\log(W/\epsilon) + \log\log R)}$	$O(\epsilon^{-1}\log(RW)\log\log(RW))$ [14]
Exact Window	$\Theta(W \log R)$	$\Theta(\epsilon^{-1} + \log W)$ [3]	$O(\epsilon^{-1}\log^2(RW))$ [22]	$O(\epsilon^{-1}\log RW \log(W \log R))$ [14]

Our Contributions

This paper studies the space and time complexity reductions that can be attained by allowing *slack* – an error in the window size. Our results demonstrate exponentially smaller and asymptotically faster data structures compared to various problems over exact windows. We start with deriving lower bounds for three variants of the Basic-Summing problem – when computing an exact sum over a slack window, or when combined with an additive and a multiplicative error in the sum. We present algorithms that are based on dividing the stream into $W\tau$-sized blocks. Our algorithms sum the elements within each block and represent each block's sum in a cyclic array of size τ^{-1}. We use multiple compression techniques during different stages to drive down the space complexity. The resulting algorithms are space optimal, substantially simpler than previous work, and reduce update time to $O(1)$.

For exact Slack Summing, we present a lower bound of $\Omega(\tau^{-1}\log(RW\tau))$ bits. For $(1+\epsilon)$ multiplicative approximations we prove an $\Omega\big(\log(W/\epsilon) + \tau^{-1}(\log(\tau/\epsilon) + \log\log(RW))\big)$ bits bound when $\tau = \Omega\left(\frac{1}{\log RW}\right)$. We show that $\Omega(\tau^{-1}\log\lfloor 1+\tau/\epsilon\rfloor + \log(W/\epsilon))$ bits are required for $RW\epsilon$ additive approximations.

Next, we introduce algorithms for the Slack Summing problem, which asymptotically reduce the required memory compared to the sliding window model. For the exact and additive error versions of the problem, we provide memory optimal algorithms. In the multiplicative error setting, we provide an $O\big(\tau^{-1}\left(\log\epsilon^{-1} + \log\log(RW\tau)\right) + \log(RW)\big)$ space algorithm. This is asymptotically optimal when $\tau = \Omega(\log^{-1} W)$ and $R = \text{poly}(W)$. It also asymptotically improves [14] when $\tau^{-1} = o(\epsilon^{-1}\log(RW))$. We further provide an asymptotically optimal solution for constant τ, even when $R = W^{\omega(1)}$. All our algorithms are deterministic and operate in worst case constant time. In contrast, the algorithm of [14] works in $O(\log RW)$ worst case time.

To exemplify our results, consider monitoring the average bandwidth (in bytes per second) passed through a router in a 24 hours window, i.e., $W \triangleq 86400$ seconds. Assuming we use a 100GbE fiber transceiver, our stream values are bounded by $R \approx 2^{34}$ bytes. If we are willing to withstand an error of $\epsilon = 2^{-20}$ (i.e., about $16KBps$), the work of [3] provides an additive approximation over the sliding window and requires about 120KB. In contrast, using a 10 minutes slack ($\tau \triangleq \frac{1}{144}$), our algorithm for **exact** Slack Summing requires only 800 bytes, 99% less than approximate summing over exact sliding window. For the same slack size, the algorithm of [14] requires more space than our **exact** algorithm even for a large 3% error. Further, if we also allow the same additive error ($\epsilon = 2^{-20}$), we provide an algorithm that requires only 240 bytes - a reduction of more than 99.8% !

Table 1 compares our results for the important case of constant slack with [14]. As depicted, our *exact* algorithm is faster and more space efficient than the multiplicative approximation of [14]. Comparing our multiplicative approximation algorithm to that of [14], we present *exponential* space reductions in the dependencies on ϵ^{-1} and R, with an asymptotic reduction in W as well. We also improve the update time from $O(\log(RW))$ to $O(1)$.

Finally, we apply the slack window approach to multiple streaming problems, including Maximum, General-Summing, Count-Distinct and Standard-Deviation. We show that, while some of these problems cannot be approximated on an exact window in sub-linear space (e.g. maximum and general sum), we can easily do so for slack windows. In the count distinct problem, a constant slack yields an asymptotic space reduction over [11, 19].

2 Preliminaries

For $\ell \in \mathbb{N}$, we denote $[\ell] \triangleq \{0, 1, \ldots, \ell\}$. We consider a stream of data elements $x_1, x_2, \ldots, x_t$, where at each step a new element $x_i \in [R]$ is added to S. A *W-sized window* contains only the last W elements: $x_{t-W+1} \ldots x_t$. We say that $\mathcal{F}$ is a *τ-slack W-sized window* if there exists $c \in [W\tau - 1]$ such that $\mathcal{F} = x_{t-(W+c)+1} \ldots x_t$. For simplicity, we assume that τ^{-1} and $W\tau$ are integers. Unless explicitly specified, the base of all logs is 2.

Algorithms for the Slack Summing problem are required to support two operations:

1. Update(x_t) Process a new element $x_t \in [R]$.
2. Output () Return a pair $\langle \widehat{S}, c \rangle$ such that $c \in \mathbb{N}$ is the slack size and $\widehat{S}$ is an estimation of the last $W + c$ elements sum, i.e., $S \triangleq \sum_{k=t-(W+c)+1}^{t} x_k$.

We consider three types of algorithms for Slack Summing:

1. **Exact algorithms:** an algorithm $\mathbb{A}$ solves (W, τ)-Exact Summing if its Output returns $\langle \widehat{S}, c \rangle$ that satisfies $0 \leq c < W\tau$ and $\widehat{S} = S$.
2. **Additive algorithms:** we say that $\mathbb{A}$ solves (W, τ, ϵ)-Additive Summing if its Output function returns $\langle \widehat{S}, c \rangle$ that satisfies $0 \leq c < W\tau$ and $|S - \widehat{S}| < RW\epsilon$.
3. **Multiplicative algorithms:** $\mathbb{A}$ solves (W, τ, ϵ)-Multiplicative Summing if its Output returns $\langle \widehat{S}, c \rangle$ satisfying $0 \leq c < W\tau$ and $\frac{S}{1+\epsilon} < \widehat{S} \leq S$ if $S > 0$, and $\widehat{S} = 0$ otherwise.

3 Lower Bounds

In this section, we analyze the space required for solving the Slack Summing problems. Intuitively, our bounds are derived by constructing a set of inputs that any algorithm must distinguish to meet the required guarantees. There are two tricks that we frequently use in these lower bounds. The first is setting the input such that the slack consists only of zeros, and thus the algorithm must return the desired approximation of the remaining window. The next is using a "cycle argument" – consider two inputs x and $x \cdot y$ for $x, y \in \{0, 1, \ldots, R\}^*$. If both lead to the same memory configuration, so do such xy^k for any $k \in \mathbb{N}$. Thus, if there is a k such that no single answer approximates x and xy^k well, then x and xy had to lead to separate memory configurations in the first place.

3.1 (W, τ)-Exact Summing

We start by proving lower bounds on the memory required for exact Slack Summing.

▶ **Lemma 1.** *Any deterministic algorithm $\mathbb{A}$ that solves the (W, τ)-Exact Summing problem must use at least $\lceil \log(RW(W+1)/2 + 1) \rceil \geq \lfloor \log(RW^2) \rfloor$ bits.*

Proof. Consider the following language

$$L_{E_1} \triangleq \left\{ 0^{W\tau+i} \sigma R^{W-i-1} 0^j \mid i, j \in [W-1], i \geq j, \sigma \in ([R] \setminus \{0\}) \right\} \cup \{0^{W+W\tau}\}.$$

That is, L_{E_1} contains a word with $W + W\tau$ consecutive zeros and the rest of the words in L_{E_1} are composed of these components in this order:

- $W\tau + i$ zeros for some $i \in [W-1]$.
- a non zero symbol σ.
- $W - i - 1$ repetitions of the maximal symbol (R).
- j zeros for some $j \in [i]$.

Our lower bound stems from the observation that every word in L_{E_1} must lead to a different state. The language size is: $|L_{E_1}| = 1 + \sum_{i=0}^{W-1} R(i+1) = 1 + RW(W+1)/2$. Therefore, the number of required bits is at least: $\lceil \log |L_{E_1}| \rceil > \left(\log(RW^2) - 1\right)$. Further, this number is an integer and therefore at least $\left\lfloor \log(RW^2) \right\rfloor$ bits are required.

First, notice that the word composed of $W + W\tau$ zeros requires a unique configuration as $\mathbb{A}$ must return 0 after processing that word. In contrast, it must not return 0 after processing any other word as there is at least a single R within the last W elements.

Let $w_1, w_2 \in L_{E_1}$ be two different words that are not all-zeros. We need to show that w_1 and w_2 require different memory configuration.

By definition of L_{E_1}, $w_1 = 0^{W\tau+i_1}\sigma_1 R^{W-i_1-1}0^{j_1}$ and $w_2 = 0^{W\tau+i_2}\sigma_2 R^{W-i_2-1}0^{j_2}$. Observe that the last W elements of w_1, w_2 are $0^{i_1-j_1}\sigma_1 R^{W-i_1-1}0^{j_1}$ and $0^{i_2-j_2}\sigma_1 R^{W-i_2-1}0^{j_2}$ respectively and that both are preceded with at least $W\tau$ zeros. If $i_1 \neq i_2$ or $\sigma_1 \neq \sigma_2$, then $\sigma_1 + R \cdot (W - i_1 - 1) \neq \sigma_2 + R \cdot (W - i_2 - 1)$ and thus $\mathbb{A}$ cannot return the same count for both, regardless of the slack, as it is all zeros ib both w_1 and w_2.

Next, assume that $i_1 = i_2$, $\sigma_1 = \sigma_2$ and that without loss of generality $j_1 < j_2$. This means that both w_1 and w_2 have the same count.

Since $j_1 < j_2$, w_1 is a strict prefix of w_2, i.e., $w_2 = w_1 \cdot 0^{j_2-j_1}$. Assume by contradiction that after processing w_1, w_2 $\mathbb{A}$ reaches the same memory configuration. Since $\mathbb{A}$ is deterministic, this means that it must reach the same configuration after seeing $w_1 \cdot 0^{z(j_2-j_1)}$ for any integer z. By choosing $z = W(1+\tau)$, we get that the algorithm reaches this configuration once again while the entire window consists of zeros. This is a contradiction since $\sigma_1, \sigma_2 \neq 0$, and the algorithm cannot answer both w_1 and $w_1 \cdot 0^{z(j_2-j_1)}$ correctly. ◀

We now use Lemma 1 to show the following lower bound on (W,τ)-Exact Summing algorithms:

▶ **Theorem 2.** *Any deterministic algorithm $\mathbb{A}$ that solves the (W,τ)-Exact Summing problem must use at least* $\max\left\{\left\lfloor \log\left(RW^2\right)\right\rfloor, \left\lceil \left\lceil \tau^{-1}/2 \right\rceil \log\left(RW\tau+1\right)\right\rceil\right\}$ *bits.*

Proof. Lemma 1 shows a $\left\lfloor \log\left(RW^2\right)\right\rfloor$ bound. We proceed with showing a lower bound $\left\lceil \left\lceil \tau^{-1}/2 \right\rceil \log\left(RW\tau+1\right)\right\rceil$ bits. Consider the following languages:

$$L_{E_2} \triangleq \left\{0^{W\tau+i}\sigma R^{W\tau-i-1} \mid i \in [W\tau-1], \sigma \in [R]\right\},$$
$$\overline{L_{E_2}} \triangleq \left\{w_1 \cdot w_2 \cdots w_{\lceil \tau^{-1}/2 \rceil} \mid \forall i : w_i \in L_{E_2}\right\}.$$

Notice that $|\overline{L_{E_2}}| = (RW\tau+1)^{\lceil \tau^{-1}/2 \rceil}$ since each of the words in L_{E_2} has a distinct sum of literals, and each number in $\{0, 1, \ldots, RW\tau\}$ is the sum of a word. We show that each input in $\overline{L_{E_2}}$ must be mapped into a distinct memory configuration. Let $S_1 \triangleq w_{1,1} \cdot w_{2,1} \cdots w_{\lceil \tau^{-1}/2 \rceil,1}$, $S_2 \triangleq w_{1,2} \cdot w_{2,2} \cdots w_{\lceil \tau^{-1}/2 \rceil,2}$ be two distinct inputs in $\overline{L_{E_2}}$ such that $\forall i : w_{i,1}, w_{i,2} \in L_{E_2}$. Denote $\chi \triangleq \max\left\{i \in \left[\left\lceil \tau^{-1}/2 \right\rceil\right] \mid w_{i,1} \neq w_{i,2}\right\}$ – the last place in which S_1 differs from S_2; also, denote $w_{\chi,1} \triangleq 0^{W\tau}a, w_{\chi,2} \triangleq 0^{W\tau}b$. Consider the sequences $S_1^* = S_1 \cdot 0^{2W\tau(\chi-1/2)}$ and $S_2^* = S_2 \cdot 0^{2W\tau(\chi-1/2)}$. Notice that the last W elements windows for S_1^*, S_2^* are $a \cdot w_{\chi+1,1} \cdots w_{\lceil \tau^{-1}/2 \rceil,1} \cdot 0^{2W\tau(\chi-1/2)}$ and $b \cdot w_{\chi+1,2} \cdots w_{\lceil \tau^{-1}/2 \rceil,2} \cdot 0^{2W\tau(\chi-1/2)}$ respectively, and that the preceding $W\tau$ elements of both are all zeros. An illustration of the setting appears in Figure 2. By our choice of χ, we have that the sum of the last W elements of S_1^*

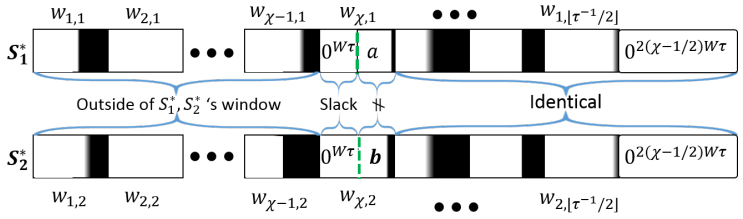


Figure 2 An illustration of the $\lceil\tau^{-1}/2\rceil\log(RW\tau+1)$ lower bound setting. If we assume that after seeing $w_{1,1}\cdot w_{2,1}\cdots w_{\lceil\tau^{-1}/2\rceil,1}$ we reach the same configuration as after processing $w_{1,2}\cdot w_{2,2}\cdots w_{\lceil\tau^{-1}/2\rceil,2}$, then we provide a wrong answer for at least one of S_1^*, S_2^*.

and S_2^* is different, and since the slack is all zeros, no answer is correct on both. Finally, note that this implies that S_1, S_2 had to reach different configurations, as otherwise $\mathbb{A}$ would reach the same configuration after processing the additional $2W\tau(\chi-1/2)$ zeros. ◀

3.2 (W,τ,ϵ)-Additive Summing

Next, Theorem 3 shows a lower bound for additive approximations of Slack Summing.

▶ **Theorem 3.** *For $\epsilon<1/4$, any deterministic algorithm $\mathbb{A}$ that solves the (W,τ,ϵ)-*Additive Summing *problem requires* $\max\left\{\log(W/\epsilon)-O(1),\left\lceil\lceil\tau^{-1}/2\rceil\log\lfloor\tau/2\epsilon+1\rfloor\right\rceil\right\}$ *bits.*

Before we prove Thorem 3, we start with a simpler lower bound.

▶ **Lemma 4.** *Let $\epsilon<1/4$. Any deterministic algorithm that solves the (W,τ,ϵ)-*Additive Summing *problem must use at least* $\log(W/\epsilon)-O(1)$ *bits.*

Proof. Denote by $rep(x)\triangleq(x \mod R)\cdot R^{\lfloor x/R\rfloor}$ a sequence in $\{\sigma R^*\mid\sigma\in[R]\}$ whose sum is x. Next, consider the following languages:

$$L_{A_1}\triangleq\{rep(k\cdot 2RW\epsilon)\mid k\in[\lfloor 1/4\epsilon\rfloor]\setminus\{0\}\};\qquad \overline{L_{A_1}}\triangleq 0^{W+W\tau}\cdot L_{A_1}\cdot\{0^q\mid q\in[\lfloor W/2\rfloor]\}.$$

First, notice that $|L_{A_1}|=\lfloor 1/4\epsilon\rfloor$ and that all words in L_{A_1} have length of at most $W/2$. This means that $|\overline{L_{A_1}}|=\lfloor 1/4\epsilon\rfloor\lfloor W/2+1\rfloor>\lfloor W/8\epsilon\rfloor$.

We now show that every word in $\overline{L_{A_1}}$ must have a dedicated memory configuration, thereby implying a $\lceil\log\lfloor W/8\epsilon\rfloor\rceil$ bits bound. Let $w_1=0^{W+W\tau}\cdot x_1\cdot 0^{q_1}$ and $w_2=0^{W+W\tau}\cdot x_2\cdot 0^{q_2}$ be two distinct words in $\overline{L_{A_1}}$ such that $x_1,x_2\in L_{A_1}$ and $q_1,q_2\in\lfloor W/2\rfloor$. If $x_1\neq x_2$, then their most recent W elements differ by more than $2RW\epsilon$ and there is no output that is correct for both. Note that the slack of both w_1 and w_2 is all zeros. Hence, w_1 and w_2 require different memory configurations.

Assume that $x_1=x_2$ and that by contradiction both w_1 and w_2 reached the same memory configuration. Since $w_1\neq w_2$ and $x_1=x_2$, then $q_1\neq q_2$ and without loss of generality $q_1<q_2$. This implies that w_1 is a prefix of w_2 so that $w_2=w_1\cdot 0^{q_2-q_1}$. Thus, $\mathbb{A}$ enters the shared configuration after reading w_1 and revisits it after reading $0^{q_2-q_1}$. $\mathbb{A}$ is a deterministic algorithm and therefore it reaches the same configuration also for the following word: $w_1\cdot 0^{(W+W\tau)(q_2-q_1)}$. In that word, the last $W+W\tau$ elements are all zeros while the sum of the last W elements in w_1 is at least $2RW\epsilon$. Hence, there is no return value that is correct for both w_1 and $w_1\cdot 0^{(W+W\tau)(q_2-q_1)}$. ◀

We are now ready to prove Theorem 3. The theorem says that for $\epsilon<1/4$, any deterministic algorithm $\mathbb{A}$ that solves the (W,τ,ϵ)-Additive Summing problem requires $\max\left\{\log(W/\epsilon)-O(1),\left\lceil\lceil\tau^{-1}/2\rceil\log\lfloor\tau/2\epsilon+1\rfloor\right\rceil\right\}$ bits.

Proof. Lemma 4 shows that $\mathbb{A}$ must use at least $\log(W/\epsilon) - O(1)$ bits. Given $x \in [RW\tau]$, we denote by $rep(x) \triangleq 0^{2W\tau - \lfloor x/R \rfloor - 1} \cdot (x \mod R) \cdot R^{\lfloor x/R \rfloor}$ a sequence of the following form: $\left\{0^{W\tau+i}\sigma R^{W\tau-i-1} \mid i \in [W\tau - 1], \sigma \in [R]\right\}$ whose sum is x. We consider the following languages:

$$L_{A_2} \triangleq \{rep(k \cdot 2RW\epsilon) \mid k \in [\lfloor \tau/2\epsilon \rfloor]\}; \qquad \overline{L_{A_2}} \triangleq \left\{w_1 \cdot w_2 \cdots w_{\lceil \tau^{-1}/2 \rceil} \mid \forall i : w_i \in L_{A_2}\right\}.$$

Our goal is to show that no two words in $\overline{L_{A_2}}$ have the same memory configuration. Let $S_1, S_2 \in \overline{L_{A_2}}$ so that $S_1 \neq S_2$. Denote $S_1 \triangleq w_{1,1} \cdot w_{2,1} \cdots w_{\lfloor \tau^{-1}/2 \rfloor,1}$ and $S_2 \triangleq w_{1,2} \cdot w_{2,2} \cdots w_{\lfloor \tau^{-1}/2 \rfloor,2}$, while $\forall i : w_{i,1}, w_{i,2} \in L_{A_2}$. We denote $\chi \triangleq \max\{i \mid w_{i,1} \neq w_{i,2}\}$ – the last place in which S_1 differs from S_2.

Next, consider the following sequences: $S_1^* = S_1 \cdot 0^{2W\tau(\chi-1/2)}$ and $S_2^* = S_2 \cdot 0^{2W\tau(\chi-1/2)}$. The last $W + W\tau$ elements in S_1^* are $w_{\chi,1} \cdots w_{\lceil \tau^{-1}/2 \rceil,1} \cdot 0^{2W(\chi-1/2)\tau}$ and in S_2^* $w_{\chi,2} \cdots w_{\lceil \tau^{-1}/2 \rceil,2} \cdot 0^{2W(\chi-1/2)\tau}$. Additionally, the $W\tau$ elements slack in both S_1^* and S_2^* are all zeros. Now, since the sum of $w_{\chi,1}$ and $w_{\chi,2}$ must differ by at least $2RW\epsilon$, no number can approximate both with less than $RW\epsilon$ error. ◀

3.3 (W, τ, ϵ)-Multiplicative Summing

In this section, we show lower bounds for multiplicative approximations of Slack Summing. We start with Lemma 5, whose proof appears in the full version of this paper [5].

▶ **Lemma 5.** *For $\epsilon < 1/4$, any deterministic algorithm $\mathbb{A}$ for the (W, τ, ϵ)-Multiplicative Summing problem requires at least $\log(W/\epsilon) + \log\log(RW\epsilon) - O(1)$ memory bits.*

To extend our multiplicative lower bound, we use the following fact:

▶ **Fact 1.** *For any $x \neq 1, y \in \mathbb{R}$, the sequence $\{c_i\}_{i=1}^{\infty}$, defined as* $c_n \triangleq \begin{cases} 1 & \text{n} = 1 \\ x \cdot c_{n-1} + y & \text{otherwise} \end{cases}$ *can be represented using a closed form as $c_n = x^{n-1} + y \cdot \frac{x^n - 1}{x - 1}$.*

Next, let $k \in \mathbb{N}$ and $\psi, \epsilon \in \mathbb{R}$, such that $\psi \geq 2$, $\epsilon > 0$, $k \geq 1$; consider the integer sequence

$$a_{n,k} \triangleq \begin{cases} 1 & \text{n} = 1 \\ \left\lceil (1+\epsilon)\left(a_{n-1,k} + \sum_{i=1}^{k-1} \psi^i\right)\right\rceil & \text{otherwise.} \end{cases}$$

Using the fact above, we show the following lemma:

▶ **Lemma 6.** *For every integer $n \geq 1$ we have $a_{n,k} \leq 4\epsilon^{-1}(1+\epsilon)^{n+1}\psi^{k-1}$.*

Proof. To apply Fact 1, we define an upper bounding sequence $\{b_{i,k}\}_{i=1}^{\infty}$ as follows:

$$b_{n,k} \triangleq \begin{cases} 1 & \text{n} = 1 \\ (1+\epsilon)\left(b_{n-1,k} + \sum_{i=1}^{k-1} \psi^i\right) + 1 & \text{otherwise.} \end{cases}$$

Thus, we can rewrite the n'th element of the sequence as:

$$b_{n,k} = (1+\epsilon)^{n-1} + \tfrac{(1+\epsilon)^n - 1}{(1+\epsilon) - 1}\left((1+\epsilon)\textstyle\sum_{i=1}^{k-1} \psi^i + 1\right).$$

We can now use this representation to derive an upper bound of $b_{n,k}$:

$$\begin{aligned} b_{n,k} &= (1+\epsilon)^{n-1} + \left((1+\epsilon)\textstyle\sum_{i=1}^{k-1} \psi^i + 1\right)\tfrac{(1+\epsilon)^n - 1}{(1+\epsilon) - 1} \\ &\leq (1+\epsilon)^{n-1} + \left((1+\epsilon)2\psi^{k-1}\right)\frac{(1+\epsilon)^n - 1}{\epsilon} \leq 4\epsilon^{-1}(1+\epsilon)^{n+1}\psi^{k-1}. \end{aligned}$$

Finally, since $a_{n,k} \leq b_{n,k}$ for any n, k, we conclude that $a_{n,k} \leq 4\epsilon^{-1}(1+\epsilon)^{n+1}\psi^{k-1}$. ◀

We now define the integer set I_k as $I_k \triangleq \left\{a_{n,k} \mid a_{n,k} \leq \psi^k\right\}$, and proceed to bound $|I_k|$.

▶ **Lemma 7.** *For any $k \geq 1$ we have $|I_k| \geq \epsilon^{-1} \ln(\psi\epsilon/4) - 1$.*

Proof. Clearly, the cardinality of I_k is the largest n for which $a_{n,k} \leq \psi^k$. According to Lemma 6, we have that $a_{n,k} \leq 4\epsilon^{-1}(1+\epsilon)^{n+1}\psi^{k-1}$, and thus:

$$\begin{aligned}|I_k| &= \arg\max\left\{n \mid 4\epsilon^{-1}(1+\epsilon)^{n+1}\psi^{k-1} \leq \psi^k\right\} \\ &\geq \log_{1+\epsilon}(\psi\epsilon/4) - 1 = \frac{\ln(\psi\epsilon/4)}{\ln(1+\epsilon)} - 1 \geq \epsilon^{-1}\ln(\psi\epsilon/4) - 1.\end{aligned}$$ ◀

We proceed with a stronger lower bound for non-constant τ values.

▶ **Lemma 8.** *For $\frac{1}{2\log(RW)-8} \leq \tau \leq 1$, any deterministic algorithm $\mathbb{A}$ that solves (W,τ,ϵ)-*Multiplicative Summing *requires at least $\Omega\left(\tau^{-1}\left(\log(\tau/\epsilon) + \log\log(RW)\right)\right)$ bits.*

Proof. We use $rep(x) \triangleq (x \mod R) \cdot R^{\lfloor x/R \rfloor}$ to denote a sequence in $\{\sigma R^* \mid \sigma \in [R]\}$ that has a sum of x. For an integer set I_k, we denote $rep(I_k) \triangleq \{rep(x) \mid x \in I_k\}$. We now choose the value of ψ to be $\psi \triangleq \sqrt[\lceil \tau^{-1}/2 \rceil]{RW/8}$; notice that $\psi \geq 2$ as required. Next, consider:

$$\begin{aligned}\overline{L_{M,2}} &\triangleq 0^W \cdot 0^{W\tau} \cdot rep(I_{\lceil \tau^{-1}/2 \rceil}) \cdot 0^{W\tau} \cdot rep(I_{\lceil \tau^{-1}/2 \rceil - 1}) \cdots 0^{W\tau} \cdot rep(I_1) \\ &= \left\{0^W \cdot w_1 \cdot w_2 \cdots w_{\lceil \tau^{-1}/2 \rceil} \mid \forall i : w_i \in \left\{0^{W\tau} \cdot rep(x) \mid x \in I_{\lceil \tau^{-1}/2 \rceil + 1 - i}\right\}\right\}.\end{aligned}$$

That is, every word in the $\overline{L_{M,2}}$ language consists of a concatenation of words $w_1, \ldots, w_{\lceil \tau^{-1}/2 \rceil}$, such that every w_i starts with $W\tau$ zeros followed by a string representing an integer in $I_{\lceil \tau^{-1}/2 \rceil + 1 - i}$, which is defined above. According to Lemma 7 we have that

$$\begin{aligned}\log(|\overline{L_{M,2}}|) &\geq \log\left(\left(\epsilon^{-1}\ln(\psi\epsilon/4) - 1\right)^{\lceil \tau^{-1}/2 \rceil}\right) \\ &= \lceil \tau^{-1}/2 \rceil \left(\log\epsilon^{-1} + \log\log(\psi\epsilon) - O(1)\right) \\ &= \Omega\left(\tau^{-1}\left(\log\epsilon^{-1} + \log\log\left(\sqrt[\lceil \tau^{-1}/2 \rceil]{RW/8} \cdot \epsilon\right)\right)\right) \\ &= \Omega\left(\tau^{-1}\left(\log\epsilon^{-1} + \log\left(\frac{\log(RW/8)}{\lceil \tau^{-1}/2 \rceil} + \log\epsilon\right)\right)\right) \\ &= \Omega\left(\tau^{-1}\left(\log(\tau/\epsilon) + \log\log(RW)\right)\right).\end{aligned}$$

Next, we show that every two words in $\overline{L_{M,2}}$ must reach different memory configurations, thereby implying a $\Omega\left(\log\left(|\overline{L_{M,2}}|\right)\right)$ bits lower bound. Let $S_1 \neq S_2 \in \overline{L_{M,2}}$ such that $S_1 = 0^W \cdot w_{1,1} \cdots w_{\lceil \tau^{-1}/2 \rceil,1}$, $S_2 = 0^W \cdot w_{1,2} \cdots w_{\lceil \tau^{-1}/2 \rceil,2}$, and $\forall i \in \left\{1, \ldots, \lceil \tau^{-1}/2 \rceil\right\} j \in \{1,2\} : w_{i,j} \in \left\{0^{W\tau} \cdot rep(x) \mid x \in I_{\lceil \tau^{-1}/2 \rceil + 1 - i}\right\}$. We next assume by contradiction that S_1 and S_2 leads $\mathbb{A}$ to the same memory configuration. Let $\chi \in \left\{1, \ldots, \lceil \tau^{-1}/2 \rceil\right\}$ such that $w_{\chi,1} \neq w_{\chi,2}$. Since $\mathbb{A}$ reaches an identical configuration after reading S_1, S_2, and as it is deterministic, $\mathbb{A}$ must reach the same configuration when processing $S_1 \cdot 0^{2W\tau(\chi - 1/2)}$ and $S_2 \cdot 0^{2W\tau(\chi - 1/2)}$. Next, observe that for every $k \in \{1, \ldots, \lceil \tau^{-1}/2 \rceil\}$, the representation length of any of its words is bounded by $\lceil \psi^k/R \rceil$. Thus, the length of a word in

$$\left\{w_1 \cdot w_2 \cdots w_{\lceil \tau^{-1}/2 \rceil} \mid \forall i : w_i \in \left\{0^{W\tau} \cdot rep(x) \mid x \in I_{\lceil \tau^{-1}/2 \rceil + 1 - i}\right\}\right\} \text{ is at most}$$

$$\sum_{k=1}^{\lceil \tau^{-1}/2 \rceil} \lceil W\tau + \psi^k/R \rceil \leq \lceil \tau^{-1}/2 \rceil (W\tau + 1) + 2\psi^{\lceil \tau^{-1}/2 \rceil}/R$$
$$= \lceil \tau^{-1}/2 \rceil (W\tau + 1) + 2W/8 \leq 3W/4 + \lceil \tau^{-1}/2 \rceil + W\tau \leq W + W\tau.$$

Now, since every word $w_{i,j}$ starts with a sequence of $W\tau$ zeros, the slack size chosen by the algorithm is irrelevant and the sums the algorithm must estimate are $\sum_{i=\chi}^{\lceil \tau^{-1}/2 \rceil} s(w_{i,1})$ and $\sum_{i=\chi}^{\lceil \tau^{-1}/2 \rceil} s(w_{i,2})$, where $s(w_{i,j})$ is simply the sum of the symbols in $w_{i,j}$. Note that $s(w_{\chi,1})$ and $s(w_{\chi,2})$ are integers in $I_{\lceil \tau^{-1}/2 \rceil + 1 - \chi}$. We assume without loss of generality that $s(w_{\chi,1}) < s(w_{\chi,2})$ (i.e., $s(w_{\chi,1}) < s(w_{\chi,2}) \in I_{\lceil \tau^{-1}/2 \rceil + 1 - \chi}$). Finally, it follows that

$$\sum_{i=\chi}^{\lceil \tau^{-1}/2 \rceil} s(w_{i,1}) \leq s(w_{\chi,1}) + \sum_{i=\chi+1}^{\lceil \tau^{-1}/2 \rceil} \max(I_{\lceil \tau^{-1}/2 \rceil + 1 - i}) \leq s(w_{\chi,1}) + \sum_{k=1}^{\chi-1} \psi^k \leq \frac{s(w_{\chi,2})}{1+\epsilon},$$

where the last inequality follows from the definition of $I_{\lceil \tau^{-1}/2 \rceil + 1 - \chi}$. Thus, no $\widehat{S}$ value is correct for both $S_1 \cdot 0^{2W\tau(\chi - 1/2)}$ and $S_2 \cdot 0^{2W\tau(\chi - 1/2)}$. ◀

Finally, we combine Lemma 5 and Lemma 8 to obtain the following lower bound:

▶ **Theorem 9.** *For* $\epsilon < 1/4$, $\frac{1}{2\log(RW)-8} \leq \tau \leq 1$, *any deterministic algorithm for the* (W, τ, ϵ)-Multiplicative Summing *problem requires at least*

$$\Omega\big(\log(W/\epsilon) + \tau^{-1}(\log(\tau/\epsilon) + \log\log(RW))\big) \text{ bits.}$$

4 Upper Bounds

In this section, we introduce solutions for the Slack Summing problems. In general, all our algorithms have a structure that consists of a subset of the following, where "rounding" has a different meaning for the exact, additive and multiplicative variants:

- Round the arriving item.
- Add the item into a counter y and round the counter.
- If a $W\tau$-sized *block* ends, store it as a compressed representation of y. Sometimes we propagate the compression error to the following block; otherwise, we zero y.
- Use the block values and y to construct an estimation for the sum.

A key idea in our additive and multiplicative algorithms is to introduce rounding errors but maintain the accountability trail so that they do not snowball and exceed the desired guarantees. In the additive algorithm, our *double rounding* technique asymptotically improves over running $1/\tau$ separate plain stream (insertion only) algorithm instances.

4.1 (W, τ)-Exact Summing

We divide the stream into $W\tau$-sized blocks and sum the number of arriving elements in each block with a $\lceil \log(RW\tau + 1) \rceil$ bits counter. We maintain the sum of the current block in a variable called y, c maintains the number of elements within the current block, and i is the current block number. The variable b is a cyclic buffer of τ^{-1} blocks. Every $W\tau$ steps, we assign the value of y to the oldest block (b_i) and increment i. Intuitively, we "forget" b_i when its block is no longer part of the window. To satisfy queries in constant time, we also maintain the sum of all active counters in a $\lceil \log(RW(1+\tau)+1) \rceil$-bits variable named B. Algorithm 1 provides pseudocode for the described algorithm. We now analyze the memory consumption of Algorithm 1.

Algorithm 1 (W,τ)-Exact Summing Algorithm.

Initialization: $y=0, b=\bar{0}, B=0, i=0, c=0$.
function Update(x)
  $y \leftarrow y+x$
  $c \leftarrow (c+1) \mod W\tau$
  if $c=0$ **then** ▷ End of block
    $B \leftarrow B-b_i+y$
    $b_i \leftarrow y$
    $y \leftarrow 0$
    $i \leftarrow (i+1) \mod \tau^{-1}$
function Output
  return $\langle B+y, c\rangle$

▶ **Theorem 10.** *Algorithm 1 uses* $(\tau^{-1}+1)\lceil\log(RW\tau+1)\rceil+\log\left(RW^2\right)+O(1)$ *bits.*

Proof. y takes $\lceil\log(RW\tau+1)\rceil$ bits; B requires $\lceil\log(RW+1)\rceil$; i adds $\left\lceil\log\tau^{-1}\right\rceil$ bits, while c needs $\lceil\log W\tau\rceil$ bits. Finally, b is a τ^{-1}-sized array of counters, each allocated with $\lceil\log(RW\tau+1)\rceil$ bits. Overall, it uses $(\tau^{-1}+1)\lceil\log(RW\tau+1)\rceil+\log\left(RW^2\right)+4$ bits. ◀

We conclude that Algorithm 1 is asymptotically optimal.

▶ **Theorem 11.** *Let* $\mathcal{B} \triangleq \max\left\{\left\lfloor\log\left(RW^2\right)\right\rfloor, \left\lceil\left\lceil\tau^{-1}/2\right\rceil\log(RW\tau+1)\right\rceil\right\}$ *be the* (W,τ)-Exact Summing *lower bound of Theorem 2. Algorithm 1 uses at most* $\mathcal{B}(4+o(1))$ *memory bits.*

Theorem 11 shows that Algorithm 1 is only x4 larger than the lower bound. In the full version of this paper [5], we show that in some cases we can get considerably closer to the lower bound.

Finally, in the full version of this paper [5] we show that Algorithm 1 is correct.

4.2 (W,τ,ϵ)-Additive Summing

We now show that additional memory savings can be obtained by combining slackness with an additive error. First, we consider the case where $\tau \le 2\epsilon$. In [3], we proposed an algorithm that sums over (exact) W elements window using the optimal $\Theta(\epsilon^{-1}+\log W)$ bits, with an additive error of $RW\epsilon$. Next, notice that if an algorithm solves (W,τ,ϵ)-Additive Summing, it also solves $(W,\tau,\tau/2)$-Additive Summing; hence, we can apply Theorem 3 to conclude that it requires $\Omega(\tau^{-1}+\log W)=\Omega(\epsilon^{-1}+\log W)$. Thus, we can run the algorithm from [3] and remain asymptotically memory optimal with no slack at all!

Henceforth, we assume that $\tau > 2\epsilon$; we present an algorithm for the problem using a 2-*stage rounding* technique. When a new item arrives, we scale it by R and then round the results to $O(\log\epsilon^{-1})$ bits. As in Section 4.1, we break the stream into non-overlapping blocks of size $W\tau$ and compute the sum of each block separately. However, we now sum the rounded values rather than the exact input, with a $O(\log\frac{W\tau}{\epsilon})$-bits counter denoted y. Once the block is completed, we *round its sum* such that it is represented with $O(\log\frac{\tau}{\epsilon})$ bits. Note that this second rounding is done for the entire block's sum while we still have the "exact" sum of rounded fractions. Thus, we *propagate* the second rounding error to the following block. An illustration of our algorithm appears in Figure 3. Here, $\text{Round}_v(z)$ refers to rounding a fractional number $z \in [0,1]$ into the closest number $\widetilde{z}$ such that $2^v \cdot \widetilde{z} \in \mathbb{N}$. Algorithm 2 provides pseudo code for the algorithm, which uses the following variables:

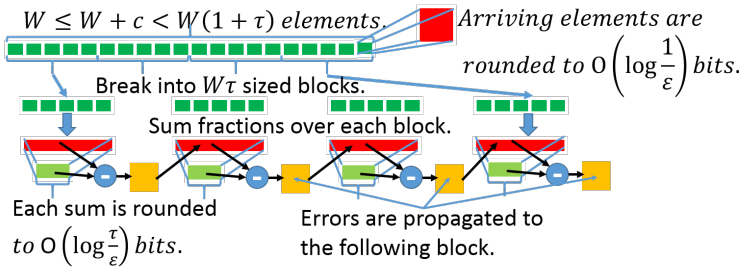


Figure 3 An illustration of our 2-stage rounding technique. Arriving elements are rounded to $\left(\left\lceil \log \epsilon^{-1} \right\rceil + 1\right)$ bits. We then sum the rounded fractions of each block and round the resulting sum into $\left\lceil \log \frac{\tau}{\epsilon} \right\rceil$ bits. The second rounding error is propagated to the next block.

Algorithm 2 (W, τ, ϵ)-ADDITIVE SUMMING Algorithm.

Initialization: $y = 0, b = 0, B = 0, i = 0, c = 0$.
1: **function** UPDATE(x)
2: $x' \leftarrow \text{Round}_{\upsilon_1}\left(\frac{x}{R}\right)$ ▷ Round $\left(\frac{x}{R}\right)$ such that $x' \cdot 2^{\upsilon_1} \in \mathbb{N}$
3: $y \leftarrow y + x'$
4: $c \leftarrow (c+1) \mod W\tau$
5: **if** $c = 0$ **then** ▷ End of block
6: $B \leftarrow B - b_i$
7: $b_i \leftarrow \text{Round}_{\upsilon_2}(\frac{y}{W\tau})$ ▷ Replace the value for the block that has left the window.
8: $B \leftarrow B + b_i$
9: $y \leftarrow y - W\tau \cdot b_i$
10: $i \leftarrow (i+1) \mod \tau^{-1}$
11: **function** OUTPUT
12: **return** $\langle R \cdot (W\tau \cdot B + y), c \rangle$

1. y - a fixed point variable that uses $\lceil \log W\tau \rceil + 1$ bits to store its integral part and additional $\upsilon_1 \triangleq \left\lceil \log \epsilon^{-1} \right\rceil + 1$ bits for storing the fractional part.
2. b - a cyclic array that contains τ^{-1} elements, each of which takes $\upsilon_2 \triangleq \left\lceil \log \frac{\tau}{\epsilon} \right\rceil$ bits.
3. B - keeps the sum of elements in b and is represented using $\log\left(\tau^{-1}\left\lceil \log \frac{\tau}{\epsilon} \right\rceil + 1\right)$ bits.
4. i - the index variable used for tracking the oldest block in b.
5. c - a variable that keeps the offset within the $W\tau$ sized block.

We now analyze the memory consumption of Algorithm 2.

▶ **Theorem 12.** *Algorithm 2 uses* $\tau^{-1} \log\left(\frac{\tau}{\epsilon}\right)(1 + o(1)) + 2\log(W/\epsilon)$ *bits.*

Proof. y requires $\log\left(\frac{W\tau}{\epsilon}\right) + O(1)$ bits; b requires another $\tau^{-1}\left\lceil \log\left(\frac{\tau}{\epsilon}\right)\right\rceil$; B takes additional $\log\left(\tau^{-1}\left\lceil \log \frac{\tau}{\epsilon} \right\rceil + 1\right)$ bits; i adds $\left\lceil \log \tau^{-1} \right\rceil$ bits, while and c is represented with $\lceil \log W\tau \rceil$ bits. Overall, the space requirement is $\tau^{-1}\left\lceil \log\left(\frac{\tau}{\epsilon}\right)\right\rceil(1 + o(1)) + 2\log(W/\epsilon)$ bits. ◀

▶ **Corollary 13.** *Let* $\mathcal{B} \triangleq \max\left\{\log(W/\epsilon) - O(1), \left\lceil \lceil \tau^{-1}/2 \rceil \log \lfloor \tau/2\epsilon + 1 \rfloor \right\rceil\right\}$ *be the* (W, τ, ϵ)-ADDITIVE SUMMING *space lower bound of Theorem 3, then Algorithm 2 uses* $\mathcal{B} \cdot (4 + o(1))$ *bits.*

Finally, Theorem 14 shows that Algorithm 2 is correct. The proof is deferred to the full version of this paper [5].

▶ **Theorem 14.** *Algorithm 2 solves the* (W, τ, ϵ)-ADDITIVE SUMMING *problem.*

Algorithm 3 (W, τ, ϵ)-Multiplicative Summing Algorithm.

Initialization: $y = 0, b = \bar{0}, B = 0, i = 0, c = 0$.
function Update(x)
  $y \leftarrow y + x$
  $c \leftarrow (c + 1) \mod W\tau$
  if $c = 0$ **then** ▷ End of block
    $\rho \leftarrow \left\lfloor \log_{(1+\epsilon/2)} y \right\rfloor$ ▷ If $y = 0$ we use $\rho = -\infty$ and $(1+\epsilon/2)^{\rho} = 0$
    $B \leftarrow B - \left((1+\epsilon/2)^{b_i}\right)_{\downarrow} + \left((1+\epsilon/2)^{\rho}\right)_{\downarrow}$
    $b_i \leftarrow \rho$
    $y \leftarrow 0$
    $i \leftarrow (i + 1) \mod \tau^{-1}$
function Output
  return $\langle B + y, c \rangle$

4.3 (W, τ, ϵ)-Multiplicative Summing

In this section, we present Algorithm 3 that provides a $(1+\epsilon)$ multiplicative approximation of the Slack Summing problem. Compared to Algorithm 1, we achieve a space reduction by representing each sum of $W\tau$ elements using $O(\log\log(RW\tau) + \log\epsilon^{-1})$ bits. Specifically, when a block ends, if its sum was y, we store $\rho = \left\lfloor \log_{(1+\epsilon/2)} y \right\rfloor$ (we allow a value of $-\infty$ for ρ if $y = 0$). To achieve $O(1)$ Output, we also store an approximate window sum B, which is now a *fixed point* fractional variable with $O(\log RW)$ bits for its integral part and additional $O(\log\epsilon^{-1})$ bits for storing a fraction. To update B's value for a new ρ, we *round down* the value of $(1+\epsilon)^{\rho}$. Specifically, for a real number x, we denote $(x)_{\downarrow} \triangleq \lfloor x \cdot k \rfloor / k$, for $k \triangleq \lceil \frac{4}{\epsilon} \rceil$. Our pseudo code appears in Algorithm 3. The algorithm requires $O\left(\tau^{-1}\left(\log\log(RW\tau) + \log\epsilon^{-1}\right) + \log RW\right)$ bits of space and is memory optimal when $R = W^{O(1)}$ and $\tau = \Omega\left(\frac{1}{\log RW}\right)$.

The full analysis of Algorithm 3 is deferred to the full version of this paper [5]. Next, we present an alternative (W, τ, ϵ)-Multiplicative Summing algorithm that achieves optimal space consumption for $\tau = \Theta(1)$, regardless of the value of R.

Improved (W, τ, ϵ)-Multiplicative Summing for for $\tau = \Theta(1)$

Algorithm 4 is more space efficient than Algorithm 3 but has a query time of $O(\tau^{-1})$. For $\tau = \Theta(1)$, Algorithm 4 is memory optimal *and* supports constant time queries even if $R = W^{\omega(1)}$; for this case, Algorithm 3 requires $\Omega(\log R)$ bits which is sub optimal.

Intuitively, we shave the $\Omega(\log R)$ bits from the space requirement of Algorithm 3 using an approximate representation for our y variable and by not keeping the B variable that allowed $O(1)$ time queries regardless of the value of τ. To avoid using $\Omega(\log R)$ bits in y, we use a *fixed point* representation in which $O(\log\epsilon^{-1} + \log\log(RW\tau))$ bits are allocated for its integral part and another $O(\log W\tau)$ for the fractional part. The goal of y is still to approximate the sum of the elements within a block, but now we aim for the sum to be approximately $(1+\epsilon/3)^y$. Whenever a block ends, we store only the integral part of y in our cyclic array b to save space. When queried, we compute an estimate for the sum using all of the values in b, which makes our query procedure take $O(\log\tau^{-1})$ time. To use the fixed point structure of y, we use the operator $(\cdot)_{\underline{\Downarrow}}$ that rounds a real number x into $(x)_{\underline{\Downarrow}} \triangleq \lfloor x \cdot W\tau \rfloor / W\tau$. We denote $\log_{(1+\epsilon/3)}(0) = -\infty, (-\infty)_{\underline{\Downarrow}} = -\infty, \lfloor -\infty \rfloor = -\infty$ and $(1+\epsilon/3)^{-\infty} = 0$.

Algorithm 4 (W, τ, ϵ)-Multiplicative Summing Algorithm for $\tau = \Theta(1)$.

Initialization: $y = -\infty, b = \bar{0}, i = 0, c = 0$.
1: **function** Update(x)
2: $y \leftarrow \left(\log_{(1+\epsilon/3)}\left(x + (1+\epsilon/3)^y\right)\right)_{\underline{\Downarrow}}$
3: $c \leftarrow (c+1) \mod W\tau$
4: **if** $c = 0$ **then** ▷ End of block
5: $b_i \leftarrow \lfloor y \rfloor$
6: $y \leftarrow -\infty$
7: $i \leftarrow (i+1) \mod \tau^{-1}$
8: **function** Output
9: **return** $\left\langle (1+\epsilon/3)^y + \sum_{i=0}^{\tau^{-1}-1}(1+\epsilon/3)^{b_i}, c \right\rangle$

In the full version of this paper [5], we prove the following theorem:

▶ **Theorem 15.** *For $\tau = \Theta(1)$, Algorithm 4 processes elements and answers queries in $O(1)$ time, uses $O(\log(W/\epsilon) + \log\log R)$ bits, and is asymptotically optimal.*

4.4 The Mean of a Slack Window

For some applications there is value in knowing the *mean* of a slack window. For example, a load balancer may be interested in the average transmission throughput. In exact windows, the sum and the mean can be derived from each other as the window size is constant. In slack windows, the window size changes but our algorithms also return the current slack offset $0 \leq c < W\tau$. That is, by dividing $\widehat{S}$ by $W + c$ we get an estimation of the mean (we assume that stream size is larger than W). Specifically, Algorithm 1 provides the exact mean; Algorithm 2 approximates it with $R\epsilon$ additive error, while Algorithm 3 yields a $(1+\varepsilon)$ multiplicative approximation.

5 Other Measurements over Slack Windows

We now explore the benefits of the slack model for other problems.

Maximum. While maintaining the maximum of a sliding window can be useful for applications such as anomaly detection [26, 21], tracking it over an exact window is often infeasible. Specifically, any algorithms for a maximum over an (exact) window must use $\Omega\left(W \log\left(R/W\right)\right)$ bits [14]. The following theorem shows that we can get a much more efficient algorithm for slack windows. The proof appears in the full version of this paper [5] Observe the the following bounds match for τ values that are not too small ($\tau = R^{\Omega(1)-1}$).

▶ **Theorem 16.** *Tracking the maximum over a slack window deterministically requires $O\left(\tau^{-1}\log R\right)$ and $\Omega\left(\tau^{-1}\log R\tau\right)$ bits.*

Standard-Deviation. Building on the ability of our summing algorithms to provide the size of the slack window that they approximate, we can compute standard deviations over slack windows. Intuitively, the standard deviation of the window can be expressed as

$$\sigma_{\overline{W}} \triangleq \sqrt{\frac{\sum_{x\in\overline{W}}(x - m_{\overline{W}})^2}{\left|\overline{W}\right| - 1}} = \sqrt{\frac{\sum_{x\in\overline{W}} x^2 - 2m_{\overline{W}}\sum_{x\in\overline{W}} x + \overline{W}\cdot m_{\overline{W}}^2}{\left|\overline{W}\right| - 1}} = \sqrt{\frac{\sum_{x\in\overline{W}} x^2 - \overline{W}\cdot m_{\overline{W}}^2}{\left|\overline{W}\right| - 1}},$$

there $\overline{W}$ is the slack window and $m_{\overline{W}}$ is its mean. We can then use two slack summing instances to track $\sum_{x\in\overline{W}} x^2$ and $m_{\overline{W}} = |\overline{W}|^{-1}\sum_{x\in\overline{W}} x$. This gives us an algorithm that computes the exact standard deviation over slack windows using $O(\tau^{-1}\log(RW\tau))$ space. Similarly, by using approximate rather than exact summing solutions we can compute a $(1+\epsilon)$ multiplicative approximation for the standard deviation using $O\big(\tau^{-1}\big(\log\epsilon^{-1} + \log\log(RW\tau)\big) + \log W\big)$ bits, or an $R\epsilon$-additive approximation using $O(\tau^{-1}\log\left(\frac{\tau}{\epsilon}\right) + \log W)$ space. We expand on this further in the full version of this paper [5].

General-Summing. General-Summing is similar to Basic-Summing, except that the integers can be in the range $\{-R,\ldots,R\}$. That is, we now allow for negative elements as well. Datar et al. [14] proved that General Sum requires $\Omega(W)$ bits, even for $R=1$ and constant factor approximation. In contrast, our exact summing algorithm from section 4.1 trivially generalizes to General-Summing and allows exact solution over slack windows.

Count-Distinct. Estimating the number of **distinct** elements in a stream is another useful metric. In networking, the packet header is used to identify different flows, and it is useful to know how many distinct of them are currently active. A sudden spike in the number of active flows is often an indication of a threat to the network. It may indicate the propagation of a worm or virus, port scans that are used to detect vulnerabilities in the system and even *Distributed Denial of Service (DDoS)* attacks [12, 18, 20].

Here, we have studied the memory reduction that can be obtained by following a similar flow to our summing algorithms – we break the stream into $W\tau$ sized blocks and run the state of the art approximation algorithm on each block separately. Luckily, count distinct algorithms are *mergable* [1]. That is, we can merge the summaries for each block to obtain an estimation of the number of distinct items in the union of the blocks. In the full version of this paper [5], we show that this approach yields an algorithm with superior space and query time compared to the state of the art algorithms for counting distinct elements over sliding windows [11, 19]. Formally, we prove the following theorem.

▶ **Theorem 17.** *For $\tau=\Theta(1)$ and any fixed $m>0$, there exists an algorithm that uses $O(m)$ space, performs updates in constant time and answers queries in time $O(m)$, such that the result approximates a window whose size is in $[W, W(1+\tau)]$; the resulting estimation is asymptotically unbiased and has a standard deviation of $\sigma = O(\frac{1}{\sqrt{m}})$. State of the art approaches for exact windows [11, 19] require $O(m\log(W/m))$ space and $O(m\log(W/m))$ time per query for a similar standard deviation.*

6 Discussion

In this work we have explored the slack window model for multiple streaming problems. We have shown that it enables asymptotic space and time improvements. Particularly, introducing slack enables logarithmic space exact algorithms for certain problems such as Maximum and General-Summing. In contract, these problems do not admit sub-linear space *approximations* in the exact window model. Even in problems that do have sub-linear space approximations such as Standard-Deviation and Count-Distinct, adding slack asymptotically improves the space requirement and allows for constant time updates.

Much of our work has focused on the classic Basic-Summing problem. Based on our findings, we argue that allowing a slack in the window size is an attractive approximation axis as it enables greater space reductions compared to an error in the sum. As an example, for a fixed ϵ value, computing a $(1+\epsilon)$-multiplicative approximation requires $\Omega(\log(RW)\log W)$

space [14]. Conversely, a $(1+\tau)$ multiplicative error in the window size, for a constant τ, allows summing using $\Theta(\log(RW))$ bits – same as in summing W elements without sliding windows! Given that for exact windows randomized algorithms have the same asymptotic complexity as deterministic ones [3, 14], we expect randomization to have limited benefits for slack windows as well.

References

1 Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. In *ACM PODS*, 2012.

2 Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. Conga: Distributed congestion-aware load balancing for datacenters. In *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*, ACM SIGCOMM 2014, 2014. doi:10.1145/2619239.2626316.

3 Ran Ben Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Efficient Summing over Sliding Windows. In *SWAT*, 2016.

4 Ran Ben Basat, Gil Einziger, Roy Friedman, Marcelo Caggiani Luizelli, and Erez Waisbard. Constant time updates in hierarchical heavy hitters. In *ACM SIGCOMM*, 2017.

5 R. Ben Basat, G. Einziger, and R. Friedman. Give Me Some Slack: Efficient Network Measurements. *ArXiv e-prints*, 2018. arXiv:1703.01166.

6 Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Heavy hitters in streams and sliding windows. In *IEEE INFOCOM*, 2016.

7 Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Optimal elephant flow detection. In *IEEE INFOCOM*, 2017.

8 Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Randomized admission policy for efficient top-k and frequency estimation. In *IEEE INFOCOM*, 2017.

9 Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Microte: Fine grained traffic engineering for data centers. In *ACM CoNEXT*, 2011.

10 Flavio Bonomi, Michael Mitzenmacher, Rina Panigrahy, Sushil Singh, and George Varghese. An improved construction for counting bloom filters. In *ESA*, 2006.

11 Y. Chabchoub and G. Hebrail. Sliding hyperloglog: Estimating cardinality in a data stream over a sliding window. In *2010 IEEE ICDM Workshops*, 2010.

12 Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. In *ACM CSUR*, 2007.

13 Min Chen and Shigang Chen. Counter tree: A scalable counter architecture for per-flow traffic measurement. In *IEEE ICNP*, 2015.

14 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal of Computing*, 2002.

15 G. Einziger and R. Friedman. TinyLFU: A highly efficient cache admission policy. In *PDP 2014*, 2014.

16 Gil Einziger, Benny Fellman, and Yaron Kassner. Independent counter estimation buckets. In *IEEE INFOCOM*, 2015.

17 Gil Einziger and Roy Friedman. Counting with TinyTable: Every Bit Counts! In *ICDCN 2016*, 2016.

18 Cristian Estan, George Varghese, and Mike Fisk. Bitmap algorithms for counting active flows on high speed links. In *ACM IMC*, 2003.

19 Éric Fusy and Frécéric Giroire. Estimating the number of active flows in a data stream over a sliding window. In *ANALCO*, 2007.

20 Sumit Ganguly, Minos Garofalakis, Rajeev Rastogi, and Krishan Sabnani. Streaming algorithms for robust, real-time detection of ddos attacks. In *27th IEEE International Conference on Distributed Computing Systems (ICDCS 2007), June 25-29, 2007, Toronto, Ontario, Canada*, ICDCS, 2007.

21 Pedro Garcia-Teodoro, Jesús E. Díaz-Verdejo, Gabriel Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers and Security*, 2009.

22 Phillip B. Gibbons and Srikanta Tirthapura. Distributed streams algorithms for sliding windows. In *SPAA*, 2002.

23 Nan Hua, Bill Lin, Jun (Jim) Xu, and Haiquan (Chuck) Zhao. Brick: A novel exact active statistics counter architecture. In *ACM/IEEE ANCS*, 2008.

24 Abdul Kabbani, Mohammad Alizadeh, Masato Yasuda, Rong Pan, and Balaji Prabhakar. Af-qcn: Approximate fairness with quantized congestion notification for multi-tenanted data centers. In *IEEE HOTI*, 2010.

25 Yang Liu, Wenji Chen, and Yong Guan. Near-optimal approximate membership query over time-decaying windows. In *IEEE INFOCOM*, 2013.

26 B. Mukherjee, L.T. Heberlein, and K.N. Levitt. Network intrusion detection. *Network, IEEE*, 1994.

27 Moni Naor and Eylon Yogev. Sliding bloom filters. In *ISAAC*. Springer, 2013.

28 Erez Tsidon, Iddo Hanniel, and Isaac Keslassy. Estimators also need shared values to grow together. In *IEEE INFOCOM*, 2012.

29 Hao Wang, H. Zhao, Bill Lin, and Jun Xu. Dram-based statistics counter array architecture with performance guarantee. *IEEE/ACM Transactions on Networking*, 2012.

30 Li Yang, Wu Hao, Pan Tian, Dai Huichen, Lu Jianyuan, and Liu Bin. Case: Cache-assisted stretchable estimator for high speed per-flow measurement. In *IEEE INFOCOM*, 2016.

31 L. Ying, R. Srikant, and X. Kang. The power of slightly more than one sample in randomized load balancing. In *IEEE INFOCOM*, 2015.

Spanning-Tree Games

Dan Hefetz[1]
Department of Computer Science, Ariel University, Israel

Orna Kupferman[2]
School of Computer Science and Engineering, The Hebrew University, Israel

Amir Lellouche
Department of Computer Science, Weizmann Institute of Science, Israel

Gal Vardi
School of Computer Science and Engineering, The Hebrew University, Israel

Abstract

We introduce and study a game variant of the classical spanning-tree problem. Our *spanning-tree game* is played between two players, MIN and MAX, who alternate turns in jointly constructing a spanning tree of a given connected weighted graph G. Starting with the empty graph, in each turn a player chooses an edge that does not close a cycle in the forest that has been generated so far and adds it to that forest. The game ends when the chosen edges form a spanning tree in G. The goal of MIN is to minimize the weight of the resulting spanning tree and the goal of MAX is to maximize it. A strategy for a player is a function that maps each forest in G to an edge that is not yet in the forest and does not close a cycle.

We show that while in the classical setting a greedy approach is optimal, the game setting is more complicated: greedy strategies, namely ones that choose in each turn the lightest (MIN) or heaviest (MAX) legal edge, are not necessarily optimal, and calculating their values is NP-hard. We study the approximation ratio of greedy strategies. We show that while a greedy strategy for MIN guarantees nothing, the performance of a greedy strategy for MAX is satisfactory: it guarantees that the weight of the generated spanning tree is at least $\frac{w(MST(G))}{2}$, where $w(MST(G))$ is the weight of a maximum spanning tree in G, and its approximation ratio with respect to an optimal strategy for MAX is $1.5 + \frac{1}{w(MST(G))}$, assuming weights in $[0, 1]$. We also show that these bounds are tight. Moreover, in a stochastic setting, where weights for the complete graph K_n are chosen at random from $[0, 1]$, the expected performance of greedy strategies is asymptotically optimal. Finally, we study some variants of the game and study an extension of our results to games on general matroids.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Algorithms, Games, Minimum/maximum spanning tree, Greedy algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.35

Related Version A full version of the paper is available at `http://www.cs.huji.ac.il/~ornak/publications/mfcs18a.pdf`.

Acknowledgements We thank Yuval Peled for helpful discussions.

[1] The research leading to this paper was done when the author was visiting the Hebrew University.
[2] The research leading to this paper has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013).

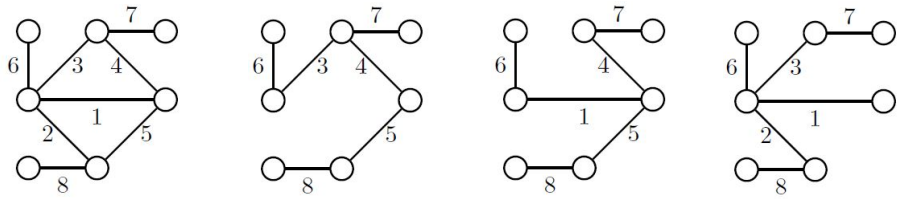


Figure 1 A weighted graph (a), its maximal spanning tree (b), and the outcomes of an optimal strategy (c) and a greedy one (d).

1 Introduction

The fundamental *minimum (respectively, maximum) spanning tree problem* receives as an input a connected edge-weighted undirected graph and searches for a spanning tree, namely an acyclic subgraph that connects all vertices, with a minimum (respectively, maximum) weight. The problem can be solved efficiently [19, 26]. It has attracted much attention, has led to a lot of research on algorithms, and has many applications [28, 10, 14].

We introduce and study a natural game variant of the classical problem. Our *spanning-tree game* is played between two players, MIN and MAX, who alternate turns in jointly constructing a spanning tree of a given connected weighted graph $G = \langle V, E, w \rangle$. Starting with the empty graph, in each turn a player chooses an edge that does not close a cycle in the forest that has been generated so far and adds it to that forest. The game ends when the chosen edges form a spanning tree in G, that is, after $|V| - 1$ turns. The goal of MIN is to minimize the weight of the resulting spanning tree and the goal of MAX is to maximize it. A *strategy* for a player is a function that maps each forest in G to one of its legal moves, namely, it maps a forest $F \subseteq E$ to an edge $e \in E \setminus F$ such that $F \cup \{e\}$ is also a forest. Given two strategies π_{max} and π_{min}, we define the *outcome* of π_{max} and π_{min} as the spanning tree obtained when MAX and MIN follow π_{max} and π_{min}, respectively, in a turn-based game in which MAX moves first. The *value* of a strategy π_{max} of MAX is the minimum over all strategies π'_{min} of MIN of the weight of the spanning tree that is the outcome of the game in which MAX follows π_{max} and MIN follows π'_{min}. Then, an optimal strategy for MAX is a strategy with a maximum value. Thus, an optimal strategy for MAX is one that obtains the maximal value against the most hostile behavior (intuitively, the "most minimizing" strategy) of MIN. The value of a strategy for MIN is defined dually. In particular, an optimal strategy for MIN is one that obtains the minimal value against the "most maximizing" strategy for MAX. In this paper we focus on values of strategies of MAX. Indeed, unless we bound the ratio between the weights of the heaviest and lightest edges in the graph, we cannot bound the "damage" that MAX can cause MIN, namely the ratio between the performance of min strategies and the minimum spanning tree, making the study of the game setting from the viewpoint of MIN less interesting.

▶ **Example 1.** Consider the weighted graph G appearing in Figure 1 (a). The weight of G's (unique, in this example) maximum spanning tree is 33 (see (b)). An optimal strategy for MAX chooses in its first two moves the edges with weights 5 and 4, leading, against an optimal strategy of MIN, to the spanning tree of weight 31 appearing in (c).

The transition from the classical *one-player* setting of the spanning-tree problem to a *two-player* setting corresponds to a transition from *closed systems*, which are completely under our control, to *open systems*, in which we have to contend with adversarial environments. Such a transition has been studied in computer science in logic [8, 27], complexity [6], and

temporal reasoning [23], and it attracts growing attention now in *algorithmic game theory*, cf. [24]. Our work here studies this transition in graph theory. For the basic problem of reachability, the two-player setting has given rise to *alternating graph reachability* [8]. We find it very interesting to study how other basic problems and concepts in graph algorithms evolve when we shift to a two-player setting [20]. Several graph games of this type were previously studied. For example, consider the general setting in which MAX and MIN alternately claim edges of a graph G while making sure the graph they build together satisfies some monotone decreasing property. The *Turán numbers* and *Saturation numbers* refer to the number of edges that can be claimed while the property is maintained [13, 17]. Likewise, researchers have studied the *game chromatic number* of G, namely the smallest k for which MIN has a strategy to color all vertices in a game in which MAX and MIN alternately properly color the vertices of G using the colors $\{1, \ldots, k\}$ [1]. Finally, a game variant of the *maximum-flow problem*, where the algorithm can direct the flow only in a subset of the vertices is studied in [21].

Before we continue to describe our results, let us survey several games that have been studied and are based on minimum or maximum spanning trees. In the *cooperative minimum cost spanning tree game* [7, 2], the cost allocation between users of a minimum spanning tree is considered. Different properties of this cooperative game have been studied, such as the core and the nucleolus [15, 16], the Shapley value [18], and more [11]. The *Stackelberg minimum spanning tree game* [4, 5] is a one-round two-player network pricing game. The game is played on a graph, whose edges are colored either red or blue, with the red edges having a given fixed cost. The first player chooses an assignment of prices to the blue edges, and the second player then buys the cheapest possible minimum spanning tree, using any combination of red and blue edges. The goal of the first player is to maximize the total price of purchased blue edges. *Shannon's switching game* is another related two-player game. Two players take turns coloring the edges of an arbitrary graph. One player has the goal of connecting two distinguished vertices by a path of edges of her color. The other player aims to prevent this by using her color instead (or, equivalently, by erasing edges) [22, 3].

The classical maximum spanning-tree problem can be solved efficiently. Indeed, the forests embodied in a graph induce a *matroid* [25], and thus a greedy approach is optimal. Accordingly, in Kruskal's algorithm [19] for the maximum spanning-tree problem, the edges are chosen in a greedy manner, where in each step an edge with a maximum weight that does not close a cycle is added.

We study greedy strategies in the spanning-tree game. There, MAX always chooses an edge with a maximum weight that does not close a cycle. We first show that the game setting is indeed more complicated. First, greedy strategies are not necessarily optimal. For example, in the graph from Example 1, a greedy strategy for MAX chooses in its first three moves the edges with weight 8, 7, and 6, leading to the spanning tree of weight 27 appearing in Figure 1 (d). In addition, we show that given a strategy for MAX, it is NP-complete to calculate its value, and NP-hardness holds already for greedy strategies. Subsequently, we turn to study how well greedy strategies for MAX perform. We evaluate them with respect to the value of the maximum spanning tree, and with respect to the value of an optimal strategy for MAX. We analyze both the general and stochastic settings. We view our findings in both evaluations as good news. Indeed, greedy strategies for MAX ensure surprisingly tight approximations in all cases.

It is not hard to see that the value of any greedy strategy for MAX is at least half the weight of a maximum spanning tree. Indeed, the tree generated by such a strategy includes at least the heavier half of the set of edges that are chosen by a greedy algorithm in the classical

setting. Much harder is the study of the approximation ratio of a greedy strategy for MAX with respect to an optimal strategy for her. We show that when the weight of the maximum spanning tree tends to infinity, the approximation ratio tends to 1.5. More formally, assuming that the weights are normalized to values in $[0, 1]$ (note that such a normalization does not affect the ratio between the values of different strategies), we show an approximation ratio of $1.5 + \frac{1}{w(MST(G))}$, where $w(MST(G))$ is the weight of a maximum spanning tree of G. We show that our results are tight: for every odd integer $n \geq 1$, there exists a weighted graph $G = \langle V, E, w\rangle$ with $w(MST(G)) = 2n$, such that the value of the greedy strategy for MAX is n, whereas the value of an optimal strategy is $\lceil \frac{n}{2} \rceil + n$. Thus, the ratio between the maximal spanning tree and the value of the greedy strategy is 2, and the ratio between the values of the optimal and the greedy strategies is $1.5 + \frac{1}{w(MST(G))}$. We also show that, unlike the case of greedy strategies of MAX, one cannot bound the approximation ratio of greedy strategies of MIN. As we elaborate in Section 7, since the set of forests that are subgraphs of a given graph form the family of independent sets of a matroid, many of our results go beyond the spanning-tree problem and apply to matroids in a game setting.

We then study the approximation ratio of greedy strategies for MAX in a stochastic setting. Namely, we study the game played on complete graphs whose edge-weights are chosen by a uniform distribution over $[0, 1]$. Building on results of [12] regarding the weight of maximum and minimum spanning trees in such randomly weighted graphs, we are able to show that, in this setting, the approximation ratio of any greedy max strategy is asymptotically almost surely (a.a.s., for brevity) 1. Thus, while in the worst case the approximation ratio is 2 with respect to a maximum spanning tree and it tends to 1.5 with respect to an optimal strategy, it is a.a.s. 1 when we choose the edge-weights uniformly at random.

Finally, we study two variants of the setting. First, a finer definition of an approximation ratio, where performance of a strategy for MAX is examined with respect to all strategies of MIN, and second, a two-turn variant of the game, where MAX first chooses a forest of size k, for a parameter k of the game, and then MIN completes the forest to a spanning tree.

2 Preliminaries

2.1 Graphs and Weighted Graphs

An undirected *graph* is a pair $G = \langle V, E\rangle$, where V is a finite set and E is a set of pairs of elements of V. We refer to the elements of V as *vertices* and to the elements of E as *edges*. A graph may contain parallel edges. A *path* in G is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that $\langle v_i, v_{i+1}\rangle \in E$ for all $1 \leq i < k$. A *cycle* in G is a path $v_1, v_2, \ldots, v_k$ for which $v_1 = v_k$. A graph $G = \langle V, E\rangle$ is *connected* if for every two vertices $v, v' \in V$, there is a path between v and v' in G. A *tree* is a connected graph with no cycles. A *forest* is a graph with no cycles, namely a collection of trees. A *spanning tree of* G is a tree $\langle V, T\rangle$, for a subset $T \subseteq E$. Note that the size of a spanning tree is $n - 1$. When the set V of vertices is clear from the context, we describe trees and forests by their sets of edges only.

A *weighted graph* $G = \langle V, E, w\rangle$ augments a graph with a weight function $w : E \to \mathbb{R}^+$. We extend w to subsets of E in the expected way, i.e., $w : 2^E \to \mathbb{R}^+$ is such that for all $A \subseteq E$, we have $w(A) = \sum_{e\in A} w(e)$. In the *maximum spanning tree* problem, we are given a weighted graph G and seek a spanning tree for G of a maximum weight. Note that G may have several maximum spanning trees. By abuse of notation, we use $MST(G)$ to denote any maximum spanning tree of G.

2.2 Matroids

A finite matroid $\mathcal{M}$ is a pair $\langle E, \mathcal{I} \rangle$, where E is a finite set (called *the ground set*) and $\mathcal{I}$ is a family of subsets of E (called *the independent sets*) that satisfies the following three properties: (1) $\mathcal{I}$ is not empty, (2) *The hereditary property*: If $X \in \mathcal{I}$ and $Y \subseteq X$, then $Y \in \mathcal{I}$, and (3) *The independent set exchange property*: If X and Y are in $\mathcal{I}$ and $|X| > |Y|$, then there is an element $x \in X \setminus Y$ such that $Y \cup \{x\}$ is in $\mathcal{I}$.

For a graph $G = \langle V, E \rangle$, let $\mathcal{F}_G$ be the set of forests in G. The pair $\langle E, \mathcal{F}_G \rangle$ is a matroid and is called *the cycle matroid of* G (see, e.g., [25]).

2.3 The Spanning-Tree Game

We consider a game variant of the maximum spanning tree problem: there are two players, MAX and MIN, who alternate turns in jointly constructing a spanning tree of a given weighted graph. Starting with the empty graph, in each turn, a player chooses an edge that does not close a cycle in the forest that has been generated so far and adds it to that forest. The game ends when the chosen edges are forming a spanning tree, that is, after $n-1$ turns. The goal of MIN is to minimize the weight of the resulting spanning tree and the goal of MAX is to maximize it. Formally, we have the following.

Let $G = \langle V, E, w \rangle$ be a weighted graph, and let $\mathcal{F}_G$ be the set of all forests $F \subseteq E$. A *configuration* in the *spanning-tree game* is a forest $F \in \mathcal{F}_G$. Let $M : \mathcal{F}_G \to 2^E$ be a function that maps a configuration F to the set of all legal moves for a player when the game is in F. Formally, $M(F) = \{e \in E \setminus F : \text{ the graph } \langle V, F \cup \{e\} \rangle \text{ has no cycles}\}$.

A *strategy* for a player is a function $\pi : \mathcal{F}_G \to E$ that maps each configuration to one of its legal moves. Thus, for all $F \in \mathcal{F}_G$, we have $\pi(F) \in M(F)$. If $M(F) = \emptyset$ (that is, when F is already a spanning tree), then $\pi(F)$ is undefined.[3] Given two strategies π_{max} and π_{min}, we define the *outcome* of π_{max} and π_{min}, denoted $T(\pi_{max}, \pi_{min})$, as the spanning tree obtained when MAX and MIN follow π_{max} and π_{min}, respectively, in a turn-based game in which MAX moves first. Formally, $T(\pi_{max}, \pi_{min}) = \{e_1, \ldots, e_{n-1}\}$ is such that for all $1 \leq i \leq n-1$, the following holds.

$$e_i = \begin{cases} \pi_{max}(\{e_1, e_2, \ldots, e_{i-1}\}) & \text{if } i \text{ is odd,} \\ \pi_{min}(\{e_1, e_2, \ldots, e_{i-1}\}) & \text{if } i \text{ is even.} \end{cases}$$

We use $w(\pi_{max}, \pi_{min})$ to denote the weight of $T(\pi_{max}, \pi_{min})$. Thus, $w(\pi_{max}, \pi_{min}) = w(T(\pi_{max}, \pi_{min}))$.

We refer to a strategy for MAX as a *max strategy* and to a strategy for MIN as a *min strategy*. Note that MAX moves when the current configuration has an even number of edges, and MIN moves when the configuration has an odd number of edges. Let $\mathcal{F}_G^{even}$ and $\mathcal{F}_G^{odd}$ be the subsets of $\mathcal{F}_G$ that contain forests of even and odd sizes, respectively. Let Π_{max} and Π_{min} be the set of all possible strategies for the MAX and MIN players, respectively. By the above, Π_{max} contains strategies $\pi_{max} : \mathcal{F}_G^{even} \to E$ and Π_{min} contains strategies

[3] We could have defined π to return a special signal, say $\perp$, in this case, but we ignore it and assume that the game ends after $n-1$ rounds, so there is no need to apply a strategy from configurations that are spanning trees.

$\pi_{min} : \mathcal{F}_G^{odd} \to E$.[4] We evaluate a max strategy π_{max} by its performance against a best (that is, most minimizing) min strategy. Formally, we define the *value* of a max strategy by

$$val_{max}(\pi_{max}) = \min\{w(\pi_{max}, \pi_{min}) : \pi_{min} \in \Pi_{min}\}.$$

Since the number of strategies is finite, the above expression always has a minimum and is thus well defined. Dually, we evaluate a min strategy π_{min} by its performance against a best (that is, most maximizing) max strategy. Formally, we define the value of a min strategy by $val_{min}(\pi_{min}) = \max\{w(\pi_{max}, \pi_{min}) : \pi_{max} \in \Pi_{max}\}$. Our study here focuses on max strategies. Essentially, our choice follows from the fact that, unlike the case of max strategies, one cannot bound the ratio between the outcome of an optimal or a greedy min strategy and the minimum spanning tree. Intuitively, it follows from the fact that the performance of strategies is strongly related to our ability to guarantee a favorable outcome even if we can control only half of the choices. Such a control guarantees that MAX can add to the spanning tree at least half of the heaviest edges in a maximum spanning tree. Such a control also guarantees that MIN can add to the spanning tree at least half of the lightest edges in a minimum spanning tree. Without, however, a bound on the ratio between the heaviest and lightest edge, such a guarantee is not of much help. In the full version, we motivate this choice further and present some results on min strategies.

The following lemma is an easy useful observation on the amount of control MAX and MIN have on the outcome of the game.

▶ **Lemma 2.** *Let $G = \langle V, E, w\rangle$ be a weighted graph and let F be a forest of G. Then,* MAX *has a strategy to ensure that the outcome includes at least $\lceil |F|/2 \rceil$ edges of F, and* MIN *has a strategy to ensure that the outcome includes at least $\lfloor |F|/2 \rfloor$ edges of F.*

Proof. We prove our claim for MIN; the proof for MAX is analogous. It suffices to show that, in each of his first $\lfloor |F|/2 \rfloor$ moves, MIN can claim an edge of F. For every $1 \leq i \leq \lfloor |F|/2 \rfloor$, let $e_1, \ldots, e_{2i-1}$ denote the edges claimed by both players up until MIN's i-th move. In his i-th move, MIN claims an arbitrary edge $e_{2i} \in F \setminus \{e_1, \ldots, e_{2i-1}\}$ such that $\{e_1, \ldots, e_{2i-1}, e_{2i}\}$ spans a forest. Such an edge e_{2i} exists since $|F| > 2i - 1 = |\{e_1, \ldots, e_{2i-1}\}|$ and both F and $\{e_1, \ldots, e_{2i-1}\}$ are forests of G, i.e., independent sets in its cycle matroid. ◀

2.4 Optimal and Greedy Strategies

We define the following strategies:

- An *optimal max strategy* is a strategy $\pi^*_{max} \in \Pi_{max}$ such that for every strategy $\pi_{max} \in \Pi_{max}$, we have $val_{max}(\pi^*_{max}) \geq val_{max}(\pi_{max})$. Such a strategy necessarily exists as the number of max strategies is finite.
- Similarly, $\pi^*_{min} \in \Pi_{min}$ is an *optimal min strategy*, if for every strategy $\pi_{min} \in \Pi_{min}$, we have $val_{min}(\pi^*_{min}) \leq val_{min}(\pi_{min})$.
- A strategy $g_{max} \in \Pi_{max}$ is a *greedy strategy* for MAX if for every configuration $F \in \mathcal{F}_G^{even}$, it holds that $g_{max}(F)$ is a heaviest edge in $M(F)$. Formally, for every configuration $F \in \mathcal{F}_G^{even}$, we have $g_{max}(F) \in \{e \in M(F) : w(e) = \max\{w(e') : e' \in M(F)\}\}$.

[4] Formally, by our definition of a strategy, every strategy for MAX and every strategy for MIN should have a well-defined legal move for every configuration in $\mathcal{F}_G$. We have chosen to restrict the definition of such strategies only to the configurations they might actually encounter during play. For completeness, one can define them for all the remaining configurations arbitrarily or, again, by using the symbol $\perp$. Also, note that strategies are *positional*, in the sense they ignore the way in which configurations have been obtained. It is easy to see that memoryfull strategies are not stronger in the spanning-tree game.

▶ Remark. There may be several optimal and greedy strategies but, from now on, for each weighted graph G we define π^*_{min}, π^*_{max}, and g_{max} as one of the strategies that satisfy the corresponding conditions and, sometimes, write "the optimal min strategy" or "the greedy max strategy". Moreover, when evaluating the performance of a greedy strategy, we consider the worst case. That is, the value of a greedy strategy is $\min\{val_{max}(g_{max}) : g_{max} \text{ is a greedy strategy in } \Pi_{max}\}$.

2.5 On the Complexity of Evaluating Strategies for MAX

Recall that the maximum spanning-tree problem can be solved in polynomial time. A possible way of computing π^*_{min} and π^*_{max} is by solving a Minmax problem, which requires exponential time. We show here that the game setting is indeed more complex than the classical one-player setting. In fact, even evaluating the value of a symbolically given max strategy is co-NP-complete, and the co-NP lower bound holds also for greedy strategies.

▶ **Theorem 3.** *Let π_{max} be a max strategy given by a linear ordering $e_1, \ldots, e_{|E|}$ of the edges in E, where π_{max} chooses in each step the edge e_j with the minimal index j for which e_j is a legal move. Let k be an integer. Deciding whether $val_{max}(\pi_{max}) > k$ is co-NP-complete. Furthermore, it is co-NP-hard already when π_{max} is a worst greedy strategy for* MAX*, that is, a greedy strategy with the lowest value.*

Proof. First, if $val_{max}(\pi_{max}) \leq k$ then there is a polynomial witness that includes the edges that MIN chooses in each turn, such that the weight of the outcome is at most k. Hence the membership in co-NP.

We now show the lower bound. Let $G = \langle V, E\rangle$ be a graph, let $S \subseteq V$, and let k be an integer. The Steiner-tree problem, namely, deciding whether there is a tree of size at most k in G that spans S, is NP-hard. We show a reduction from the Steiner tree problem. We construct a weighted graph $G' = \langle V', E', w'\rangle$ as follows. Let u_0 be a vertex in V. The set V' is obtained from V by adding k new vertices, namely $V' = V \cup \{u_1, \ldots, u_k\}$. The set E' is obtained from E by adding the edges $\{\langle u_i, u_{i+1}\rangle : 0 \leq i < k\} \cup (S \times S)$, where parallel edges are allowed. That is, an edge $e \in S \times S$ is added even if it already appears in E. For every $e \in E$ we define $w'(e) = 0$, and for every new edge $e \in E' \setminus E$ we define $w'(e) = 1$. Let π_{max} be a max strategy in which MAX first chooses edges in $\{\langle u_i, u_{i+1}\rangle : 0 \leq i < k\}$, and when it is not possible anymore she chooses edges in $S \times S$, and when it is not possible anymore she chooses edges in E. We prove that there is a tree in G that spans S and has size at most k iff $val_{max}(\pi_{max}) \leq k$. Assume that there is a tree in G that spans S and has size at most k. We denote this tree by T. Then, while MAX chooses edges in $\{\langle u_i, u_{i+1}\rangle : 0 \leq i < k\}$, MIN can choose all the edges of T and thus ensure that MAX will not be able to choose edges in $S \times S$ later. Since the edges $\{\langle u_i, u_{i+1}\rangle : 0 \leq i < k\}$ appear in every spanning tree, the value of π_{max} is k.

Assume now that there is no tree in G that spans S and has size at most k. Thus, after all the edges in $\{\langle u_i, u_{i+1}\rangle : 0 \leq i < k\}$ are chosen, there are still edges in $S \times S$ that MAX can choose, and therefore the value of π_{max} is strictly larger than k.

Finally, note that the strategy π_{max} is a worst greedy strategy for MAX, and hence the problem is co-NP-hard already for this case. ◀

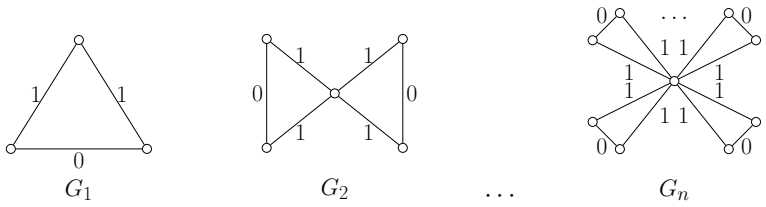


Figure 2 A sequence of weighted graphs $G_1, G_2, \ldots$ such that G_n satisfies $n = val_{max}(g_{max}) = val_{max}(\pi^*_{max}) = \frac{1}{2} \cdot w(MST(G_n))$.

3 The Performance of Optimal and Greedy Strategies w.r.t. the Maximum Spanning Tree

In the game setting, MAX has a chance to choose only half of the edges in the spanning tree. It is thus not surprising that the outcome of an optimal strategy may be only half of the weight of an MST. Below we formalize this intuition, and show that the half-ratio may be obtained already by a greedy strategy (Theorem 4) and that this upper bound is tight (Theorem 5).

▶ **Theorem 4.** *For every weighted graph G, we have that $val_{max}(g_{max}) \geq \frac{1}{2} \cdot w(MST(G))$.*

Proof. Let $G = \langle V, E, w \rangle$, and let $\langle e_1, \ldots, e_{n-1} \rangle$ be a vector of the edges of some maximum spanning tree of G, where $w(e_i) \geq w(e_{i+1})$ for every $1 \leq i < n-1$. Consider the game on G in which MAX plays according to g_{max} and MIN plays according to some strategy π_{min}. For every $1 \leq j \leq \lceil (n-1)/2 \rceil$, let x_j denote the edge of G that MAX chooses in her j-th move. For every $1 \leq j \leq \lfloor (n-1)/2 \rfloor$, let y_j denote the edge of G that MIN chooses in his j-th move. Our goal is to prove that

$$\sum_{j=1}^{\lceil (n-1)/2 \rceil} w(x_j) + \sum_{j=1}^{\lfloor (n-1)/2 \rfloor} w(y_j) \geq \frac{1}{2} \cdot \sum_{j=1}^{n-1} w(e_j).$$

We prove that, in fact, already $\sum_{j=1}^{\lceil (n-1)/2 \rceil} w(x_j) \geq \frac{1}{2} \cdot \sum_{j=1}^{n-1} w(e_j)$. Since all edge-weights are non-negative, this implies our goal.

To see this, consider an integer $0 \leq k < (n-1)/2$. Note that $|\{x_1, \ldots x_k, y_1, \ldots, y_k\}| = 2k < 2k+1 = |\{e_1, \ldots, e_{2k+1}\}|$. Since, moreover, $\{x_1, \ldots x_k, y_1, \ldots, y_k\}$ and $\{e_1, \ldots, e_{2k+1}\}$ are independent sets of a matroid (namely, the cycle matroid of G), there exists some edge $e \in \{e_1, \ldots, e_{2k+1}\} \cap M(\{x_1, \ldots x_k, y_1, \ldots, y_k\})$. Since MAX plays according to the greedy strategy, it must be that $w(x_{k+1}) \geq w(e) \geq w(e_{2k+1})$. Hence, $\sum_{j=1}^{\lceil (n-1)/2 \rceil} w(x_j) \geq \sum_{j=0}^{\lceil (n-1)/2 \rceil - 1} w(e_{2j+1}) \geq \frac{1}{2} \cdot \sum_{j=1}^{n-1} w(e_j)$, and the statement follows. ◀

▶ **Theorem 5.** *For every $n \geq 1$, there is a weighted graph G_n such that $n = val_{max}(\pi^*_{max}) = \frac{1}{2} \cdot w(MST(G_n))$. In fact, for G_n we also have $val_{max}(g_{max}) = n$.*

Proof. See the weighted graphs $G_1, G_2, \ldots$ in Figure 2. Note that $MST(G_n)$ includes all the edges with weight 1, and that MIN can ensure that all the edges with weight 0 are chosen. ◀

4 The Performance of Greedy Strategies w.r.t. Optimal Ones

In this section we study the performance of the greedy max strategy in comparison to the optimal max strategy. We first define formally what it means for two strategies to approximate each other.

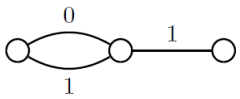


Figure 3 $val_{max}(g_{max}) = 1$ whereas $val_{max}(\pi^*_{max}) = 2$.

4.1 Approximating Strategies

Given a weighted graph $G = \langle V, E, w\rangle$, consider two max strategies $\pi_{max}, \pi'_{max} \in \Pi_{max}$ and a factor $\alpha \geq 1$. We say that π_{max} is an *α-max-approximation* of π'_{max} if

$$val_{max}(\pi_{max}) \geq 1/\alpha \cdot val_{max}(\pi'_{max}).$$

That is, intuitively, π'_{max} is at most α times better than π_{max}, where, in both cases, we assume that MIN follows an optimal min strategy.

The *max competitive ratio* of a strategy $\pi_{max} \in \Pi_{max}$ is then the smallest factor α such that π_{max} is an α-max approximation of π^*_{max}. Namely, $\frac{val_{max}(\pi^*_{max})}{val_{max}(\pi_{max})}$.

▶ Remark (Universal Approximation). We could have defined strategy approximations in a different way, by stating that π_{max} is an α-max-approximation of π'_{max} if for every strategy $\pi_{min} \in \Pi_{min}$, we have that $w(\pi_{max}, \pi_{min}) \geq \frac{1}{\alpha} \cdot w(\pi'_{max}, \pi_{min})$. We refer to such an approximation as *α-max universal approximation*. Intuitively, while in α-max-approximation the performance of the two max strategies is examined with respect to optimal (possibly different from each other) min strategies, in α-max universal approximation the performance is examined with respect to every min strategy – the same min strategy against both max strategies. In the full version, we show that α-max universal approximation is strictly finer than α-max approximation. That is, for all $\pi_{max}, \pi'_{max} \in \Pi_{max}$ and $\alpha \geq 1$, if π_{max} is an α-max universal approximation of π'_{max}, then π_{max} is an α-max approximation of π'_{max}, yet possibly π_{max} is an α-max approximation of π'_{max} but it is not an α-max universal approximation of π'_{max}. Moreover, working with a max universal approximation, the competitive ratio of the greedy strategy with respect to the optimal strategy is 2, higher than the ratio we prove in Theorem 7, when working with a max approximation.

4.2 The Competitive Ratio of Greedy Max Strategies

We turn to study the max competitive ratio of the greedy strategy. For convenience, we assume that the weight function w is normalized so that $\max\{w(e) : e \in E\} = 1$. It is easy to see that such a normalization is always possible and does not change the ratio of the weights of any two spanning trees.

▶ **Theorem 6.** *The max competitive ratio of the greedy strategy is* 2.

Proof. We first prove that g_{max} is a 2-max approximation. By Theorem 4, we have $2 \cdot val_{max}(g_{max}) \geq w(MST(G))$. In addition, as no max strategy can perform better than the weight of a maximum spanning tree, we have that $w(MST(G)) \geq val_{max}(\pi_{max})$ for all $\pi_{max} \in \Pi_{max}$. Hence, $val_{max}(g_{max}) \geq \frac{1}{2} \cdot val_{max}(\pi_{max})$ for all $\pi_{max} \in \Pi_{max}$, and we are done.

Next, in order to prove that the factor 2 is tight, consider the graph in Figure 3. It is easy to see that while an optimal max strategy would choose first the parallel edge with weight 1, leading to a spanning tree of weight 2, a greedy strategy may choose first the edge on the right, leading to a spanning tree of weight 1. ◀

4.3 A Tighter Analysis

While showing tightness in the general case, the lower-bound proof in Theorem 6 is based on a graph with a maximum spanning tree of a very small weight. In this section we show that g_{max} approximates π^*_{max} better when $w(MST(G))$ is large.

▶ **Theorem 7.** *Let $G = \langle V, E, w\rangle$ be a weighted graph, and assume that the weights in G are normalized such that the maximum weight of an edge in E is 1. Then, g_{max} is a $1.5 + \frac{1}{w(MST(G))}$-max-approximation of π^*_{max}.*

Proof. We start with a brief description of the main idea of the proof. Let $\langle e_1, \ldots, e_{n-1}\rangle$ be the edges claimed by MAX and MIN in this order when MAX follows a greedy strategy g_{max} and MIN follows a strategy π_{min} that is optimal against g_{max}. Using the fact that g_{max} is a greedy strategy, we will show that MIN has a strategy π'_{min} such that, when pitted against an optimal strategy π^*_{max} of MAX (in fact, against any max strategy), it ensures that the weight of the resulting spanning tree is at most $(1.5 + 1/w(MST(G))) \cdot \sum_{i=1}^{n-1} w(e_i)$. Note that π'_{min} might not be an optimal min strategy, but this only makes the proven result stronger. The heart of the argument is that as long as MAX can claim high (in comparison to what she claimed when she followed g_{max}) weight edges, MIN can claim quite a few low (in comparison to what he claimed when he followed π_{min}) weight edges.

We proceed to the formal proof. Let $\pi_{min} \in \Pi_{min}$ be a min strategy for which $val_{max}(g_{max}) = w(g_{max}, \pi_{min})$. Let $\langle e_1, \ldots, e_{n-1}\rangle$ be a vector of edges of $T(g_{max}, \pi_{min})$, where, for every $1 \leq i \leq n-1$, if i is odd, then e_i is chosen by MAX in her $((i+1)/2)$-th move, and if i is even, then e_i is chosen by MIN in his $(i/2)$-th move. Let $E_{odd} = \{e_1, e_3, \ldots, e_b\}$, where $b = n - 1 - (n \mod 2)$, be the edges chosen by MAX, and let $E_{even} = \{e_2, e_4, \ldots, e_a\}$, where $a = n - 2 + (n \mod 2)$, be the edges chosen by MIN. Let $d_1 > \ldots > d_k$ be the distinct weights of the edges in E_{odd}, and let $t_1, \ldots, t_k$ be positive integers such that E_{odd} contains exactly t_i edges of weight d_i for every $1 \leq i \leq k$. Let $t'_0 = 0$ and, for every $1 \leq i \leq k$, let $t'_i = t'_{i-1} + 2t_i$. Thus, $t'_i = \sum_{j=1}^{i} 2t_j$. Note that, for every $1 \leq i \leq k$, the edges of E_{odd} whose weight is d_i are $\{e_{t'_{i-1}+1}, e_{t'_{i-1}+3}, \ldots, e_{t'_i-1}\}$. For example, $w(e_1) = w(e_3) = \ldots = w(e_{2t_1-1}) = d_1$, and $w(e_{2t_1+1}) = w(e_{2t_1+3}) = \ldots = w(e_{2t_1+2t_2-1}) = d_2$. Since the weights in G are normalized so that the maximum weight of an edge in G is 1 and since g_{max} is greedy, we have that $d_1 = 1$.

We argue that MIN has a strategy π'_{min} with which he can ensure that, by deviating from the greedy strategy g_{max}, MAX does not greatly improve the weight of the tree she builds with him. We define the strategy π'_{min} as follows. Consider a forest $F_m = \{e'_1, e'_2, \ldots, e'_m\} \in \mathcal{F}^{odd}_G$, where $m < \lfloor \frac{n-1}{2} \rfloor$. Let $0 \leq i < k$ be the unique integer for which $\frac{t'_i}{2} \leq m < \frac{t'_{i+1}}{2}$. Then, $\pi'_{min}(F_m)$ is an arbitrary edge in $M(F_m) \cap \{e_2, e_4, \ldots, e_{t'_{i+1}}\}$; by definition, this is a legal move. Moreover, by the independent set exchange property of the cycle matroid of G, such an edge exists. For example, if $m < t_1$, then $\pi'_{min}(F_m)$ is an arbitrary edge of $\{e_2, e_4, \ldots, e_{2t_1}\}$ that was not chosen earlier and does not close a cycle with F_m.

Since $val_{max}(\pi^*_{max}) \leq w(\pi^*_{max}, \pi'_{min})$, it suffices to prove that $\frac{w(\pi^*_{max}, \pi'_{min})}{val_{max}(g_{max})} \leq 1.5 + \frac{1}{w(MST(G))}$. For an integer t, let $V^t_1, \ldots, V^t_{s_t}$ be the vertex sets of the connected components induced by the forest $\{e_1, \ldots, e_t\}$. Let E^t denote the set of edges of G that are contained in some connected component of $\{e_1, \ldots, e_t\}$, that is, $\langle u, v\rangle \in E^t$ if and only if there exists some $1 \leq i \leq s_t$ such that $u, v \in V^t_i$. Note that every forest in G contains at most $\sum_{j=1}^{s_t}(|V^t_j| - 1) = t$ edges of E^t.

Let $E' = \{e'_1, \ldots, e'_{n-1}\}$ denote the edge set of $T(\pi^*_{max}, \pi'_{min})$. Note that by the description of the strategy π'_{min}, for every $1 \leq i < k$, the forest $\{e'_1, e'_2, \ldots, e'_{t'_i/2}\}$ contains at least $\lfloor \frac{t'_i/2}{2} \rfloor$ edges from $E^{t'_i} \cap E_{even}$. Since $E' \cap E^{t'_i}$ contains at most t'_i edges, it follows that

$E' \cap E^{t'_i}$ contains at most $t'_i - \lfloor \frac{t'_i/2}{2} \rfloor = \lceil 1.5 \cdot \frac{t'_i}{2} \rceil$ edges from $E \setminus E_{even}$. Note that for every edge $e \notin E^{t'_i}$, we have that $w(e) \leq d_{i+1}$. Indeed, otherwise MAX would have chosen $e_{t'_i+1}$ such that $w(e_{t'_i+1}) > d_{i+1}$. Hence, $E' \setminus E_{even}$ contains at most $1.5 \cdot \frac{t'_i}{2} + 0.5$ edges from $\{e \in E : w(e) > d_{i+1}\}$.

We now show that $E' \setminus E_{even}$ contains at most $1.5 \cdot \frac{t'_k}{2} + 0.5$ edges. Assume first that $n-1$ is even and thus $t'_k = n-1$. The forest $\{e'_1, e'_2, \ldots, e'_{t'_k/2}\}$ contains at least $\left\lfloor \frac{t'_k/2}{2} \right\rfloor$ edges from E_{even}. Since E' contains t'_k edges, it follows that E' contains at most $t'_k - \lfloor \frac{t'_k/2}{2} \rfloor = \lceil 1.5 \cdot \frac{t'_k}{2} \rceil$ edges from $E \setminus E_{even}$. Hence, $E' \setminus E_{even}$ contains at most $1.5 \cdot \frac{t'_k}{2} + 0.5$ edges. Now, assume that $n-1$ is odd and thus $t'_k = n$. Note that E' contains at least $\lfloor \frac{\lfloor \frac{n-1}{2} \rfloor}{2} \rfloor = \lfloor \frac{0.5n-1}{2} \rfloor$ edges from E_{even}. Therefore, the size of $E' \setminus E_{even}$ is at most $n-1-\lfloor \frac{0.5n-1}{2} \rfloor = \lceil n-1-\frac{0.5n-1}{2} \rceil = \lceil \frac{3n}{4} - 0.5 \rceil \leq \lceil \frac{3n}{4} \rceil = \lceil 1.5 \cdot \frac{t'_k}{2} \rceil \leq 1.5 \cdot \frac{t'_k}{2} + 0.5$.

Since for every $1 \leq i < k$ the forest $E' \setminus E_{even}$ contains at most $1.5 \cdot \frac{t'_i}{2} + 0.5$ edges from $\{e \in E : w(e) > d_{i+1}\}$, and since $E' \setminus E_{even}$ contains at most $1.5 \cdot \frac{t'_k}{2} + 0.5$ edges, then the total weight of $E' \setminus E_{even}$ is at most $d_1(1.5 \cdot \frac{t'_1}{2} + 0.5) + \sum_{i=2}^{k} d_i \cdot [(1.5 \cdot \frac{t'_i}{2} + 0.5) - (1.5 \cdot \frac{t'_{i-1}}{2} + 0.5)] = d_1(1.5t_1 + 0.5) + \sum_{i=2}^{k} d_i \cdot (1.5t_i) = 0.5d_1 + \sum_{i=1}^{k} 1.5t_i d_i$.

We are now ready to bound $\frac{w(\pi^*_{max}, \pi'_{min})}{val_{max}(g_{max})}$ from above.

$$
\begin{aligned}
\frac{w(\pi^*_{max}, \pi'_{min})}{val_{max}(g_{max})} &= \frac{w(E')}{w(E_{even}) + \sum_{i=1}^{k} t_i d_i} \leq \frac{w(E_{even}) + w(E' \setminus E_{even})}{w(E_{even}) + \sum_{i=1}^{k} t_i d_i} \\
&\leq \frac{w(E_{even}) + 0.5d_1 + \sum_{i=1}^{k} 1.5t_i d_i}{w(E_{even}) + \sum_{i=1}^{k} t_i d_i} = \frac{w(E_{even}) + \sum_{i=1}^{k} t_i d_i + \sum_{i=1}^{k} 0.5t_i d_i + 0.5d_1}{w(E_{even}) + \sum_{i=1}^{k} t_i d_i} \\
&\leq 1 + \frac{\sum_{i=1}^{k} 0.5t_i d_i + 0.5d_1}{\sum_{i=1}^{k} t_i d_i} = 1.5 + \frac{0.5d_1}{\sum_{i=1}^{k} t_i d_i} \leq 1.5 + \frac{0.5}{0.5 \cdot w(MST(G))} \\
&= 1.5 + \frac{1}{w(MST(G))}.
\end{aligned}
$$

The last inequality follows from the fact $\sum_{i=1}^{k} t_i d_i \geq 0.5 \cdot w(MST(G))$ (see proof of Theorem 4) and $d_1 = 1$. ◀

The following theorem asserts that the approximation ratio given in Theorem 7 is tight.

▶ **Theorem 8.** *Let $n \geq 1$ be an odd integer. There exists a weighted graph G_n with $w(MST(G_n)) = 2n$ and with a maximum edge weight of 1, such that $\frac{val_{max}(\pi^*_{max})}{val_{max}(g_{max})} = 1.5 + \frac{1}{w(MST(G))}$.*

Proof. We define $G_n = \langle V, E, w \rangle$ as follows. First, let $V = V_1 \cup V_2$, where $V_1 = \{v_0, v_1, \ldots, v_n\}$ and $V_2 = \{v_0, u_1, \ldots, u_n\}$. Note that the vertex v_0 appears in both V_1 and V_2. Then, let $E = E_1 \cup E_2$ where $E_1 = \{\langle v_i, v_{i+1} \rangle : 0 \leq i \leq n-1\}$ and $E_2 \subseteq V_2 \times V_2$ is the disjoint union of two spanning trees T_0 and T_1 on the vertices of V_2. It is not hard to see that such two spanning trees always exist. For $n \leq 2$, one needs parallel edges, as in G_1, which appears in Figure 3. For $n \geq 3$, the graph G_n appears in Figure 4, where the edges in T_1 are solid, and these in T_0 are dashed.

For every edge $e \in E_1 \cup T_1$ we have $w(e) = 1$ and for every edge $e \in T_0$ we have $w(e) = 0$. The edges in E_1 must be contained in every spanning tree of G_n. Therefore, if m edges from T_1 are chosen during the game for some $m \leq n$, then the outcome of the game is $m + n$.

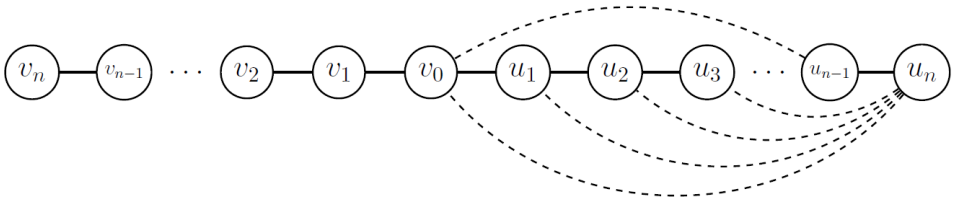


Figure 4 The graph G_n with $\frac{val_{max}(\pi^*_{max})}{val_{max}(g_{max})} = 1.5 + \frac{1}{w(MST(G_n))}$.

Thus, an optimal strategy π^*_{max} is to have as many edges from T_1 as possible. Hence, by Lemma 2 we have $val_{max}(\pi^*_{max}) = \lceil \frac{n}{2} \rceil + n$. In the strategy g_{max}, MAX chooses only the n edges in E_1, and hence $val_{max}(g_{max}) = n$.

Since n is odd, we have $\frac{val_{max}(\pi^*_{max})}{val_{max}(g_{max})} = \frac{\lceil \frac{n}{2} \rceil + n}{n} = \frac{\frac{n}{2}+0.5+n}{n} = 1.5 + \frac{1}{2n} = 1.5 + \frac{1}{w(MST(G_n))}$. ◀

5 A Stochastic Setting

The weighted graphs $\{G_n : n \in \mathbb{N}\}$ depicted in Figure 2 form an infinite family of games in which g_{max} is an optimal strategy for MAX. In this section we prove that g_{max} is not far from being optimal in a very natural and general case.

▶ **Theorem 9.** *Consider the weighted graph $G = \langle V, E, w \rangle$, where $V = [n]$, $E = \binom{[n]}{2}$, and $\{w(e) : e \in E\}$ are independent random variables, each having a uniform distribution over $[0, 1]$. Then, asymptotically almost surely (a.a.s., for brevity)*

$$\lim_{n\to\infty} \frac{val_{max}(g_{max})}{val_{max}(\pi^*_{max})} = 1.$$

The main ingredient in our proof of Theorem 9 is the following result, which is an immediate corollary of the main result of [12] (see also [9] and the many references therein).

▶ **Theorem 10.** *For $n \geq 1$, consider the complete graph with n vertices K_n, and let $\{X_e : e \in E(K_n)\}$ be independent random variables, each having a uniform distribution over $[0, 1]$. Let Y_m (respectively, Y_M) denote the weight of a minimum (respectively, maximum) spanning tree. Then*

(a) $\lim_{n\to\infty} Pr(Y_m \leq 1.21) = 1$.

(b) $\lim_{n\to\infty} Pr(Y_M \geq n - 2.21) = 1$.

Proof of Theorem 9. It readily follows from Theorem 4 and Part (b) of Theorem 10 that a.a.s. $val_{max}(g_{max}) \geq (n - 2.21)/2$. Let T be a spanning tree with weight at most 1.21; such a tree exists a.a.s. by Part (a) of Theorem 10. It follows by Lemma 2 that MIN has a strategy to ensure that the tree he builds with MAX contains at least $\lfloor |T|/2 \rfloor = \lfloor (n-1)/2 \rfloor$ edges of T. The weight of the tree they build is thus at most $1.21 + \lceil (n-1)/2 \rceil \leq (n + 2.42)/2$. Hence, a.a.s.

$$\lim_{n\to\infty} \frac{val_{max}(g_{max})}{val_{max}(\pi^*_{max})} \geq \lim_{n\to\infty} \frac{(n - 2.21)/2}{(n + 2.42)/2} = 1$$

as claimed. ◀

6 A Two-Turn Variant of the Spanning-Tree Game

In this section we study a variant of the game in which the players alternate turns only once. Formally, we have the following. A *game* is a pair $\langle G, k\rangle$, where $G = \langle V, E, w\rangle$ is a weighted graph with n vertices and $1 \leq k \leq n-1$ is an integer. In a game on $\langle G, k\rangle$, first MAX chooses a forest $F \subseteq E$ of size k. Then, MIN complements F to a spanning tree of G by choosing $n-1-k$ edges. MAX wants to maximize the weight of the resulting spanning tree and MIN aims to minimize it. Let $g_{max} \subseteq E$ be a strategy for MAX in which she chooses a forest of size k with a maximum weight, that is, MAX chooses a forest in a greedy manner. Note that while we still use the notation which was introduced in Subsection 2.4 (e.g., g_{max}), the definition of a strategy is different in this setting. A strategy π_{max} of MAX is simply the edge set of some forest of G of size k. Similarly, a strategy π_{min} for MIN is a function that, given a forest F of size k, returns a forest F' of size $n-1-k$ such that $F \cup F'$ is a spanning tree.

▶ **Theorem 11.** *Let $\langle G, k\rangle$ be a game, where $G = \langle V, E, w\rangle$ and $|V| = n$. Then, $val_{max}(g_{max}) \geq \frac{k}{n-1} \cdot w(MST(G))$.*

Proof. Let $T = \{e_1, \ldots, e_{n-1}\}$, where $w(e_1) \geq \ldots \geq w(e_{n-1})$, be an MST obtained by complementing g_{max} in a greedy manner. That is, $g_{max} = \{e_1, \ldots, e_k\}$. Note that for every $k < i \leq n-1$ we have $w(e_i) \leq w(e_k)$. Therefore, $w(MST(G)) = w(T) = w(\{e_1, \ldots, e_k\}) + w(\{e_{k+1}, \ldots, e_{n-1}\}) \leq w(g_{max}) + (n-k-1) \cdot w(e_k)$. Since $w(e_k) \leq \frac{1}{k} \cdot w(g_{max})$, we have $w(MST(G)) \leq w(g_{max}) + (n-k-1) \cdot \frac{1}{k} \cdot w(g_{max}) = \frac{n-1}{k} \cdot w(g_{max}) \leq \frac{n-1}{k} \cdot val_{max}(g_{max})$. ◀

▶ **Theorem 12.** *Let $\langle G, k\rangle$ be a game, where $G = \langle V, E, w\rangle$ and $|V| = n$. Then, g_{max} is a 2-max-approximation.*

Proof. Let π_{min} be a strategy for which $val_{max}(g_{max}) = w(g_{max}, \pi_{min})$ and let $T = T(g_{max}, \pi_{min})$. Let π^*_{max} be an optimal strategy for MAX. Consider the strategy π'_{min} of MIN in which π^*_{max} is complemented to a spanning tree as follows. Since $|\pi^*_{max}| = k$ and $|T| = n-1$, MIN can choose $n-1-k$ edges from T due to the independent set exchange property of the cycle matroid of G. For such a strategy π'_{min}, we have $val_{max}(\pi^*_{max}) \leq w(\pi^*_{max}, \pi'_{min}) \leq w(\pi^*_{max}) + w(T)$. Since g_{max} is a forest of maximum weight among all forests of G with k edges, it follows that $w(\pi^*_{max}) \leq w(g_{max})$, and thus $val_{max}(\pi^*_{max}) \leq w(g_{max}) + w(T) \leq 2 \cdot w(T) = 2 \cdot val_{max}(g_{max})$. ◀

The following result is a straightforward consequence of Theorems 11 and 12.

▶ **Corollary 13.** *Let $\langle G, k\rangle$ be a game, where $G = \langle V, E, w\rangle$ and $|V| = n$. Then, g_{max} is a $\min\{2, \frac{n-1}{k}\}$-max-approximation.*

In the following theorem we show that the approximation ratio in Corollary 13 is tight.

▶ **Theorem 14.** *Let $n > 1$ and $1 \leq k \leq n-1$ be integers. There exists a game $\langle G, k\rangle$, where $G = \langle V, E, w\rangle$ and $|V| = n$, such that $\frac{val_{max}(\pi^*_{max})}{val_{max}(g_{max})} = \min\{2, \frac{n-1}{k}\}$, where π^*_{max} is an optimal strategy for MAX in G.*

Proof. Let $V = V_1 \cup V_2$, where $V_1 = \{v_0, v_1, \ldots, v_k\}$ and $V_2 = \{v_0, u_1, \ldots, u_{n-1-k}\}$. Note that the vertex v_0 appears in both V_1 and V_2. Let $E = E_1 \cup E_2$, where $E_1 = \{\langle v_i, v_{i+1}\rangle : 0 \leq i \leq k-1\}$ and $E_2 = E(T_0) \cup E(T_1)$, where T_0 and T_1 are edge-disjoint spanning trees of $G[V_2]$ (we allow parallel edges in E_2). For every edge $e \in E_1 \cup T_1$ we set $w(e) = 1$ and for every edge $e \in T_0$ we set $w(e) = 0$. Note that if MAX chooses m edges in T_1 for some $m \leq n-1-k$, then MIN can choose $n-1-k-m$ edges in T_0 due to the independent set

exchange property of the cycle matroid of G. The edges of E_1 must be contained in every spanning tree of G. Therefore, if MAX chooses m edges from T_1, then the outcome of the game is $m+k$. Thus, the optimal strategy π^*_{max} contains as many edges from T_1 as possible, namely, $\min\{k, n-1-k\}$ edges from T_1. The strategy g_{max} contains the k edges in E_1, and therefore $val_{max}(g_{max}) = k$.

If $k \leq \frac{n-1}{2}$ then π^*_{max} contains k edges from T_1 and hence we have $\frac{val_{max}(\pi^*_{max})}{val_{max}(g_{max})} = \frac{2k}{k} = 2 = \min\{2, \frac{n-1}{k}\}$. If $k > \frac{n-1}{2}$ then π^*_{max} contains $n-1-k$ edges from T_1 and hence we have $\frac{val_{max}(\pi^*_{max})}{val_{max}(g_{max})} = \frac{n-1}{k} = \min\{2, \frac{n-1}{k}\}$. ◀

7 Discussion

We studied a game variant of the classic maximum spanning-tree problem. Both the classic problem and our spanning-tree game can be generalized in a straightforward way to all matroids. In the game setting, given a weighted matroid $M = \langle E, \mathcal{I}, w\rangle$, MAX and MIN alternate turns in claiming elements of E while ensuring that the set of elements claimed so far by both players is in $\mathcal{I}$. The game is over as soon as the set of claimed elements is a basis B of M. MAX aims to maximize the total weight of B and MIN aims to minimize it. It is not hard to show that all of our results (with the exception of Theorem 9, which deals only with weighted complete graphs) apply in this more general setting. The only non-trivial generalization is that of one specific point in the proof of Theorem 7, which we explain below.

When defining E^t, instead of relying on the connected components of the forest $\{e_1, \ldots, e_t\}$, one can use the rank function[5] r of the matroid. That is, $E^t = \{e \in E : r(\{e\} \cup \{e_1, \ldots, e_t\}) = r(\{e_1, \ldots, e_t\})\}$. It then readily follows from the definitions of r and of E^t that $|B \cap E^t| \leq t$ holds for every $B \in \mathcal{I}$.

The graph depicted in Figure 3, which is used to show that, in general, the competitive ratio of greedy strategies is 2, contains parallel edges. One then wonders whether the competitive ratio of greedy strategies is better than 2 under the assumption that the graph on which the game is played is simple. At the moment we only know that this ratio is between 5/3 and 2. One can also consider graphs that are not only simple, but have a large girth[6]. The intuition behind this is that, in order to prevent MAX from claiming a certain edge, MIN must ensure that claiming it closes a cycle, and this seems harder if all cycles are long. Moreover, when the girth is 2, i.e., there are parallel edges, we know that the competitive ratio is 2. On the other hand, when the game is played on a tree, i.e., the girth is infinite, the competitive ratio is trivially 1. This shows that increasing the girth does decrease (in some way) the competitive ratio of greedy strategies from 2 to 1.

Finally, our game is a special case of the so-called *biased game*, in which MAX claims p edges per turn and then MIN claims q edges per turn, where p and q are positive[7] integers that are allowed to grow with n. It would be interesting to study how changing the parameters p and q would affect our results.

---

[5] The rank function of a matroid $M = \langle E, \mathcal{I}\rangle$ is a mapping $r : 2^E \to \mathbb{N}$ that maps each subset A of E to the size of a largest independent set it contains; i.e., $r(A) = \max\{|B| : B \subseteq A, B \in \mathcal{I}\}$.

[6] The girth of a graph G is the length of a shortest cycle in G. If G is a forest, then its girth is defined to be ∞.

[7] In fact, by allowing $p = 0$ (respectively, $q = 0$) we get the original minimum (resp., maximum) spanning tree problem for which greedy strategies are optimal regardless of the value of q (resp., p).

References

1 T. Bartnicki, J. A. Grytczuk, H. A. Kierstead, and X. Zhu. The map coloring game. *American Mathematical Monthly*, 114:793–803, 2007.

2 C.G. Bird. On cost allocation for a spanning tree: a game theoretic approach. *Networks*, 6(4):335–350, 1976.

3 J. Bruno and L. Weinberg. A constructive graph-theoretic solution of the shannon switching game. *IEEE Transactions on Circuit Theory*, 17(1):74–81, 1970.

4 J. Cardinal, E.D. Demaine, S. Fiorini, G. Joret, S. Langerman, I. Newman, and O. Weimann. The stackelberg minimum spanning tree game. *Algorithmica*, 59(2):129–144, 2011.

5 J. Cardinal, E.D. Demaine, S. Fiorini, G. Joret, I. Newman, and O. Weimann. The stackelberg minimum spanning tree game on planar and bounded-treewidth graphs. *Journal of combinatorial optimization*, 25(1):19–46, 2013.

6 A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981.

7 A. Claus and D. J. Kleitman. Cost allocation in networks: The bulk supplier problem. *Networks*, 4(1):1–17, 1974.

8 S.A. Cook. Path systems and language recognition. In *Proc. 2nd ACM Symp. on Theory of Computing*, pages 70–72, 1970.

9 C. Cooper, A. Frieze, N. Ince, S. Janson, and J. Spencer. On the length of a random minimum spanning tree. *Combinatorics, Probability and Computing*, 25:89–107, 2016.

10 T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 1990.

11 B. Dutta and A. Kar. Cost monotonicity, consistency and minimum cost spanning tree games. *Games and Economic Behavior*, 48(2):223–248, 2004.

12 A. M. Frieze. On the value of a random minimum spanning tree problem. *Discrete Applied Mathematics*, 10:47–56, 1985.

13 Z. Füredi, D. Reimer, and A. Seress. Triangle-free game and extremal graph problems. *Congressus Numerantium*, 82:123–128, 1991.

14 R.L Graham and P. Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985.

15 D. Granot and G. Huberman. Minimum cost spanning tree games. *Mathematical programming*, 21(1):1–18, 1981.

16 D. Granot and G. Huberman. On the core and nucleolus of minimum cost spanning tree games. *Mathematical programming*, 29(3):323–347, 1984.

17 D. Hefetz, M. Krivelevich, A. Naor, and M. Stojaković. On saturation games. *European Journal of Combinatorics*, 41:315–335, 2016.

18 A. Kar. Axiomatization of the shapley value on minimum cost spanning tree games. *Games and Economic Behavior*, 38(2):265–277, 2002.

19 J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

20 O. Kupferman. Examining classical graph-theory problems from the viewpoint of formal-verification methods. In *Proc. 49th ACM Symp. on Theory of Computing*, page 6, 2017.

21 O. Kupferman, G. Vardi, and M.Y. Vardi. Flow games. In *Proc. 37th Conf. on Foundations of Software Technology and Theoretical Computer Science*, 2017, to appear.

22 A. Lehman. A solution of the shannon switching game. *Journal of the Society for Industrial and Applied Mathematics*, 12(4):687–725, 1964.

23 O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. on Principles of Programming Languages*, pages 97–107, 1985.

24 N. Nisan, T. Roughgarden, E. Tardos, and V.V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

25 J. Oxley. *Matroid Theory, 2nd edition*. Oxford University Press, 2011.

26 R.C. Prim. Shortest connection networks and some generalizations. *Bell Labs Technical Journal*, 36(6):1389–1401, 1957.

27 L.J. Stockmeyer. On the combinational complexity of certain symmetric boolean functions. *Mathematical Systems Theory*, 10:323–336, 1977.

28 R.E. Tarjan. *Data structures and network algorithms*. SIAM, 1983.

Faster Exploration of Degree-Bounded Temporal Graphs

Thomas Erlebach
Department of Informatics, University of Leicester, Leicester, England
te17@leicester.ac.uk
https://orcid.org/0000-0002-4470-5868

Jakob T. Spooner
Department of Informatics, University of Leicester, Leicester, England
jts21@leicester.ac.uk
https://orcid.org/0000-0003-3816-6308

Abstract

A temporal graph can be viewed as a sequence of static graphs indexed by discrete time steps. The vertex set of each graph in the sequence remains the same; however, the edge sets are allowed to differ. A natural problem on temporal graphs is the Temporal Exploration problem (TEXP): given, as input, a temporal graph $\mathcal{G}$ of order n, we are tasked with computing an *exploration schedule* (i.e., a temporal walk that visits all vertices in $\mathcal{G}$), such that the time step at which the walk arrives at the last unvisited vertex is minimised (we refer to this time step as the *arrival time*). It can be easily shown that general temporal graphs admit exploration schedules with arrival time no greater than $O(n^2)$. Moreover, it has been shown previously that there exists an infinite family of temporal graphs for which any exploration schedule has arrival time $\Omega(n^2)$, making these bounds tight for general TEXP instances. We consider restricted instances of TEXP, in which the temporal graph given as input is, in every time step, of maximum degree d; we show an $O(\frac{n^2}{\log n})$ bound on the arrival time when d is constant, and an $O(d \log d \cdot \frac{n^2}{\log n})$ bound when d is given as some function of n.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases temporal graph exploration, algorithmic graph theory, *NP*-complete problem

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.36

1 Introduction

A natural generalisation of the static, undirected graph is one by which a notion of time is introduced to the edge set. Such a generalisation provides a flexible tool for the modelling of problems/scenarios of a dynamic nature; particularly those which incorporate some temporal aspect into their structure. Informally, a *temporal graph* can be viewed as a graph whose edges are allowed to change over time. Time, herein, refers to an interval comprised of discrete time steps; the model we consider sees a temporal graph as an ordered sequence of static graphs indexed by the steps in this time interval (we call the number of steps contained in the interval the *lifetime* of the graph). In each graph of the sequence, the edge set may differ, whilst the vertex set remains constant. Additionally, we require that the edges that appear in each time step originate from some pre-specified static graph, which we call the *underlying graph*. It is this potential for the edges connecting the vertices in the graph to change over time that allows us to model problems, scenarios and systems in which the relationships between entities can evolve. Various "dynamic graph" models exist; they, and the problems defined on them, have been explored in a number of other studies. For an

overview we refer the reader to [9] and [4]. Since the addition of an element of dynamicity fundamentally makes temporal graphs different from their static predecessors, it is clear that the development of new techniques for their analysis (and for the analysis of problems defined upon them) is required. In this paper, we take an algorithmic standpoint, and consider a restricted case of the problem of computing a walk around a temporal graph that visits all vertices by the earliest time step possible.

1.1 Contribution

This paper considers the problem of Temporal Exploration, or TEXP (defined originally in [10] by Michail and Spirakis), which asks that, given a temporal graph $\mathcal{G}$, we compute an *exploration schedule* (i.e., a walk visiting each vertex in $\mathcal{G}$ at least once) such that the time step at which the last unvisited vertex is reached is minimal amongst all possible schedules (we call this time step the *arrival time*). In [10], Michail and Spirakis gave an $O(dn)$ upper bound on the arrival time of exploration schedules for arbitrary temporal graphs, where d is the *dynamic diameter* of the graph. Erlebach et al. [5] observed that, in general temporal graphs that are connected in every step, d can be at most $n-1$, implying an $O(n^2)$ bound on arrival times in the worst case. Moreover, they provide an explicit construction of an infinite family of graphs for which any exploration schedule has arrival time $\Omega(n^2)$, making these bounds tight for general instances of TEXP. We note that this construction is, as far as we are aware, the only one that has been shown to require this many steps for exploration; this, coupled with the fact that all graphs within this family contain a vertex of high degree $(n-1)$ in each time step, motivates our examination of the problem when restricted to instances in which the temporal graph is of maximum degree d in every time step. Further, provided in [5] is a construction of a degree-bounded family of graphs (with planar underlying graphs), such that each graph in the family is of maximum degree 2 in every time step. They prove $\Omega(n \log n)$ arrival times of any exploration schedule for any graph in this family. No better bound than the general $O(n^2)$ bound is known for the arrival time of exploration schedules for degree-bounded temporal graphs. We make the first step towards narrowing this large gap by showing that, for any graph contained within the class of degree-bounded temporal graphs, one can guarantee that there exists an exploration schedule, W, such that the arrival time of W is $O(d \log d \cdot \frac{n^2}{\log n})$ when d is given as some function of n, and $O(\frac{n^2}{\log n})$ when d is constant.

1.2 Related work

A number of previous studies have considered how standard results, problems and definitions are affected when viewed in the context of a variety of dynamic graph models. For example, it was found by Berman in [1] that the vertex variant of Menger's theorem does not hold when applied to a particular model of dynamic graphs (termed *scheduled networks*), in which each edge is assigned both an arrival and departure time. On the other hand, Kempe et al. showed in [7] that, under this same model, there is a class of temporal graphs for which the vertex variant of Menger's theorem does hold – they show this by means of a forbidden minor characterisation.

As well as their previously mentioned results, Kempe et al. [7] showed that it is *NP*-complete to decide whether or not there exist two vertex-disjoint, *time-respecting* paths (i.e., each edge in the path is traversed during a time step that is strictly greater than the previous edge) between a given source and sink. Under a similar model, Bui-Xuan et al. [3] consider the problems of computing an s-t path in a temporal graph that is *shortest* (minimal number

of edges), *foremost* (arrives at t at the earliest possible time step) or *fastest* (the time spent between traversing the first and last edges in the path is minimal), showing that there are a number of natural path parameters one might wish to optimise when considering dynamic graphs.

In [8], Mertzios et al. consider a temporal graph model in which each edge, e, is labelled with the set of time steps during which e appears in the graph. They specify a polynomial-time algorithm for the problem of computing a foremost path between two given vertices, and, complementing the work in [1] and [7], present a temporal analogue of Menger's theorem which holds for general graphs adhering to their model.

Brodén et al. [2] study a temporal analogue of the TSP problem, in which the graph is a complete graph in every step, and a cost belonging to the set $\{1, 2\}$ is assigned to each edge; additionally, these costs can change in each time step. They assume that the costs of the edges can change at most k times over the course of the graph's lifetime, and manage to provide a polynomial-time approximation algorithm with approximation ratio $2 - \frac{2}{3k}$. In [10], Michail and Spirakis also considered this model, showing the general problem to be *APX*-hard, whilst improving on the results of [2] with a $(1.7 + \varepsilon)$-approximation algorithm.

Under the same model as [8], Michail and Spirakis [10] formally introduced the problem of Temporal Exploration, showing that the general problem of deciding whether a temporal graph admits any valid exploration schedule is *NP*-complete if the graph is not assumed to be connected in each time step. They suggested making the assumption that the graph is connected in every step. Erlebach et al. [5] further studied the TEXP problem under this assumption. In addition to their previously mentioned results, they obtained an $O(n^{1-\varepsilon})$-inapproximability result, for any $\varepsilon > 0$, ruling out the possibility of any constant-factor approximation algorithms. Additionally, they considered the problem of TEXP when the input graph is subject to various structural restrictions: amongst other things, they proved $O(n)$ arrival times for temporal graphs in which the underlying graph is a cycle; $O(n^{1.5}k^2 \log n)$ arrival times for underlying graphs of treewidth k; and $O(n \log^3 n)$ arrival times for underlying graphs that are $2 \times n$ grids. From a different angle, they also considered a model in which the edges of the graph are present in each step with a particular probability, or appear with a certain regularity.

In [6], Fluschnik et al. considered the *NP*-hard problem of Temporal (s, z)-Separation, in which we are are asked to separate two vertices, s and z, in a given temporal graph, by removing from it a minimal number of vertices. Similarly to [5], they showed that there are a number of restricted graph classes for which Temporal (s, z)-Separation becomes easier, and a number for which the problem remains hard to solve; for example they showed that the problem becomes fixed-parameter tractable when parameterised by $k + l$, where k is the size of a solution, and l is the longest temporal path in the input graph. Negatively, they showed that for graphs in which at most one edge is present in every time step, the problem remains *NP*-hard.

2 Preliminaries

In this section, we introduce those definitions key to formulating the general problem of Temporal Exploration.

▶ **Definition 1** (Temporal graph)**.** We represent a temporal graph, $\mathcal{G}$, with *underlying graph*, $G = (V, E)$, using an ordered sequence of static graphs: $\mathcal{G} = \langle G_1, G_2, ..., G_\tau \rangle$. Further, we let the set $V(\mathcal{G}) = V$ and $|V| = n$. The subscripts $i \in \{1, 2, ..., \tau\}$ indexing the graphs in the

sequence are the discrete *time steps* 1 to τ, where τ is known as the *lifetime* of $\mathcal{G}$. Each G_i represents the structure of $\mathcal{G}$ in time step i. More precisely, $G_i = (V, E_i)$ is a subgraph of G; in particular, $V(G_i) = V(G)$, $E_i \subseteq E$, for all $1 \leq i \leq \tau$.

▶ **Definition 2** (Temporal walk)**.** A temporal walk W through a temporal graph $\mathcal{G}$ is given as an alternating sequence of vertices and edge-time pairs,

$$W = v_0, (e_0, i_0), v_1, (e_1, i_1), v_2, ..., v_{k-1}, (e_{k-1}, i_{k-1}), v_k,$$

that starts at vertex v_0 and ends at vertex v_k. Additionally, we require that $i_0 < i_1 < ... < i_{k-1}$, so that an agent following W can traverse at most one edge per time step. Each edge-time pair, (e_j, i_j), denotes the traversal of edge $e_j = \{v_j, v_{j+1}\}$ at timestep i_j. For such a traversal to be possible, e_j must be present in graph G_{i_j}, i.e. $e_j \in E_{i_j}$. We say that a walk, W, *departs* at time $i_0 = \delta(W)$ and *arrives* at time $i_{k-1} + 1 = \alpha(W)$, and may refer to the time steps $\delta(W)$ and $\alpha(W)$ as the *departure* and *arrival* times of W, respectively. Finally, let $t_0 \leq \delta(W)$ be an arbitrary time step, and assume that an agent waits at vertex v_0 for $\delta(W) - t_0$ time steps before traversing edge e_0; we refer to the difference, $|W| = \alpha(W) - t_0$, as the *duration* of W.

The following result is implied by Lemma 1 in [5]. It provides an upper bound on the number of time steps it might take to reach one vertex from another in a temporal graph that is connected in every time step.

▶ **Lemma 3** (Erlebach, Hoffmann and Kammer [5])**.** *Let $\mathcal{G}$ be a temporal graph with vertex set V, and assume $\mathcal{G}$ is connected in each step. Then an agent situated at any vertex $u \in V$ at any time $t \leq \tau - n$ can reach any other vertex $v \in V$ in at most $|V| - 1 = n - 1$ steps, i.e., by time step $t + n - 1$.*

▶ **Definition 4** (Exploration schedule)**.** We say that a walk, W, is an *exploration schedule* if, for all $v \in V(\mathcal{G})$, there exists some $v_i \in W$ such that $v = v_i$. Additionally, W is said to be *foremost* if it reaches the n-th unique vertex $v \in V(\mathcal{G})$ at time $\alpha(W)$, and if there exists no other temporal walk W', such that $\alpha(W') < \alpha(W)$.

▶ **Definition 5** (Temporal walk concatenation)**.** We define two temporal walks, W and W', to be *compatible under concatenation* if the departure time of W' is greater than or equal to the arrival time of W (i.e., $\delta(W') \geq \alpha(W)$), and W' departs from the same vertex at which W arrives. The result of their concatenation is the walk obtained by following W, then following W' directly after.

▶ **Problem** (Temporal Exploration)**.** An instance of the general Temporal Exploration (TEXP) problem is given as a pair $(\mathcal{G}, s)$, where $\mathcal{G} = \langle G_1, G_2, ..., G_\tau \rangle$ is an arbitrary temporal graph with lifetime $\tau \geq |V(\mathcal{G})|^2 = n^2$ (in order to ensure that there exist feasible solutions for any instance), and $s \in V(\mathcal{G})$ is a start vertex. The problem then asks for a temporal walk, W, such that W is an exploration schedule, W is foremost, and W departs from vertex s. We make the additional assumption that the graph is connected in each step; without this it could happen that there exists no valid exploration schedule.

We note that since (as part of any instance of the TEXP problem) we are given a temporal graph, $\mathcal{G} = \langle G_1, G_2, ..., G_\tau \rangle$, any candidate algorithm for the problem knows, in advance, the structure of the graph in each step of its lifetime (and therefore knows the dynamic structure of the temporal graph's edge set in advance). The following section introduces those definitions specific to the proof of our main result.

3 Exploring degree-bounded temporal graphs

We now turn our attention to restricted instances of the Temporal Exploration problem, in which the input graph is of bounded degree in each time step. We proceed by introducing those definitions central to the proof of our main result.

▶ **Definition 6** (Temporal graph of bounded degree)**.** Let $\mathcal{G}^d = \langle G_1^d, G_2^d, ..., G_\tau^d \rangle$ be a temporal graph of order $|V(\mathcal{G}^d)| = n$, and *lifetime* τ. It is said that the degree of $\mathcal{G}^d$ is bounded by d if, for all i, $\max_{v \in V(G_i^d)} \deg_{G_i^d}(v) \leq d$. We note that d may be given as a constant, or as some function of n.

(Note that we do not place any restriction on the underlying graph of $\mathcal{G}^d$.) Consider now a partitioning of the vertex set, $V(\mathcal{G}^d)$, of a degree-bounded temporal graph, $\mathcal{G}^d$, into distinct parts T and L, such that $T \cup L = V(\mathcal{G}^d)$. We refer to the set T as the *terminal* set, containing $|T| = k$ terminal vertices $u \in T$, and L as the *leftover* set, containing the $|L| = n - k$ leftover vertices $v \in L$. The set T of terminals will contain, initially, the set of all unvisited vertices in $\mathcal{G}^d$, whilst the set L will contain the specified start vertex, s. Our aim is to form $\Omega(\frac{k}{d})$ disjoint ordered pairs of terminal vertices, (u, u'), such that there is a temporal walk through $\mathcal{G}^d$ that departs from u, arrives at u', and has duration no longer than $O(\frac{dn}{k})$. In doing so, we obtain a subset $T' \subseteq T$ of terminals which are arrived at by such a walk; by setting $T = T'$, moving all terminals not reachable by such a walk into L, and forming pairs amongst those $u \in T$ again, we obtain a new collection of walks, each of which can be concatenated with exactly one of the walks obtained from the previous application. Repeated application of this process will form the basis of our overall exploration approach, eventually enabling us to explore $\Theta(\log_d n)$ vertices in $O(dn)$ steps. The following definitions will prove useful in showing this – in each of them we consider an arbitrary temporal graph, $\mathcal{G}$, during a time period starting at time step t, and ending at time step t':

▶ **Definition 7** (Reachable set)**.** The reachable set of a vertex, $x \in V(\mathcal{G})$, is the set $R_x(t, t') = \{y \in V(\mathcal{G}) \mid$ there exists a walk from x to y starting at time l and ending at time l', with $t \leq l \leq l' \leq t'\}$. If t and t' are made clear, or are deducible from the context, we simply write R_x, rather than $R_x(t, t')$.

▶ **Definition 8** (Reachable pair)**.** We call an ordered pair of terminal vertices, $(u, u') \in T \times T$, a *reachable pair* if $u' \in R_u(t, t')$ (with $1 \leq t \leq t' \leq \tau$) and $u \neq u'$. When it is clear from the context, we may simply refer to a reachable pair as a *pair*. We also say that a reachable pair is *formed* in the step $t' - 1$, in which u' is added to $R_u(t, t')$.

▶ **Definition 9** (Home set)**.** We define a *home set* of a leftover vertex, $v \in L$, to be a set $H_v(t, t')$ of terminal vertices such that, for any terminal vertex u, the condition $u \in H_v(t, t')$ implies the following: $v \in R_u(t, t')$, and u does not already belong to a reachable pair. As such, there exist temporal walks in $\mathcal{G}$ departing from all $u \in H_v(t, t')$ and arriving at v. Again, we write H_v rather than $H_v(t, t')$ when t and t' are clear from the context.

Note that Definition 9 does not define home sets in a unique way. It only requires that $u \in H_v(t, t')$ implies $v \in R_u(t, t')$, but there is no requirement that all vertices u that satisfy $v \in R_u(t, t')$ are included in the home set. We will specify how to construct home sets in a certain way, and the home sets resulting from our construction will have the additional property that they contain at most two vertices. The purpose of home sets and the details of their construction will become clear in the proof of Lemma 13.

▶ **Definition 10** (Spread of a terminal vertex)**.** We refer to the number of home sets that a particular terminal vertex $u \in T$ belongs to as the *spread* of u.

As previously discussed, we first wish to show that in a degree-bounded temporal graph, $\mathcal{G}^d$, enough disjoint reachable pairs are formed during any period of $O(\frac{dn}{k})$ time steps (here we take k to be the number of terminals contained in T at the start of the time period we are considering). To achieve this, we introduce the following potential function, which tracks the number of terminals that each leftover vertex has contained within its respective home set. By first showing that we can increase the value of this potential function by a large enough amount in each of the $O(\frac{dn}{k})$ considered steps, we will be able to prove that $\Omega(\frac{k}{d})$ disjoint reachable pairs can be found during those same steps.

▶ **Definition 11** (Potential function, ϕ)**.** Consider an arbitrary time step t, and let i be the current time step ($1 \leq t \leq i \leq \tau$). Further, let L_0, L_1, L_2 be the sets of leftover vertices $v \in L$, such that $|H_v(t,i)| = 0$, $|H_v(t,i)| = 1$, and $|H_v(t,i)| = 2$, respectively. In other words, L_0, L_1 and L_2 are the sets of leftover vertices that have 0, 1 and 2 terminals contained in their home sets at time i. Clearly, $L = L_0 \cup L_1 \cup L_2$ at any time, since the home sets we consider grow to a size no larger than 2. We denote by P_v^i the *potential value* of vertex v at time i, and define it for all i as follows:

$$P_v^i = \begin{cases} 1, & \text{if } v \in L_0 \text{ (at time } i) \\ 2, & \text{if } v \in L_1 \text{ (at time } i) \\ 3, & \text{if } v \in L_2 \text{ (at time } i). \end{cases}$$

Given this, we introduce our potential function, ϕ, taking as argument a bounded-degree temporal graph, $\mathcal{G}^d$:

$$\phi_t^i(\mathcal{G}^d) = \sum_{v \in L} P_v^i,$$

so that $\phi_t^i(\mathcal{G}^d)$ is the sum of the values, P_v^i, during time step i ($t \leq i \leq \tau$), given that we began tracking the value of ϕ at time t. When t is clear from the context, we may refer to $\phi_t^i(\mathcal{G}^d)$ at arbitrary i ($t \leq i \leq \tau$) as $\phi^i(\mathcal{G}^d)$ or, if i is clear from the context, as $\phi(\mathcal{G}^d)$ or ϕ.

▶ **Observation 12.** If we assume that we begin tracking the value of our potential function from some time t, such that no $v \in L$ belongs to the reachable set of any $u \in T$ at time t, then initially, $\phi^t(\mathcal{G}^d) = |L|$. Further to this, $P_v^i \leq 3$ for any $v \in L$, and $|L| \leq |V(\mathcal{G}^d)|$. Therefore, $\phi^i(\mathcal{G}^d) \leq 3|V(\mathcal{G}^d)| = 3n$ for any $t \leq i \leq \tau$.

We note that when a reachable pair (u, u') is formed, then both $u, u' \in T$ are no longer considered as candidates for further reachable pairs (since we require our pairs to be disjoint). Therefore, we remove u and u' from T and place them into L. As a result, it is possible for the number of leftover vertices to grow over the course of time, allowing for larger potential values in future time steps. Moreover, in order to increase our potential in any given step, we must be able to find terminal vertices that can be added to the home set of a leftover vertex in that step.

By Definition 9, $u \in H_v$ implies that u does not already belong to any reachable pair. It therefore follows that the forming of a pair in $\mathcal{G}^d$ can also cause the potential to drop, since we are required to remove u and u' from all home sets that they are contained within. Whilst some decrease in potential is inevitable, it is important that we ensure that the decrease is not too large in any single step. If the home sets of many leftover vertices contain the same terminal, and that terminal goes on to form a pair, then it will be removed from all of these home sets, possibly generating a large drop in potential. If this type of behaviour occurs too often, it might happen that we are unable to obtain a large enough amount of potential in total, and as a result, not be able to form the required amount of reachable pairs. This issue is fully addressed by Lemma 13 in the following subsection.

3.1 Exploration method

Following the discussion above, all is in place to fully describe our overall approach to computing exploration schedules (in degree-bounded temporal graphs) that are guaranteed to have arrival time $O(d \log d \cdot \frac{n^2}{\log n})$. We begin by showing that it is possible to obtain a potential increase of $\frac{k-2p}{2d}$ in any given time step, where p is the number of pairs that have already been formed between vertices in T. For the following, assume that we have fixed a time step t, and that we consider $\mathcal{G}^d$ from time t onwards.

▶ **Lemma 13.** *Consider a degree-bounded temporal graph, $\mathcal{G}^d$, in an arbitrary timestep i, whose vertex set has been partitioned into sets T (terminals) and L (leftovers), which initially contained $|T| = k$ and $|L| = n - k$ vertices, respectively. Additionally, assume that $p \leq \frac{k-2}{2}$ reachable pairs of terminal vertices have been formed already (including those pairs formed in step i). Then, it is possible to obtain a potential increase of at least $\frac{k-2p}{2d}$ in time step i.*

Proof. Observe that, since there are already p pairs formed amongst the terminals $u \in T$, there are exactly $k - 2p$ terminals that remain in T, i.e. $|T| = k - 2p$ (since any two terminals forming a pair become leftover vertices once the pair is formed). As $p \leq \frac{k-2}{2}$ implies $k - 2p \geq 2$, we know that there are still at least 2 terminals. We wish to find at least $\frac{k-2p}{2d}$ disjoint paths, such that each path has terminals as its endpoints, and that every other vertex in the path is a leftover vertex (we will refer to such a path as a *terminal path*). Let G_i^d denote the form of $\mathcal{G}^d$ during the i-th time step; we proceed by computing a spanning tree S_i of G_i^d, selecting an arbitrary leaf, r, as S_i's root, and forming paths in a bottom-up fashion via a greedy procedure. More specifically, we consider each vertex, $x \in S_i$, in reverse level order, so that we first examine those vertices that are furthest away from r, followed by those vertices second furthest away from r, and so on, until every x has been processed. On examining a vertex x, we consider the subtree of S_i rooted at x: if it contains 2 or more terminals that do not already belong to a path, we arbitrarily select two and take the path joining them in S_i to be one of our terminal paths, discarding the remaining terminals in that subtree. We claim that in this way, we use exactly 2 terminals for each path we form, and discard at most $d - 2$ terminals whenever we form a single path. To see this, observe that each vertex in S_i has at most $d - 1$ children (since r is a leaf), and consider the situation in which the vertex we are currently processing, x, is a terminal of degree d. If each of the subtrees rooted at x's children contains exactly one terminal, then we will only be able to form one path, which must have x and one of the other $d - 1$ terminals (each of which lies alone in one of the subtrees rooted at x's children) as its endpoints – the remaining $d - 2$ terminals will be discarded. This follows from the fact that if more than one terminal lay in the subtree rooted at any of x's children, then they must already belong to a path; otherwise they would have already been discarded, as per our procedure. Therefore, on forming any path in S_i, we "use up" at most d terminals (2 terminals for the path, and at most $d - 2$ terminals are discarded). Observe now that, on processing r, it may also happen that there is a single unmatched terminal, y, in the subtree rooted at r; in this case, y will be discarded. As a result, it follows that we are able to find at least $\frac{k-2p-1}{d}$ disjoint terminal paths in S_i. Moreover, $\frac{k-2p-1}{d} \geq \frac{k-2p}{2d}$ for $d > 0$ and $p \leq \frac{k-2}{2}$, which can be easily checked. Since $k - 2p \geq 2$ precisely when $p \leq \frac{k-2}{2}$, it follows that we are able to form at least $\frac{k-2p}{2d}$ disjoint terminal paths in S_i, as required.

Now, given a set of terminal paths obtained by following the aforedescribed method, we wish to show that, per each path, we can obtain a $+1$ increase in potential. We require a way of doing so that ensures that not only are we able to increase the potential by a large enough

amount in every step, but that we are able to ensure the drop in potential experienced in any particular time step is limited. Specifically, we require a procedure with the following properties:

1. We can obtain a potential increase of +1 within any given terminal path.
2. For a single execution of our procedure on a given terminal path, we want the spread of exactly one of the path's endpoints to increase by 1, and the spread of the terminal at the opposite end of the path to remain the same.

Property **(1)** ensures that we are able to guarantee a potential increase of $\frac{k-2p}{2d}$ in each time step. Property **(2)** limits the number of home sets that any terminal vertex $u \in T$ can be added to in any particular step to 1. To this end, we specify now our procedure, demonstrating that both properties **(1)** and **(2)** are satisfied by the actions performed within each individual case.

▶ **Procedure** (Obtain-Potential)**.** Consider an arbitrary time step, t, from which we began tracking the value of $\phi(\mathcal{G}^d)$. The input to the procedure is a terminal path, $Q_i \subseteq S_i$ (we omit the i from this notation from here onwards), obtained by applying the aforedescribed greedy procedure to $\mathcal{G}^d$ in step $i \geq t$ (i.e., by applying the greedy procedure to the graph G_i^d). Let u and u' be the endpoints of Q. We proceed as follows: arbitrarily select one of u or u' (for argument's sake we select u), and begin examining the set $H_{v_j}(t,i)$ (again, we will omit the arguments from this notation) for every $v_j \in Q$ in the order in which they appear in Q (from u to u'). Let the x-th and last leftover vertex that we examine where $H_{v_j} = \{u\}$ be known as v_x; the vertex we examine next will be known as v_{x+1}. We note that since, by our earlier assumption, all pairs that will form in step i have already been formed, the existence of such a v_x is guaranteed in all but one case. If u is adjacent to the vertex $v_1 \in Q$ such that $H_{v_1} \supseteq \{u^*\}$ with $u^* \neq u$, then clearly a pair is formed; a contradiction, since the greedy procedure for forming terminal paths does not consider terminals that already belong to pairs. Similarly, if $v_x = v_{|Q|-1}$, then $H_{v_{|Q|-1}} = \{u\}$ and u and u' form a pair in that step; again, a contradiction. Thus, the exceptional case, in which there is no such v_x, occurs when $H_{v_1} = \emptyset$: here, we can instantly obtain our potential increase for Q by adding u to H_{v_1}, whilst still satisfying properties **(1)** and **(2)**. With these exceptions dealt with, we now distinguish between two main cases:

- **(i) Case 1: $H_{v_x} \cap H_{v_{x+1}} = \emptyset$.** We distinguish between two subcases:
  - **(i) Case 1.1: $H_{v_{x+1}} \neq \emptyset$.** Select an arbitrary u^* in $H_{v_{x+1}}$ and add u^* to H_{v_x}, giving our +1 potential increase (satisfying property **(1)**). We note that this is possible since $u^* \in H_{v_{x+1}}$ implies that $v_{x+1} \in R_{u^*}$, in which case an agent could move from u^* to v_{x+1}, and from v_{x+1} to v_x. It is therefore valid to select u^* from the home set of a vertex adjacent to v_x and add it to v_x's home set since, by definition, v_x would be added to R_{u^*} in that step regardless.
    In the event that $u^* \notin \{u, u'\}$, remove u^* from $H_{v_{x+1}}$, and add u to $H_{v_{x+1}}$; this ensures that the potential associated with v_{x+1} does not decrease, whilst additionally ensuring that the spread of u^* does not increase with respect to Q (satisfying property **(2)**).
  - **(ii) Case 1.2: $H_{v_{x+1}} = \emptyset$.** In this case, we add u to $H_{v_{x+1}}$, giving our +1 potential increase (property **(1)**), whilst ensuring that only u is added to the home set of exactly one leftover vertex in Q (property **(2)**).
- **(ii) Case 2: $H_{v_x} \cap H_{v_{x+1}} \neq \emptyset$.** Again, we distinguish between two subcases:
  - **(i) Case 2.1: $u' \in H_{v_{x+1}}$.** Since $u' \in H_{v_{x+1}}$, we can simply add u' to H_{v_x} and we are done – this gets us our required +1 potential increase (property **(1)**), whilst

also satisfying property **(2)**, since the only terminal that is added to a home set within Q is one of Q's endpoints, u'.

(ii) **Case 2.2: $u' \notin H_{v_{x+1}}$.** Select an arbitrary $u^* \in H_{v_{x+1}}$ (with $u^* \neq u, u'$), add u^* to H_{v_x}, and remove u^* from $H_{v_{x+1}}$. Now, continue following Q in the same direction, repeating the above process some $y-1$ times, until in the y-th iteration we examine a vertex v_{x+y+1}, such that $u \notin H_{v_{x+y+1}}$. More generally, in the j-th iteration, we check the set H_{x+j+1}, select from it an arbitrary $u^* \neq u$, add u^* to $H_{v_{x+j}}$, and remove u^* from $H_{v_{x+j+1}}$. By removing u^* from $H_{v_{x+j+1}}$, we ensure that the spread of u^* does not increase with respect to Q, satisfying property **(2)** for all iterations 1 through $y-1$ (notice that in each of these iterations, our potential value stays the same, since whenever we add a terminal to one leftover vertex's home set, we remove that terminal from another home set). If it happens that $H_{v_{x+j+1}} = \{u\}$ (i.e., we cannot select $u^* \neq u$), then we do not add any u^* to $H_{v_{x+j}}$ in that iteration; clearly property **(2)** is still satisfied in this situation. Once we begin the y-th iteration, if it happens that the set $H_{v_{x+y+1}} = \emptyset$, we simply add u to $H_{v_{x+y+1}}$ and we are done – in this case, both properties are trivially satisfied. Otherwise, select an arbitrary u^* from $H_{v_{x+y+1}}$, add u^* to $H_{v_{x+y}}$ (increasing the potential value of v_{x+y}), and remove u^* from $H_{v_{x+y+1}}$. Finally, add u to $H_{v_{x+y+1}}$; this uses the fact that $H_{v_{x+y}}$ is guaranteed to contain terminal u, and so we replace u^* with u, ensuring that the potential of vertex v_{x+y+1} does not decrease, whilst still ensuring that property **(2)** is satisfied, since only vertex u's spread has increased by exactly 1. It is this final step that ensures property **(1)** is satisfied, since we increased the potential value of v_{x+y} in this iteration, but did not decrease the potential value of v_{x+y+1}.

The lemma follows by applying the greedy procedure (for forming $\frac{k-2p}{2d}$ disjoint terminal paths) in G_i^d, and supplying each path as input to the Obtain-Potential procedure. ◀

▶ **Lemma 14.** *Let $\mathcal{G}^d$ be a degree-bounded temporal graph with a vertex set partitioned into parts T and L. Let t be the time at which tracking of the potential function began (so that $\phi^t(\mathcal{G}^d) = |L|$), and let i be the current time step. Then, the forming of a single reachable pair of terminal vertices, (u, u'), can cause the potential value to drop by at most $2l$, where $l = i - t$ is the number of steps that passed since we began tracking the value of $\phi(\mathcal{G}^d)$.*

Proof. Consider the specification of our Obtain-Potential procedure and note that, during any time step i (with $t \leq i \leq \tau$), each terminal vertex can belong to at most one disjoint terminal path, and that exactly one of the vertices in $\{u, u'\}$ (that form the endpoints of each terminal path) has its spread increase by 1. It is clear that when a reachable pair of terminal vertices (u, u') is formed, the worst case scenario is the following: both u and u' were added to the home set of a single leftover vertex in each of the l steps that have passed since time t. The lemma follows by observing that this scenario will result in an overall drop in potential of $2l$, since, by Definition 9, u and u' will be removed from the home set of all those leftover vertices $v \in L$ for which they belong to the set $H_v(t, i)$. ◀

▶ **Lemma 15.** *Let $\mathcal{G}^d$ be a degree-bounded temporal graph with an underlying graph of order n, and let $V(\mathcal{G}^d)$ be partitioned into parts T and L, with $|T| = k$ and $|L| = n - k$. Then in $10 \cdot \frac{dn}{k}$ steps, at least $\frac{k}{20d}$ disjoint reachable pairs of terminal vertices are formed.*

Proof. Consider $\mathcal{G}^d$ from time t onwards and assume that the opposite of our claim is true, so that less than $\frac{k}{20d}$ reachable pairs are formed in the space of $10 \cdot \frac{dn}{k}$ steps (i.e. less than $\frac{k}{20d}$ reachable pairs are formed within the time range t to $t + 10 \cdot \frac{dn}{k}$). Then, by the end of the

$(t + 10 \cdot \frac{dn}{k})$-th step, there are at least $k - 2 \cdot \frac{k}{20d} = k - \frac{k}{10d} = \frac{10dk-k}{10d} = k \cdot \frac{(10d-1)}{10d} \geq 0.9k$ (for all $d \geq 1$) terminal vertices that have not yet formed a pair with another terminal vertex. It follows then, by Lemma 13, that the value of $\phi(\mathcal{G}^d)$ increases by at least $\frac{0.9k}{2d} \cdot \frac{10dn}{k} = \frac{9kdn}{2dk} = \frac{9n}{2} = 4.5n$ over those $10 \cdot \frac{dn}{k}$ steps. By Lemma 14, over the same period, the potential can decrease by at most $\frac{k}{20d} \cdot 2 \cdot \frac{10dn}{k} = \frac{k}{20d} \cdot \frac{20dn}{k} = \frac{20dkn}{20dk} = n$, since, by our earlier assumption, fewer than $\frac{k}{20d}$ pairs are formed. From this, it follows that the potential at the end of those $10 \cdot \frac{dn}{k}$ steps is at least $4.5n - n = 3.5n$. But this is a contradiction since, by Observation 12, $\phi^i(\mathcal{G}^d) \leq 3n$ for any $t \leq i \leq \tau$; the lemma follows. ◀

▶ **Lemma 16.** *Consider a degree-bounded temporal graph, $\mathcal{G}^d$, in an arbitrary time step, t, and let the vertices in $V(\mathcal{G}^d)$ be divided into parts T and L, with $|T| = k$ and $|L| = n - k$ at the beginning of time t. Then, there exists at least one vertex $v \in T$ such that an agent positioned at v at the start of time t can explore $\Theta(\log_d k)$ terminal vertices in $O(dn)$ steps.*

Proof. By application of Lemma 15 to part T, we are able to form at least $\frac{k}{20d}$ disjoint reachable pairs of terminal vertices in the space of $10 \cdot \frac{dn}{k}$ time steps. As a result, we obtain a set of $\frac{k}{20d}$ temporal walks, each of which has exactly one such reachable pair constituting its endpoints, with no walk taking any longer than $10 \cdot \frac{dn}{k}$ steps. At the end of these $\frac{k}{20d}$ walks are exactly $\frac{k}{20d}$ unique (since the pairs are disjoint) terminal vertices. We proceed by taking this smaller set of terminals as our new T and reapplying Lemma 15 at time $t + 10 \cdot \frac{dn}{k}$; as a result, we obtain $\frac{k/20d}{20d} = k/(20d)^2$ new walks, each of which has endpoints that form a reachable pair, and each of which takes no longer than $10 \cdot \frac{dn}{k/20d}$ steps. Notice now that since, via our second application of Lemma 15, we formed pairs amongst only those terminals that were reachable via one of the walks obtained from our initial application of Lemma 15, there are now at least $k/(20d)^2$ walks of length at most $\frac{10dn}{k} + \frac{10dn}{k/(20d)}$, each of which visits three terminal vertices; in other words, we are able to concatenate some walk obtained by our first application of Lemma 15, with some walk obtained by our second application, to construct a walk that visits three terminals. In this fashion, we claim that we are able to construct a walk that visits $\Theta(\log_d k)$ terminals.

Generally speaking, the i-th application of Lemma 15 produces at least $k/(20d)^i$ disjoint pairs of reachable unvisited vertices, in each of which one vertex can reach the other via a walk of length at most $10 \cdot (dn)/(k/(20d)^{i-1})$. Since each time we reapply Lemma 15, it is applied only to those terminals that are reachable via a walk computed by the previous application, it follows that there must be a sequence of i walks, one resulting from each application of Lemma 15, that can be concatenated (in the order in which they were produced) to form a walk that visits exactly $i + 1$ terminals (two terminals are visited by the first computed walk, then an additional terminal is explored by the walk obtained by each successive application). Observe that we are able to repeatedly apply Lemma 15 to T exactly $\lfloor \log_{20d} k \rfloor$ times until no more pairs can be formed in T. By taking the i-th application of Lemma 15 to be the $\lfloor \log_{20d} k \rfloor$-th such application, it is clear that an agent following a walk constructed via the discussed concatenation method can explore $\lfloor \log_{20d} k \rfloor + 1 = \Theta(\log_d k)$ terminals. All that remains to be shown is that the length of any temporal walk, W, constructed in this way is of duration at most $O(dn)$. By our earlier discussion, a reachable pair of unvisited vertices, (u, u'), found as a result of the i-th application of Lemma 15, are separated by a walk with duration no longer than $10 \cdot dn/(k/(20d)^{i-1})$. Given that we apply Lemma 15 $\lfloor \log_{20d} k \rfloor$ times in order to obtain each of the shorter walks that are then concatenated to obtain W, we derive the following summation to bound the overall duration of W from above:

$$\sum_{i=0}^{\lfloor \log_{20d} k \rfloor} \left(10 \cdot \frac{dn}{k/(20d)^i}\right) = \frac{10dn}{k} \cdot \sum_{i=0}^{\lfloor \log_{20d} k \rfloor} (20d)^i \leq \frac{10dn}{k} \cdot 2k = O(dn),$$

as required. ◀

▶ **Theorem 17.** *Let $\mathcal{G}^d$ be a degree-bounded temporal graph of order $|V(\mathcal{G}^d)| = n$. Then there exists an exploration schedule, W_{exp}, starting from a given vertex, s, such that W_{exp}'s arrival time, $\alpha(W_{exp})$, is $O(d \log d \cdot \frac{n^2}{\log n})$.*

Proof. Let the current time step be $t = 1$, and let $V(\mathcal{G}^d)$ be partitioned into sets T and L, with $|T| = k$ and $|L| = n - k$ at time t (for $k = n - 1$, since s is already explored). Initially, we wish to explore $\Theta(\log_d k)$ vertices during the first $O(dn)$ steps. To achieve this, we begin by constructing a temporal walk, W_1, with departure time $\delta(W_1) = t = 1$. Let W_1 be the product of concatenating two walks, X_1 and Y_1, in that order. We note that it is not guaranteed that s will be at the start of any walk computed via our first application of Lemma 16: therefore, we set X_1 to be of duration n, and apply Lemma 16 to $\mathcal{G}^d$ at time $\delta(W_1) + |X_1| = \delta(W_1) + n$, initially setting $T = V(\mathcal{G}^d) - \{s\}$, and $L = \{s\}$, so that $T \cup L = V(\mathcal{G}^d)$. The walk resulting from that application of Lemma 16 will be known as Y_1. In this way, we ensure, by Lemma 3 (Lemma 1 in [5]), that there is enough time for an agent to move, via X_1, from s to whichever vertex Y_1 departs from. Since $|X_1| = n$, $|Y_1| = O(dn)$ (by Lemma 16), and, X_1 and Y_1 are compatible under walk concatenation, we are able to obtain the walk, W_1, of duration $|W_1| = n + O(dn) = O(dn)$, that explores $\Theta(\log_d k)$ unique vertices.

We wish to apply the above process repeatedly, until only $k \leq \frac{n}{\log_{20d} n}$ vertices remain to be explored in $\mathcal{G}^d$. This can be achieved by constructing temporal walks W_i, such that each W_i is the concatenation of two walks X_i and Y_i. Let $\alpha(W_{i-1})$ be the arrival time of walk W_{i-1}. Each X_i is a walk with duration n and departure time $\delta(X_i) = \alpha(W_{i-1})$, that departs from the last vertex of W_{i-1}, and arrives at the first vertex of Y_i. We then define Y_i to be the temporal walk, exploring $\Theta(\log_d k)$ vertices, obtained via the i-th application of Lemma 16. We perform this i-th such application at time $\delta(W_{i-1}) + |X_i| = \delta(W_{i-1}) + n$, setting

$$T = V(\mathcal{G}^d) - (\{s\} \cup V(Y_1) \cup V(Y_2) \cup \ldots \cup V(Y_{i-1})),$$

$L = V(\mathcal{G}^d) - T$ and $k = |T|$. Taking W_i to be the result of concatenating X_i and Y_i (in that order), it follows that each walk W_i departs from the last vertex of W_{i-1}, and so we can continually concatenate each W_i in the order that they are produced. The concatenation of each W_i explores an additional $\Theta(\log_d k)$ vertices; we continue this process until the number of unexplored vertices is less than or equal to $\frac{n}{\log_{20d} n}$.

In order to show now that our overall approach to exploration always produces schedules with arrival time $O(d \log d \cdot \frac{n^2}{\log n})$, we first show that $\log_{20d} k = \Theta(\log_d n)$ whenever we apply Lemma 16. As stated previously, we repeatedly apply the process discussed above until $k \leq \frac{n}{\log_{20d} n}$. This implies that whenever we apply Lemma 16, $k > \frac{n}{\log_{20d} n}$, giving that

$$\log_{20d} k > \log_{20d}(\frac{n}{\log_{20d} n}) = \log_{20d} n - \log_{20d} \log_{20d} n = \Theta(\log_d n).$$

From this, we can conclude that the duration of any W_i produced in the aforedescribed manner visits $\Theta(\log_d n)$ vertices. Since we initially explore at least $n - \frac{n}{\log_{20d} n} = O(n)$ vertices using this method, it follows that we require the concatenation of $O(n)/\Theta(\log_d n)$ W_i's, each of which is of duration $|W_i| = O(dn)$. Let the walk, resulting from the repeated concatenation of our W_i, be W^1_{exp}; it follows that W^1_{exp} explores at least $n - \frac{n}{\log_{20d} n}$ vertices in at most

$$O(dn) \cdot \frac{O(n)}{\Theta(\log_d n)} = O(d \cdot \frac{n^2}{\log_d n})$$

time steps. To deal with the remaining $\frac{n}{\log_{20d} n}$ vertices: let W_x be the last walk we produce via an application of Lemma 16, and assume it has arrival time $\alpha(W_x)$; clearly, W^1_{exp} has arrival

time $\alpha(W^1_{exp}) = \alpha(W_x) = O(d \cdot \frac{n^2}{\log_d n})$. We apply Lemma 3 (Lemma 1 in [5]), computing a walk, W^2_{exp}, which starts at time $\alpha(W^1_{exp})$, and spends a total time of $O(n)$ steps visiting each of the remaining vertices. This gives W^2_{exp} a total duration of $O(n) \cdot \frac{n}{\log_{20d} n} = O(\frac{n^2}{\log_d n})$ steps. Now, since W^1_{exp} has respective departure and arrival times $\delta(W^1_{exp}) = 1$ and $\alpha(W^1_{exp})$, and W^2_{exp} departs at time $\alpha(W^1_{exp})$ from the vertex at which W^1_{exp} ends, it follows that they are compatible under walk concatenation. The result of their concatenation is a walk, W_{exp}, such that W_{exp}'s arrival time is:

$$\alpha(W_{exp}) = |W^1_{exp}| + |W^2_{exp}| = O(d \cdot \frac{n^2}{\log_d n}) + O(\frac{n^2}{\log_d n}) = O\Big(d \cdot \frac{n^2}{\frac{\log n}{\log d}}\Big) = O\Big(d \log d \cdot \frac{n^2}{\log n}\Big),$$

and the theorem follows. ◀

From the above, it follows that whenever $d \log d = o(\log n)$, then a degree-bounded temporal graph $\mathcal{G}^d$ admits exploration schedules with arrival time $o(n^2)$. Moreover, Theorem 17 implies the following corollary:

▶ **Corollary 18.** *Let $\mathcal{G}^d$ be a temporal graph whose maximum degree in every step is at most $d \geq 2$, and d is constant. Additionally, let s be a specified start vertex. Then there exists an exploration schedule, W_{exp}, which starts at vertex s and explores all vertices in $\mathcal{G}^d$, such that the arrival time of W_{exp} is $O(\frac{n^2}{\log n})$.*

We remark that our proof is constructive and implies a polynomial-time algorithm computing an exploration schedule with arrival time $O(d \log d \cdot \frac{n^2}{\log n})$, and thus also an $O(d \log d \cdot \frac{n}{\log n})$-approximation algorithm for the TEXP problem when restricted to temporal graphs of bounded-degree.

4 Conclusion

Linear arrival times for the exploration of any static, undirected graph can be easily achieved by means of a depth-first search. The additional layer of complexity in the structure of temporal graphs, brought about by the potential for time-variance in the edge set, means that their exploration is not such a simple task: exploration schedules of general temporal graphs can require $\Theta(n^2)$ time steps.

Complementing previous results, which suggest that subjecting our input temporal graph to certain structural restrictions can improve arrival times, we have shown that by requiring the maximum degree in each time step to be bounded by d, one can guarantee arrival times of $O(d \log d \cdot \frac{n^2}{\log n})$. These results directly suggest a number of further questions: we would like to close the gap between the upper and lower bounds for exploring general degree-bounded temporal graphs – we suspect that there exists an upper bound lower than the one presented here, but it may also be the case that the current lower bound of $\Omega(n \log n)$ can be improved upon. The study of further restricted temporal graph classes, in order to establish bounds on the amount of time needed to explore them, remains an interesting question; for example, one might consider the class of temporal graphs in which only a constant number of edges are allowed to differ from each time step to the next.

A further question might ask precisely which structural properties a graph must possess in order for it to admit exploration schedules with arrival time $o(n^2)$? – it would be interesting to classify the graphs that can be explored in time strictly less than quadratic under these terms. Establishing how the computational complexity of TEXP changes whilst the problem is restricted to particular classes of graphs also presents an interesting direction; for which classes does the problem remain *NP*-hard to solve optimally, and how well can we approximate solutions for these restricted cases?

A number of other related questions also remain open. An example of one we find particularly interesting is as follows: how much quicker can general temporal graphs be explored if we allow an agent exploring the graph to make a move across two edges per time step, rather than one? We remark that the construction requiring $\Theta(n^2)$ steps to explore (given in [5]) requires only $O(n)$ steps in this two-move model. Establishing bounds for this model could provide further insight into the exploration of temporal graphs under the model considered throughout this paper.

References

1. Kenneth A. Berman. Vulnerability of scheduled networks and a generalization of Menger's theorem. *Networks*, 28(3):125–134, 1996. doi:10.1002/(SICI)1097-0037(199610)28:3<125::AID-NET1>3.0.CO;2-P.
2. Björn Brodén, Mikael Hammar, and Bengt J. Nilsson. Online and offline algorithms for the time-dependent TSP with time zones. *Algorithmica*, 39(4):299–319, 2004. doi:10.1007/s00453-004-1088-z.
3. Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *Int. J. Found. Comput. Sci.*, 14(2):267–285, 2003. doi:10.1142/S0129054103001728.
4. A. Casteigts, P. Flocchini, Quattrociocchi W., and N. Santoro. Time-varying graphs and dynamic networks. *IJPEDS*, 27(5):387–408, 2012.
5. Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. In *42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015), Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 444–455. Springer, 2015. doi:10.1007/978-3-662-47672-7_36.
6. Till Fluschnik, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Temporal graph classes: A view through temporal separators. *CoRR*, abs/1803.00882, 2018. arXiv:1803.00882.
7. David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *J. Comput. Syst. Sci.*, 64(4):820–842, 2002. doi:10.1006/jcss.2002.1829.
8. George B. Mertzios, Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Temporal network optimization subject to connectivity constraints. In *40th International Colloquium on Automata, Languages, and Programming (ICALP 2013), Part II*, volume 7966 of *Lecture Notes in Comptuer Science*, pages 657–668. Springer, 2013. doi:10.1007/978-3-642-39212-2_57.
9. Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016. doi:10.1080/15427951.2016.1177801.
10. Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theor. Comput. Sci.*, 634:1–23, 2016. doi:10.1016/j.tcs.2016.04.006.

Approximating Dominating Set on Intersection Graphs of Rectangles and L-frames

Sayan Bandyapadhyay[1]
Department of Computer Science, University of Iowa, Iowa City, USA
sayan-bandyapadhyay@uiowa.edu

Anil Maheshwari
School of Computer Science, Carleton University, Ottawa, Canada
anil@scs.carleton.ca

Saeed Mehrabi
School of Computer Science, Carleton University, Ottawa, Canada
saeed.mehrabi@carleton.ca

Subhash Suri
Department of Computer Science, UC Santa Barbara, California, USA
suri@cs.ucsb.edu

Abstract

We consider the *Minimum Dominating Set* (MDS) problem on the intersection graphs of geometric objects. Even for simple and widely-used geometric objects such as rectangles, no sub-logarithmic approximation is known for the problem and (perhaps surprisingly) the problem is NP-hard even when all the rectangles are "anchored" at a diagonal line with slope -1 (Pandit, CCCG 2017). In this paper, we first show that for any $\epsilon > 0$, there exists a $(2+\epsilon)$-approximation algorithm for the MDS problem on "diagonal-anchored" rectangles, providing the first $O(1)$-approximation for the problem on a non-trivial subclass of rectangles. It is not hard to see that the MDS problem on "diagonal-anchored" rectangles is the same as the MDS problem on "diagonal-anchored" L-*frames*: the union of a vertical and a horizontal line segment that share an endpoint. As such, we also obtain a $(2 + \epsilon)$-approximation for the problem with "diagonal-anchored" L-frames. On the other hand, we show that the problem is APX-hard in case the input L-frames intersect the diagonal, or the horizontal segments of the L-frames intersect a vertical line. However, as we show, the problem is linear-time solvable in case the L-frames intersect a vertical as well as a horizontal line. Finally, we consider the MDS problem in the so-called "edge intersection model" and obtain a number of results, answering two questions posed by Mehrabi (WAOA 2017).

2012 ACM Subject Classification Theory of computation → Algorithm design techniques, Theory of computation → Computational geometry

Keywords and phrases Minimum dominating set, Rectangles and L-frames, Approximation schemes, Local search, APX-hardness

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.37

Related Version See [3], `https://arxiv.org/abs/1803.06216`, for the full version of the paper.

Funding Research of Sayan Bandyapadhyay and Subhash Suri was supported in part by the NSF grant CCF-1525817. Research of Anil Maheshwari is supported in part by NSERC. Saeed Mehrabi is supported by a Carleton-Fields postdoctoral fellowship.

[1] The work was partially done when the author was visiting University of California, Santa Barbara.

1 Introduction

Minimum Dominating Set (MDS) is an NP-hard problem in graph theory and discrete optimization. Given a graph $G = (V, E)$, the objective of the MDS problem is to compute a minimum-size subset $V' \subseteq V$ such that every vertex not in V' is adjacent to at least one vertex in V'. For general graphs, it is known that a greedy algorithm for MDS achieves an $O(\log |V|)$-factor approximation and within a constant factor this is the best one can hope for, unless P=NP [28]. As such, the problem has been extensively studied on many subclasses of graphs, one of which is the intersection graphs of geometric objects in the plane [12, 23, 17, 16, 24, 27]. Here, each vertex of the graph is in one-to-one correspondence with a geometric object in the plane and two vertices are adjacent if and only if the corresponding objects have a non-empty intersection.

In this paper, we consider the approximability and hardness of the MDS problem on the intersection graphs of geometric objects. The MDS problem is known to admit PTASes on disk graphs [17] and the intersection graphs of non-piercing[2] objects [19]. On the other hand, it is NP-hard to obtain a $o(\log |V|)$-approximation in polynomial time for sufficiently complicated shapes, e.g. rectilinear polygons [15, 16]. However, even for simple shapes such as axis-parallel rectangles no sub-logarithmic approximation is known. The only approximation for rectangles we are aware of is due to Erlebach et al. [16] who gave an $O(c^3)$-approximation on rectangles with aspect-ratio at most c. In fact, the problem is APX-hard [16] on rectangles, and (perhaps surprisingly) the problem is shown to be NP-hard even on diagonal-anchored rectangles [27]; that is, the intersection of every rectangle and a diagonal line with slope -1 is exactly one corner of the rectangle. See Figure 1(a) for an example. However, to the best of our knowledge no sub-logarithmic approximation is known even in this case. We note that optimization problems on "diagonal-intersecting" geometric objects have been studied before through the lenses of approximation algorithms; e.g. maximum independent set [9, 5, 22] and minimum hitting set [9, 8, 25].

Our results. In this paper, we first give a $(2+\epsilon)$-approximation algorithm for the MDS problem on diagonal-anchored rectangles, providing the first $O(1)$-approximation for the problem on a non-trivial subclass of rectangles.

▶ **Theorem 1.** *For any $\epsilon > 0$, there exists a $(2 + \epsilon)$-approximation algorithm for the* MDS *problem on diagonal-anchored rectangles.*

To prove Theorem 1, we first divide the problem into two subproblems and then give a PTAS for the subproblems using the *local search technique* [6, 26]. Each such subproblem involves diagonal-anchored rectangles that lie on only one side of the diagonal. The key to obtain our PTAS is in showing a planar drawing of a bipartite graph that is required for the analysis of the local search algorithm. We note that, even in these simpler cases the problem remains sufficiently challenging due to the geometry of the rectangles, and the existing schemes are not useful to obtain a near-optimal solution. For example, the local search analyses for non-piercing objects in [19] do not hold here, as the diagonal-anchored rectangles can still "pierce" each other.

It is not hard to see that the MDS problem on "diagonal-anchored" rectangles is the same as the MDS problem on "diagonal-anchored" L-*frames* [5]. An L-frame is the union of a vertical and a horizontal line segment that share an endpoint (*corner*). Indeed, each

[2] Two connected objects A and B are called non-piercing if both $A \setminus B$ and $B \setminus A$ are connected.

rectangle in the input can be replaced by a diagonal-anchored L-frame without altering the underlying intersection graph. Thus, a dominating set for the instance with the L-frames is also a dominating set for the original instance with rectangles, and vice versa. Hence, we also obtain a $(2+\epsilon)$-approximation for the problem with diagonal-anchored L-frames. For the MDS problem on general L-frames, the only approximation we are aware of is due to Mehrabi [24] who gave an $O(1)$-approximation algorithm when every two L-frames intersect in at most one point. Asinowski et al. [1] proved that every circle graph is an intersection graph of L-frames. Since MDS is APX-hard on circle graphs [10], the problem is also APX-hard on L-frames.

Note that, by definition, we can have four different *types* of L-frames depending on the two endpoints that define the corner. Considering the problem on L-frames we extend the APX-hardness result in the general case to two constrained cases. First, we show that the problem does not admit a $(1+\epsilon)$-approximation for any $\epsilon > 0$ on "diagonal-intersecting" L-frames (see Figure 1(b) for an example).

▶ **Theorem 2.** *The* MDS *problem is* APX*-hard on* L*-frames when every* L*-frame intersects a diagonal line.*

As the construction in proving Theorem 2 shows, the theorem holds even if the input consists of only one type of L-frames and all intersection points of the L-frames lie on only one side of the diagonal. This is in contrast to the diagonal-anchored L-frames case where we obtain a PTAS . We also show that one cannot hope for a $(1+\epsilon)$-approximation for any $\epsilon > 0$ even when all the L-frames intersect a vertical line; see Figure 1(c) for an example. We refer to these L-frames as *vertical-intersecting* L-frames.

▶ **Theorem 3.** *The* MDS *problem is* APX*-hard on vertical-intersecting* L*-frames even if all the* L*-frames intersect the vertical line from one side. Moreover, the problem is* NP*-hard even if for each* L*-frame, the horizontal and vertical segments have the same length.*

Moreover, we show that the APX-hardness of Theorem 3 is almost tight in the sense that the problem admits a polynomial-time algorithm on L-frames, where each L-frame intersects a vertical line and a horizontal line. See Figure 1(d) for an example. Note that, all the L-frames in the input are of the same type.

▶ **Theorem 4.** *The* MDS *problem is linear-time solvable on* L*-frames that intersect a vertical line and a horizontal line.*

To prove Theorem 4, we show that this class of graphs are the same as permutation graphs for which given the permutation of the vertices, the MDS problem can be solved in linear time [7]. As given a set of L-frames that intersect a vertical line and a horizontal line, the corresponding permutation can be computed in linear time, the theorem follows.

While the standard notion of intersection dates back to 1970s, Golumbic et al. [18] introduced the notion of *edge intersection* of L-frames. In this model, two L-frames are considered adjacent if and only if they overlap in strictly more than a single point in the plane. More formally, the L-frames corresponding to the vertices of the graph are drawn on a grid and two vertices are adjacent in the graph if and only if their corresponding L-frames share at least one grid edge; see Figure 1(e) for an example. To distinguish between the two models we will explicitly refer the edge intersection model whenever discussing a result on this model. Otherwise, we always mean the standard intersection model.

For the edge intersection model, there is a 4-approximation algorithm for MDS on L-frames [4, 20]. Moreover, the problem was recently shown to be APX-hard by Mehrabi [24],

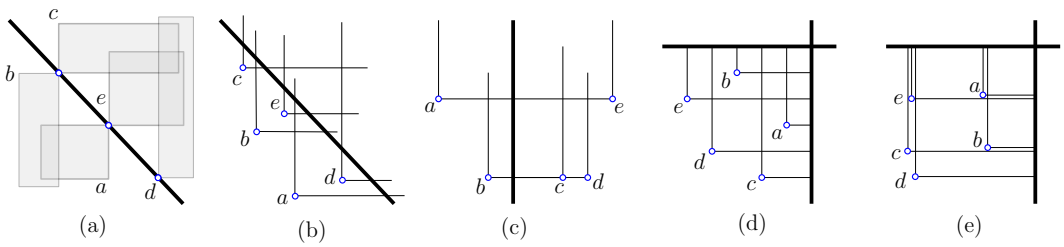


Figure 1 A graph $G = (\{a, b, c, d, e\}, E)$ with five different representations, where $E = \{(a, b), (a, e), (b, c), (c, d), (c, e), (d, e)\}$.

where two "types" of L-frames are needed for the construction. He left open whether the problem remains APX-hard when the input consists of only one type of L-frames or when the L-frames intersect a vertical line. We answer both questions affirmatively.

▶ **Theorem 5.** *In the edge intersection model, the* MDS *problem on* L*-frames of a single type is hard to approximate within a factor of 1.1377 even if all the* L*-frames intersect a vertical line from one side.*

Furthermore, we show that even intersecting two lines does not help: the MDS problem is NP-hard on L-frames in the edge intersection model even if every L-frame intersects a vertical line and a horizontal line. Observe that this is in contrast to the existence of the linear-time algorithm of Theorem 4 under the standard intersection model.

Organization. In Section 2, we give some definitions and revisit some necessary background. We prove Theorems 1 and 2 in Section 3. The proofs of Theorems 3 and 4 are given in Section 4. Finally, we show the results for the edge intersection model in Section 5 and conclude the paper in Section 6. Throughout this paper, the proofs of lemmas and theorems marked with $(*)$ are given in the full version of the paper due to space constraints.

2 Preliminaries

We denote the x- and y-coordinates of a point p by $x(p)$ and $y(p)$, respectively. For two points p and q, we denote the Euclidean distance between p and q by $\texttt{dist}(p, q)$. Given a graph $G = (V, E)$, we denote the L-frame corresponding to a vertex $u \in V(G)$ by $\mathsf{L}(u)$; we use u and $\mathsf{L}(u)$ interchangeably. We denote the corner of an L-frame l by $\texttt{cor}(l)$.

Local search. Consider an optimization problem in which the objective is to compute a feasible subset S' of a ground set S whose cardinality is minimum over all such feasible subsets of S. Moreover, it is assumed that computing some initial feasible solution and determining whether a subset $S' \subseteq S$ is a feasible solution can be done in polynomial time. The local search algorithm for a minimization problem is as follows. Fix some parameter k, and let A be some initial feasible solution for the problem. In each iteration, if there are $A' \subseteq A$ and $M \subseteq S \setminus A$ such that $|A'| \leq k$, $|M| < |A'|$ and $(A \setminus A') \cup M$ is a feasible solution, then set $A := (A \setminus A') \cup M$ and re-iterate. The algorithm returns A and terminates when no such local improvement is possible.

Clearly, the local search algorithm runs in polynomial time. Let $\mathcal{B}$ and $\mathcal{R}$ be the solution returned by the algorithm and an optimal solution, respectively. We can assume that $\mathcal{B} \cap \mathcal{R} = \emptyset$; otherwise, we can remove the common elements of $\mathcal{B}$ and $\mathcal{R}$ and analyze the

algorithm with the new sets, which guarantees that the approximation factor of the original instance is upper bounded by that of the new sets. The following result establishes the connection between local search technique and obtaining a PTAS.

▶ **Theorem 6** ([6, 26]). *Consider the solutions $\mathcal{B}$ and $\mathcal{R}$ for a minimization problem, and suppose that there exists a planar bipartite graph $H = (\mathcal{B} \cup \mathcal{R}, E)$ that satisfies the local exchange property, which is as follows: for any subset $\mathcal{B}' \subseteq \mathcal{B}$, $(\mathcal{B} \setminus \mathcal{B}') \cup N_H(\mathcal{B}')$ is a feasible solution, where $N_H(\mathcal{B}')$ denotes the set of neighbours of $\mathcal{B}'$ in H. Then, the local search algorithm yields a* PTAS *for the problem.*

3 Diagonal-intersecting Rectangles

In this section, we prove Theorems 1 and 2. To prove Theorem 1, we first give a PTAS for the problem when the rectangles are anchored at the diagonal from only one side and will then prove the theorem by applying the PTAS twice.

Recall the class of intersection graphs of diagonal-anchored rectangles is same as that of diagonal-anchored L-frames [5]. As such, to simplify the presentation of the result, we prove Theorem 1 for L-frames.

3.1 PTAS

Suppose that we are given a set of L-frames each of which is anchored at the diagonal from above; let $G = (V, E)$ be the corresponding graph. Consider any two L-frames L_1 and L_2 that intersect each other. We say that L_1 and L_2 are *coincident* if $x(\mathtt{cor}(\mathsf{L}_1))$=$x(\mathtt{cor}(\mathsf{L}_2))$. L_1 is said to intersect L_2 from left (resp. from below) if $x(\mathtt{cor}(\mathsf{L}_1)) \leq x(\mathtt{cor}(\mathsf{L}_2))$ (resp. $x(\mathtt{cor}(\mathsf{L}_1)) \geq x(\mathtt{cor}(\mathsf{L}_2))$). Notice that L_1 intersects L_2 from left if and only if L_2 intersects L_1 from below.

Consider the MDS problem on G and run the local search algorithm with $k := c/\epsilon$ for some constant c. Let $\mathcal{B}$ be the solution returned by the local search algorithm and $\mathcal{R}$ denote an optimal solution. Consider the bipartite graph $H = (\mathcal{B} \cup \mathcal{R}, E')$ in which the edge set E' is defined as follows. For any vertex $u \in V$, consider the set of all L-frames in $\{\mathcal{B} \cup \mathcal{R}\}$ that intersect $\mathsf{L}(u)$ and let (b_i, r_j), where $b_i \in \mathcal{B}$ and $r_j \in \mathcal{R}$, be a pair for which $\mathtt{dist}(\mathtt{cor}(b_i), \mathtt{cor}(r_j))$ (i.e., the Euclidean distance between their corners) is minimum over all such pairs. Then, $(b_i, r_j) \in E'$. We call u a *witness* vertex for the pair (b_i, r_j).

In the following, we show that H is planar (Lemma 7–11) and will then prove that H satisfies the local exchange property (Lemma 12). Then, by Theorem 6, our local search algorithm yields a PTAS for the problem on G. To distinguish between the edges of G and those of H, we refer to the edges of H as *arcs*. Let $b_i \in \mathcal{B}$ and $r_j \in \mathcal{R}$ such that $(b_i, r_j) \in E'$. We say that (b_i, r_j) is a *top arc* if there is a witness u for (b_i, r_j), such that *both* $\mathsf{L}(b_i)$ and $\mathsf{L}(r_j)$ intersect $\mathsf{L}(u)$ from left; choose one such u arbitrarily and denote $\mathsf{L}(u)$ by $\mathsf{w}(b_i, r_j)$. Otherwise, if there is a witness v for (b_i, r_j), such that *both* $\mathsf{L}(b_i)$ and $\mathsf{L}(r_j)$ intersect $\mathsf{L}(v)$ from below, (b_i, r_j) is a *down arc*; choose one such v arbitrarily and denote $\mathsf{L}(v)$ by $\mathsf{w}(b_i, r_j)$. Otherwise, for any witness w of (b_i, r_j), $\mathsf{L}(b_i)$ and $\mathsf{L}(r_j)$ intersect w from different sides; we call (b_i, r_j) a *mixed arc*, and choose one such w arbitrarily and denote $\mathsf{L}(w)$ by $\mathsf{w}(b_i, r_j)$.

Drawing of H. To draw H, we map u to $\mathtt{cor}(u)$ on the diagonal D for all $u \in \mathcal{B} \cup \mathcal{R}$. To draw the arcs of H, the idea is to draw each arc either above or below the diagonal depending on whether it is a top arc or a down arc, respectively. A mixed arc is drawn in such a way

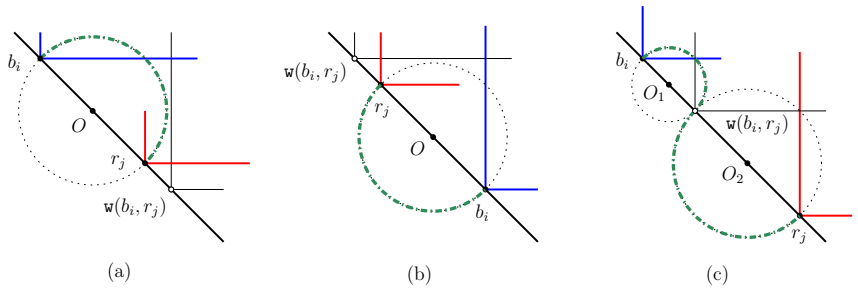


Figure 2 Drawing of (a) a top arc, (b) a down arc, and (c) a mixed arc.

that some part of the arc is drawn above and some part is drawn below the diagonal (and, hence crosses the diagonal). We next give the details of each case.

Let $b_i \in \mathcal{B}$ and $r_j \in \mathcal{R}$ such that $(b_i, r_j) \in E'$ and assume w.l.o.g. that $x(\mathtt{cor}(b_i)) < x(\mathtt{cor}(r_j))$. Moreover, let

$$O := \left(\frac{x(\mathtt{cor}(b_i)) + x(\mathtt{cor}(r_j))}{2}, \frac{y(\mathtt{cor}(b_i)) + y(\mathtt{cor}(r_j))}{2}\right),$$

and consider the circle C centred at O with radius $\mathtt{dist}(\mathtt{cor}(b_i), \mathtt{cor}(r_j))/2$. If the arc (b_i, r_j) is a top arc (resp., down arc), then we draw it on the half circle of C that lies above the diagonal D (resp., that lies below the diagonal D) starting from $\mathtt{cor}(b_i)$ and ending at $\mathtt{cor}(r_j)$; see Figure 2(a) – (b) for an illustration. A mixed arc is drawn in a slightly different way. For a mixed arc (b_i, r_j), assume w.l.o.g. that $\mathsf{L}(b_i)$ intersects $\mathtt{w}(b_i, r_j)$ from left while $\mathsf{L}(r_j)$ intersects $\mathtt{w}(b_i, r_j)$ from below. Notice that $x(\mathtt{cor}(b_i)) < x(\mathtt{cor}(\mathtt{w}(b_i, r_j))) < x(\mathtt{cor}(r_j))$. To draw (b_i, r_j), let

$$O_1 := \left(\frac{x(\mathtt{cor}(b_i)) + x(\mathtt{cor}(\mathtt{w}(b_i, r_j)))}{2}, \frac{y(\mathtt{cor}(b_i)) + y(\mathtt{cor}(\mathtt{w}(b_i, r_j)))}{2}\right),$$

$$O_2 := \left(\frac{x(\mathtt{cor}(\mathtt{w}(b_i, r_j))) + x(\mathtt{cor}(r_j))}{2}, \frac{y(\mathtt{cor}(\mathtt{w}(b_i, r_j))) + y(\mathtt{cor}(r_j))}{2}\right),$$

and consider the following two circles: the circle C_1 that is centred at O_1 and has the radius $\mathtt{dist}(\mathtt{cor}(b_i), \mathtt{cor}(\mathtt{w}(b_i, r_j)))/2$, and the circle C_2 that is centred at O_2 with radius $\mathtt{dist}(\mathtt{cor}(\mathtt{w}(b_i, r_j)), \mathtt{cor}(r_j))/2$. See Figure 2(c). We draw the first part of arc (b_i, r_j) on the half circle of C_1 that lies above the diagonal starting from $\mathtt{cor}(b_i)$ and ending at $\mathtt{cor}(\mathtt{w}(b_i, r_j))$ and then the second part on the half circle of C_2 that lies below the diagonal starting from $\mathtt{cor}(\mathtt{w}(b_i, r_j))$ and ending at $\mathtt{cor}(r_j)$. (The full version of the paper shows a complete example of graph H.)

Planarity of H. Clearly, no top arc crosses a down arc (except perhaps at their endpoints). To show the planarity of H, we show that – no top arc crosses another top arc, no down arc crosses another down arc, and no mixed arc crosses a top, a down or another mixed arc.

▶ **Lemma 7.** *No two top arcs in H cross each other.*

Proof. Suppose for a contradiction that two top arcs $(a, b), (c, d) \in E'$ cross each other, and w.l.o.g. assume that $x(a) < x(c) < x(b) < x(d)$. Since (a, b) is a top arc, we must have $x(\mathtt{cor}(\mathtt{w}(a, b))) \geq x(b)$; for a similar reason, we must have $x(\mathtt{cor}(\mathtt{w}(c, d))) \geq x(d)$. We now consider two cases. (i) If $x(\mathtt{cor}(\mathtt{w}(a, b))) \leq x(\mathtt{cor}(\mathtt{w}(c, d)))$, then $\mathsf{L}(c)$ must intersect $\mathtt{w}(a, b)$, which is a contradiction because in that case we should have added (a, c) or (c, b) to E'

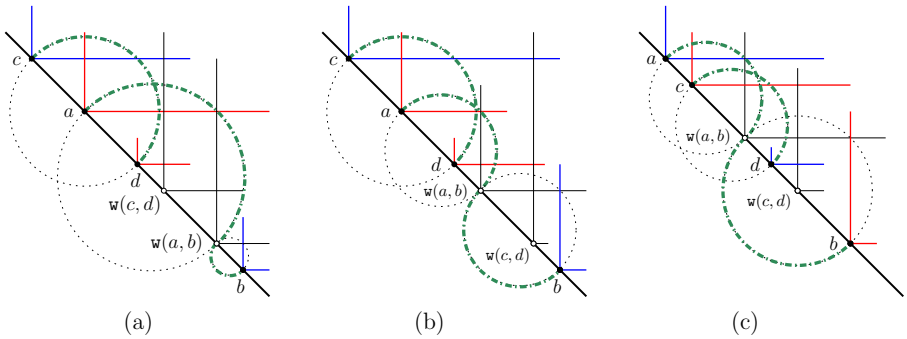


Figure 3 An illustration in supporting the proof of Lemma 9.

corresponding to $\mathsf{w}(a,b)$ instead of (a,b). (ii) If $x(\mathtt{cor}(\mathsf{w}(a,b))) > x(\mathtt{cor}(\mathsf{w}(c,d)))$, then $\mathsf{L}(b)$ must intersect $\mathsf{w}(c,d)$ – this is also a contradiction because in that case we should have added (c,b) or (b,d) to E' corresponding to $\mathsf{w}(c,d)$ instead of (c,d). ◀

▶ **Lemma 8** (∗). *No two down arcs in H cross each other.*

It remains to show that no mixed arc crosses a top arc, a down arc or another mixed arc.

▶ **Lemma 9.** *No mixed arc crosses a top arc in H.*

Proof. Suppose for a contradiction that a mixed arc (a,b) crosses a top arc (c,d) and assume w.l.o.g. that $x(a) < x(b)$ and $x(c) < x(d)$. If $x(c) < x(a)$, then we must have $x(a) < x(d) < x(\mathtt{cor}(\mathsf{w}(a,b)))$ – otherwise, the two arcs would not cross. First, we know that $x(\mathtt{cor}(\mathsf{w}(c,d))) \geq x(d)$. If $x(\mathtt{cor}(\mathsf{w}(c,d))) < x(\mathtt{cor}(\mathsf{w}(a,b)))$, then $\mathsf{L}(a)$ intersects $\mathsf{w}(c,d)$ before intersecting $\mathsf{w}(a,b)$, which is a contradiction because in that case we should have added either the top arc (c,a) or the top arc (a,d) to E' corresponding to $\mathsf{w}(c,d)$ instead of (c,d); see Figure 3(a). If $x(\mathtt{cor}(\mathsf{w}(c,d))) > x(\mathtt{cor}(\mathsf{w}(a,b)))$, then $\mathsf{L}(d)$ intersects $\mathsf{w}(a,b)$ before intersecting $\mathsf{w}(c,d)$, which is again a contradiction as we should have added either the top arc (a,d) or the mixed arc (d,b) to E' corresponding to $\mathsf{w}(a,b)$ instead of (a,b). See Figure 3(b).

Now, suppose that $x(c) > x(a)$. Then, we must have $x(a) < x(c) < x(\mathtt{cor}(\mathsf{w}(a,b))) < x(d)$ as the two arcs would not intersect otherwise. Since (c,d) is a top arc, we must have $x(\mathtt{cor}(\mathsf{w}(c,d))) \geq x(d)$ and so $\mathsf{L}(c)$ would intersect $\mathsf{w}(a,b)$ before intersecting $\mathsf{w}(c,d)$; see Figure 3(c). This is a contradiction because we should have added either the top arc (a,c) or the mixed arc (c,b) to E' corresponding to $\mathsf{w}(a,b)$ instead of (a,b). ◀

▶ **Lemma 10** (∗). *No mixed arc crosses a down arc in H.*

▶ **Lemma 11** (∗). *No two mixed arcs cross each other in H.*

By Lemma 7 – 11, it follows that H is a planar graph. The following lemma along with the planarity of H gives a PTAS for the problem on G.

▶ **Lemma 12.** *Graph $H = (\mathcal{B} \cup \mathcal{R}, E)$ satisfies the local exchange property.*

Proof. To prove the lemma, it is sufficient to show that for any vertex $u \in V$, there are $b_i \in \mathcal{B}$ and $r_j \in \mathcal{R}$ such that both $\mathsf{L}(b_i)$ and $\mathsf{L}(r_j)$ intersect $\mathsf{L}(u)$ and $(b_i, r_j) \in E'$. Take any vertex $u \in V$ and let $S \subseteq \mathcal{B} \cup \mathcal{R}$ be the set of all L-frames that intersect $\mathsf{L}(u)$. Notice that $S \cap \mathcal{B} \neq \emptyset$ and $S \cap \mathcal{R} \neq \emptyset$ because each of $\mathcal{B}$ and $\mathcal{R}$ is a feasible solution to the MDS problem

on G. Now, consider $b \in S \cap \mathcal{B}$ and $r \in S \cap \mathcal{R}$ for which $\texttt{dist}(\texttt{cor}(b), \texttt{cor}(r))$ is minimum over all such pairs (b, r). We know by definition of H that $(b, r) \in E'$, which proves the lemma. ◀

3.2 Proof of Theorem 1

We are now ready to prove Theorem 1. Here, the rectangles are anchored at the diagonal from both sides; let $G = (V, E)$ denote the corresponding graph.

Proof of Theorem 1. Let $\{X, Y\}$ be a partition of the vertices of G such that the L-frames corresponding to vertices in X (resp. Y) are anchored at D from above (resp. below). By abusing notation, we refer to these two sets of L-frames also as X and Y. For any $\epsilon > 0$, set $\epsilon' := \epsilon/2$. We apply the PTAS of Section 3.1 with parameter c/ϵ' to the L-frames in X and Y independently, and let S_X and S_Y be the solutions returned by the algorithm, respectively. We return $S := S_X \cup S_Y$ as the final solution. Let OPT, $\mathsf{OPT_X}$ and $\mathsf{OPT_Y}$ denote an optimal solution for the MDS problem on the L-frames in $X \cup Y$ (i.e., graph G), X and Y, respectively.

Consider the solution OPT. Let $S \subseteq \mathsf{OPT}$ be a minimum size set of L-frames that dominates all the L-frames in X. If there is an L-frame $P \in S$ that is in Y, then P can only dominate those L-frames in X that are anchored at the diagonal at the same point as P, in which case we can replace P by one of those L-frames from X. As such, there exists a set $S' \subseteq X$ of size at most $|S| \leq |\mathsf{OPT}|$ that dominates the L-frames of X. It follows that $|\mathsf{OPT_X}| \leq |\mathsf{OPT}|$. Similarly, one can show that $|\mathsf{OPT_Y}| \leq |\mathsf{OPT}|$. Now, by using the result from Section 3.1, we have $|S_X| \leq (1+\epsilon')|\mathsf{OPT_X}|$ and $|S_Y| \leq (1+\epsilon')|\mathsf{OPT_Y}|$. Then,

$$|S| \leq |S_X| + |S_Y| \leq (1+\epsilon')|\mathsf{OPT_X}| + (1+\epsilon')|\mathsf{OPT_Y}| \leq 2(1+\epsilon')|\mathsf{OPT}| = (2+\epsilon)|\mathsf{OPT}|,$$

which completes the proof of the theorem. ◀

3.3 Proof of Theorem 2

Recall that Theorem 2 claims APX-hardness of the problem on L-frames in the diagonal-intersecting case. We show a gap-preserving reduction from the MDS problem on *circle graphs*, which is known to be APX-hard [10]. Recall that a graph is called a circle graph, if it is the intersection graph of chords of a circle. Take any circle graph $G = (V, E)$ with n vertices, and consider a geometric representation of G. By a closer look at the APX-hardness proof [10], we can assume that no two chords share an endpoint; that is, there are exactly $2n$ distinct points on the circle determining the endpoints of the chords. Cut the circle at an arbitrary point p and consider the ordering $M := \langle p_1, p_2, \ldots, p_{2n} \rangle$ of the endpoints of chords visited in counter-clockwise along the circle starting at p. Now, let D be the diagonal line $y = (2n+1) - x$. Consider each endpoint p_i (where $1 \leq i \leq 2n$) in the order given by M. Then, we map each point p_i to the point $((2n+1) - i, i)$ on D. Let $e := (p_j, p_k)$ be a chord of G. Then, the L-frame corresponding to e is the unique L-frame that lies below D and connects the two points mapped on D corresponding to p_j and p_k; see Figure 4 for an example. Let $G' = (V', E')$ be the graph corresponding to the constructed L-frames. Clearly, D has slope -1 and the construction can be done in polynomial time. Moreover, since the $2n$ endpoints of chords of the input representation are pairwise distinct, the points mapped to the line D are in general position; that is, no two such points have the same x- or the same y-coordinates. The following lemma completes the proof of Theorem 2.

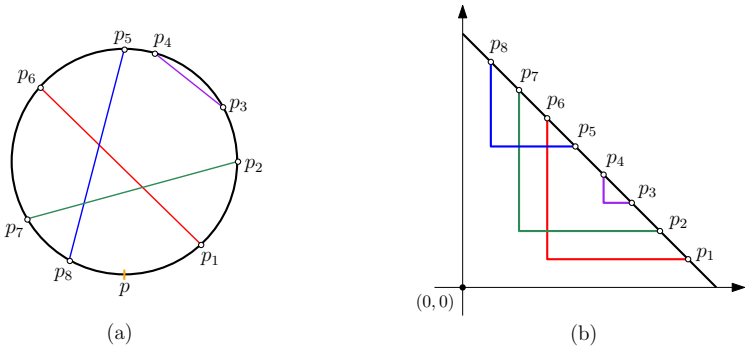


Figure 4 An illustration supporting the proof of Theorem 2.

▶ **Lemma 13.** *G has a dominating set of size k if and only if G' has a dominating set of size k.*

Proof. First, we show that there is a one-to-one correspondence between the vertices of G and those of G' such that $(i,j) \in E$ if and only if $(i,j) \in E'$. The one-to-one correspondence between the vertices of these two graphs is clear from the construction. Now, suppose that $(i,j) \in E$. This means that the endpoints of vertices i and j appear in M as either $\langle p_i, p_j, p_{i'}, p_{j'} \rangle$ or $\langle p_j, p_i, p_{j'}, p_{i'} \rangle$, where p_i and $p_{i'}$ with $i < i'$ (resp., p_j and $p_{j'}$ with $j < j'$) correspond to the endpoints of i (resp., j). By the mapping used in the construction, this ordering is preserved on D (when going from $(2n+1, 0)$ to $(0, 2n+1)$ along D) and so the L-frames $\mathsf{L}(i)$ and $\mathsf{L}(j)$ intersect each other below D; that is, $(i,j) \in E'$. Conversely, if $(i,j) \notin E$, then their endpoints appear in M as either $\langle p_i, p_j, p_{j'}, p_{i'} \rangle$ or $\langle p_j, p_i, p_{i'}, p_{j'} \rangle$, where p_i and $p_{i'}$ with $i < i'$ (resp., p_j and $p_{j'}$ with $j < j'$) correspond to the endpoints of i (resp., j). Consequently, this ordering is preserved on D by the mapping (when going from $(2n+1, 0)$ to $(0, 2n+1)$ along D) and so $\mathsf{L}(i)$ and $\mathsf{L}(j)$ do not intersect each other inside D; that is, $(i,j) \notin E'$. It is now straightforward to see that G has a dominating set of size k if and only if G' has a dominating set of size k, which is clearly a gap-preserving reduction. ◀

Remark. Using a similar reduction in the other direction one can show that, for every intersection graph G of a set of diagonal-intersecting L-frames that intersect each other only below the diagonal, one can find a circle graph G' such that G has a dominating set of size k if and only if G' has a dominating set of size k. As one can obtain a $2+\epsilon$-approximation for MDS on *circle graphs* [11], MDS can be approximated within a factor of $2+\epsilon$ in this special case of diagonal-intersecting L-frames. Note that this is the version which we have just proved to be APX-hard.

4 Vertical-intersecting L-frames

In this section, we prove Theorems 3 and 4. To prove Theorem 3, we first show that the MDS problem is APX-hard even when each L-frame intersects a vertical line. The proof is essentially the same as that of Theorem 2, but by a slight modification of replacing the diagonal D with a quarter of a circle and then extending the horizontal segment of each L-frame until it hits a vertical line that is placed far to the right. See the full version of the paper for the complete proof.

To complete the proof of Theorem 3, we show that the MDS problem is NP-hard on L-frames even if the horizontal and vertical segments of every L-frame have the same length. The reduction is from a variant of the 3SAT problem. For any 3SAT instance, one can define

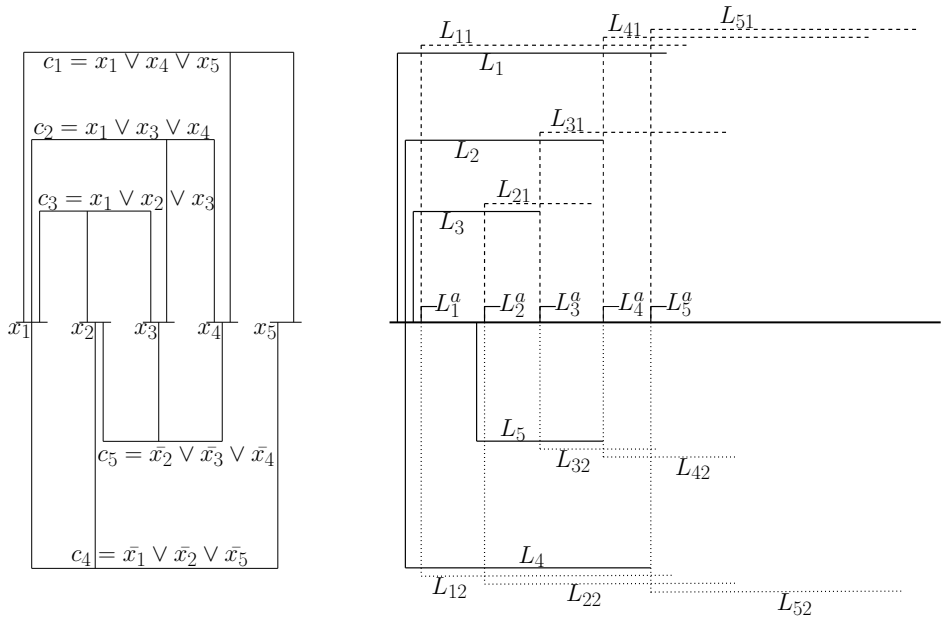


Figure 5 A drawing of an instance of Planar Monotone Rectilinear 3SAT (left) and the corresponding instance of the MDS problem before rotation (right). The variable L-frames above (resp. below) the x-axis are shown in dashed (resp. dotted) style. The clause L-frames and the auxilliary L-frames are shown in normal style.

a bipartite graph on the clauses and the variables which we refer to as the incidence graph. For any clause c_j, if c_j contains a literal corresponding to a variable v_i, the edge (v_i, c_j) is added to the graph. A drawing of a planar incidence graph is called *planar rectilinear* if it has the following properties. Each variable vertex is drawn as a horizontal segment on the x-axis and the clause vertices are drawn as horizontal segments above and below the x-axis. For each edge (v_i, c_j), the horizontal segment corresponding to c_j has a vertical connection to the horizontal segment corresponding to v_i. Moreover, this vertical connection does not intersect any other horizontal segments. See the left figure of Figure 5 for an example. An instance of the 3SAT problem is called *monotone* if, for every clause in the instance, the literals of the clause are either all positive (called a *positive* clause) or all negative (called a *negative* clause). A planar rectilinear drawing of an incidence graph is called *monotone* if it corresponds to a monotone 3SAT instance such that all the positive (resp., negative) clauses are drawn above (resp., below) the x-axis. In the Planar Monotone Rectilinear 3SAT problem the incidence graph of any instance has a planar rectilinear and monotone drawing. The problem is known to be NP-hard even when the drawing is given [13]. Given an instance of the Planar Monotone Rectilinear 3SAT, we construct a set of L-frames such that the horizontal and vertical segments of each L-frame have the same length and they all intersect a vertical line.

Let n be the number of variables. Due to the hierarchical structure of the clauses lying above and below the x-axis, respectively in the given drawing, one can modify the drawing in a way such that for each clause, the length of the horizontal segment corresponding to it is the same as the length of the vertical segments corresponding to it (see the left figure of Figure 5). Let T be the new drawing. We construct the instance I of MDS from T in the following way. For each clause c_j, removing all the vertical connections except the leftmost one creates an L-frame. We add this L-frame (denoted by L_j) to our instance. For each

variable x_i, we add two L-frames: one (denoted by L_{i1}) above and the other (denoted by L_{i2}) below the x-axis. The vertical segments of these two L-frames are the maximum length vertical connections corresponding to x_i lying above and below x-axis, respectively. For the variable L-frames, their horizontal segments lie on the right of their vertical segment. Moreover, for any variable L-frame above (resp., below) the x-axis, the corner is the topmost (resp., bottommost) point of the vertical segment. The two L-frames corresponding to each variable are shifted accordingly so that they intersect at a point on the x-axis (see Figure 5). Also, for each variable x_i, we add an auxiliary L-frame L_i^a that intersects only the two L-frames corresponding to it as shown in Figure 5. Note that in the constructed instance, one endpoint of each L-frame lies on the x-axis. Now it could be the case that, for some i, j and $k \in \{1, 2\}$, an L_{ik} intersects L_j though c_j does not contain any literal of x_i. To get rid of these intersections, we extend each L_{ik} vertically and then horizontally (to maintain the symmetry) so that it does not intersect any L_j such that c_j does not contain x_i. Note that, as we do not modify the L_j's, if c_j contains the literal x_i (resp. $\bar{x_i}$), L_{i1} (resp. L_{i2}) still intersects L_j. Finally, we rotate the created instance clockwise by 90° with respect to the origin. Hence all the L-frames in the constructed instance now intersect the vertical line $x = 0$ or the y-axis, and thus together they form an instance of the problem with vertical-intersecting L-frames. Lemma 15 below completes the proof of the second part of Theorem 3. We first need the following observation whose proof is immediate from the construction.

▶ **Observation 14.** *For all i, j, L_{i1} (resp., L_{i2}) intersects L_j if and only if the clause c_j contains the literal x_i (resp., $\bar{x_i}$).*

▶ **Lemma 15.** *The instance I has a dominating set of size n if and only if the 3SAT instance is satisfiable.*

Proof. Suppose I has a dominating set D of size n. We construct a satisfiable assignment in the following way. Consider any variable x_i. As L_i^a intersects only L_{i1} and L_{i2}, one of these three L-frames must be in the dominating set. The size of D being n, only one of the above mentioned three L-frames can be in D and no clause L-frame can be in D. If L_{i1} is in D, we set x_i to be true. Otherwise, we set x_i to be false. Now consider any clause c_j. Let x_i be a variable such that a L-frame corresponding to x_i is in D that intersects L_j. Note that there exists such an x_i. Now if L_{i1} is the chosen L-frame that dominates L_j, then L_{i1} is in D and by Observation 14 c_j must be a positive clause. It follows that, we have set x_i to be true and thus c_j is satisfied. Similarly, one can show that if L_{i2} is the chosen L-frame that dominates L_j, then also c_j is satisfied.

Now, suppose that we are given a satisfiable assignment of the 3SAT instance. We construct a dominating set in the following way. For any variable x_i, if x_i is set to be true, then we select L_{i1}. If x_i is set to be false, then we select L_{i2}. We claim that the selected L-frames form a dominating set. It is easy to see that the selected L-frames dominate the variable and auxiliary L-frames. Now, consider any clause L-frame L_j. There must be a variable x_i that satisfies c_j. If c_j is positive, the literal x_i must be set to true and we select L_{i1}. By Observation 14, L_{i1} intersects L_j, and thus L_j is being dominated. If c_j is negative, one can similarly show that L_j is also being dominated. ◀

We now prove Theorem 4. To this end, we show that the intersection graph of L-frames that inersect a vertical and a horizontal line is a permutation graph and so MDS is linear-time solvable on such a graph [7]. Geometrically, a graph G is a permutation graph if there are two embeddings of its vertices on two parallel lines such that, when connecting a vertex from

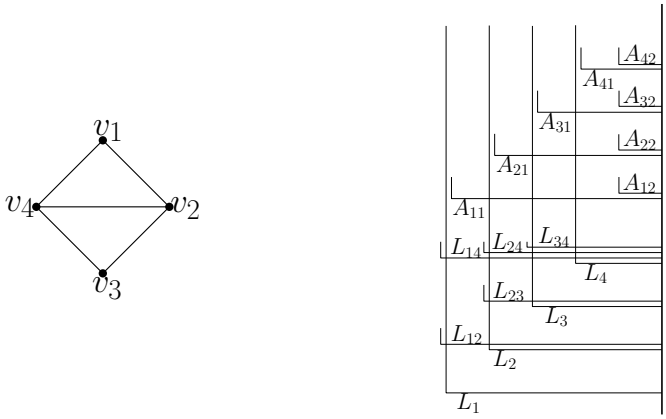


Figure 6 An example graph with 4 vertices (left) and its corresponding instance of the MDS problem (right).

the first line to the same vertex on the other line using a line segment, the edge set of the graph is realized exactly by the intersections of those segments. That is, two vertices appear in different order on the parallel lines (and, so their line segments intersect) whenever they are adjacent in the graph. By taking the two orderings in which the endpoints of the L-frames in the input graph intersect the two lines, we can construct the geometric representation of a permutation graph having the same edges. Hence, Theorem 4 follows.

5 Edge Intersection Model

In this section, we show our results for the edge intersection model [18].

Proof of Theorem 5. We show a reduction from the Vertex Cover problem. In Vertex Cover, we are given a graph $G = (V, E)$ with n vertices and the goal is to find a subset $V' \subseteq V$ such that for any edge $(v_i, v_j) \in E$, $V' \cap \{v_i, v_j\} \neq \emptyset$. Let $VC(G)$ be the size of any minimum size vertex cover of G. As noted in [2], from the work of Dinur and Safra [14] the following theorem can be derived.

▶ **Theorem 16** (Dinur and Safra [14]). *Let $1/3 < p < p_{max} = (3 - \sqrt{5})/2$ and $q = 4p^3 - 3p^4$. For any constant $\epsilon > 0$, given any unweighted graph $G = (V, E)$ with bounded degrees, it is NP-hard to distinguish between "Yes": $VC(G) < (1 - p + \epsilon)|V|$, and "No": $VC(G) > (1 - q - \epsilon)|V|$.*

Given the vertex cover graph $G = (V, E)$, we construct an instance I of MDS. In the instance I, all the L-frames lie on the left of the line $x = 0$ and intersect the line, and thus each has the ⌞ shape. The construction is as follows. For each vertex v_i, we have an L-frame L_i that intersects $x = 0$ at the point $(0, 2i)$ as shown in Figure 6. Note that these L-frames do not intersect each other. For each edge (v_i, v_j) with $i < j$, we have an L-frame L_{ij} that intersects $x = 0$ at $(0, 2j)$, and thus intersects (i.e., has a common horizontal grid edge) with v_j. It also intersects (has a common vertical grid edge) with v_i (see Figure 6). Also, for each vertex v_i, we have two auxiliary L-frames A_{i1} and A_{i2} that intersect $x = 0$ at $(0, 2n + i)$, and A_{i1} shares a vertical grid edge with L_i. Note that the only L-frame A_{i2} intersects is A_{i1}. Also, all the L-frames intersect the vertical line $x = 0$. We first need the following observations.

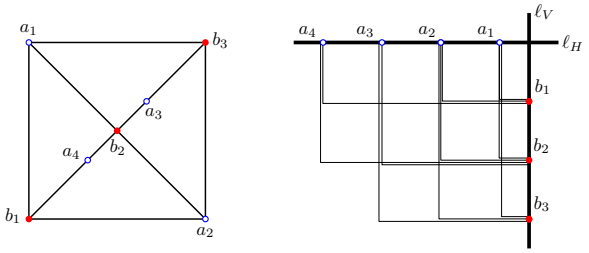


Figure 7 A bipartite graph on 7 vertices (left) and its corresponding representation in the edge intersection model (right).

▶ **Observation 17.** *For $1 \leq i < j \leq n$, the only* L*-frames that L_{ij} intersects are L_i, L_j and L_{tj}, where $(v_t, v_j) \in E$ with $t < j$.*

▶ **Observation 18.** *The only* L*-frames that get dominated by the* L*-frames in $\{A_{i1} : 1 \leq i \leq n\}$ are L_i and A_{i2} for $1 \leq i \leq n$.*

▶ **Observation 19.** *Any dominating set contains at least one of A_{i1} and A_{i2} for each $1 \leq i \leq n$. Also for any dominating set of size k, there is a dominating set D of size at most k such that D contains A_{i1} for all $1 \leq i \leq n$ and does not contain any A_{i2} for $1 \leq i \leq n$.*

▶ **Lemma 20.** *G has a vertex cover of size k if and only if I has a dominating set of size $k + n$.*

Proof. Suppose G has a vertex cover of size k. We construct a dominating set D of size $k + n$ in the following way. At first add all the L-frames $\{A_{i1} : 1 \leq i \leq n\}$ to D. Then for each v_i in the vertex cover, add L_i to D. Clearly $|D| = k + n$. By Observation 18, D dominates any L_i and A_{i2} for $1 \leq i \leq n$. Now consider any L_{ij}. Note that we have added at least one of L_i and L_j to D, and hence D dominates L_{ij} as well.

Now, suppose we have a dominating set D' of size $k + n$. By Observation 19, we can assume that D' contains A_{i1} for all i and does not contain any A_{i2}. Thus D' contains k L-frames corresponding to the vertices and the edges. Now suppose D' contains an edge L-frame L_{ij}. By Observation 17 and 18, the only L-frames that cannot get dominated by D' by the removal of L_{ij} from D' are L_{tj}, where $(v_t, v_j) \in E$ with $t < j$. Thus if we replace L_{ij} by L_j in D', D' still remains a dominating set. Thus, we can assume w.l.o.g. that D' does not contain any L_{ij}. Now we construct a subset $V' \subseteq V$ by selecting vertices corresponding to the vertex L-frames contained in D'. Clearly $|V'| = k$. We claim that V' is a vertex cover. Consider any edge (v_i, v_j). Then by Observation 18, at least one of L_i and L_j must be in D'. It follows that at least one of v_i and v_j is present in V' which finishes the proof. ◀

By Theorem 16 and Lemma 20, we have the following lemma.

▶ **Lemma 21.** *Let $1/3 < p < p_{max} = (3 - \sqrt{5})/2$ and $q = 4p^3 - 3p^4$. For any constant $\epsilon > 0$, given an input graph (in the edge intersection model) H with $O(n)$ vertices, it is* NP*-hard to distinguish between "Yes": H has a dominating set of size $< (2 - p + \epsilon)n$, and "No": The size of any dominating set of H is $> (2 - q - \epsilon)n$.*

Therefore, the gap between the sizes of the minimum dominating set in the "yes" and the "no" cases approaches $(2 - 4p^3 + 3p^4)/(2 - p) \approx 1.1377$ for $p = p_{max}$. Hence, Theorem 5 follows.

NP-hardness. We show a reduction from the *Edge Dominating Set* problem that is known to be NP-hard, even on planar bipartite graphs [21]. Recall that the objective of the Edge Dominating Set problem on a graph is to choose a minimum-cardinality set S of edges of the graph such that every edge not in S shares at least one endpoint with one edge in S. Given a planar bipartite graph $G = (A \cup B, E)$, we construct an intersection graph G' of L-frames in polynomial time such that G has an edge dominating set of size k if and only if G' has a dominating set of size k (in the edge intersection model). To this end, let $A = \{a_1, \ldots, a_r\}$ and $B = \{b_1, \ldots, b_s\}$, and for each vertex $a_i \in A$ (resp., vertex $b_j \in B$), consider the point $(-i, 0)$ (resp., point $(0, -j)$) of the Cartesian coordinate system in the plane. Then, for each edge $(a_i, b_j) \in E(G)$, add to $V(G')$ the unique L-frame that connects $(-i, 0)$ to $(0, -j)$ and has the point $(-i, -j)$ as its corner. See Figure 7. Let G' be the resulting graph. Clearly, every L-frame in G' intersects a vertical and a horizontal line and it can be constructed in polynomial time. Moreover, notice the one-to-one correspondence between the edges of G and the L-frames in G'. It is now easy to see that G has an edge dominating set of size k if and only if G' has a dominating set of size k (in the edge intersection model).

6 Conclusion

In this paper, we considered the MDS problem on the intersection graphs of rectangles and L-frames. Among several other approximation and hardness results, we gave a $(2+\epsilon)$-approximation algorithm for the problem on diagonal-anchored rectangles, which was based on a PTAS for when the rectangles are anchored from one side. However, the complexity of the latter problem remains open. The problem is NP-hard when the rectangles are anchored from both sides; is the problem APX-hard? Or, does the problem admit a PTAS? Even, finding an algorithm with approximation factor better than $(2+\epsilon)$ remains open.

References

1. Andrei Asinowski, Elad Cohen, Martin Charles Golumbic, Vincent Limouzy, Marina Lipshteyn, and Michal Stern. Vertex intersection graphs of paths on a grid. *J. Graph Algorithms Appl.*, 16(2):129–150, 2012.
2. Pranjal Awasthi, Moses Charikar, Ravishankar Krishnaswamy, and Ali Kemal Sinop. The hardness of approximation of Euclidean k-means. In *SoCG 2015, Netherlands, 754–767*, volume 34 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 2015.
3. Sayan Bandyapadhyay, Anil Maheshwari, Saeed Mehrabi, and Subhash Suri. Approximating dominating set on intersection graphs of L-frames. *CoRR*, abs/1803.06216, 2018.
4. Ayelet Butman, Danny Hermelin, Moshe Lewenstein, and Dror Rawitz. Optimization problems in multiple-interval graphs. *ACM Trans. Algorithms*, 6(2):40:1–40:18, 2010.
5. Daniele Catanzaro, Steven Chaplick, Stefan Felsner, Bjarni V. Halldórsson, Magnús M. Halldórsson, Thomas Hixon, and Juraj Stacho. Max point-tolerance graphs. *Discrete Applied Mathematics*, 216:84–97, 2017.
6. Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012.
7. H. S. Chao, Fang-Rong Hsu, and Richard C. T. Lee. An optimal algorithm for finding the minimum cardinality dominating set on permutation graphs. *Discrete Applied Mathematics*, 102(3):159–173, 2000.
8. Victor Chepoi and Stefan Felsner. Approximating hitting sets of axis-parallel rectangles intersecting a monotone curve. *Comput. Geom.*, 46(9):1036–1041, 2013.

9 José R. Correa, Laurent Feuilloley, Pablo Pérez-Lantero, and José A. Soto. Independent and hitting sets of rectangles intersecting a diagonal line: Algorithms and complexity. *DCG*, 53(2):344–365, 2015.

10 Mirela Damian and Sriram V. Pemmaraju. APX-hardness of domination problems in circle graphs. *Inf. Process. Lett.*, 97(6):231–237, 2006.

11 Mirela Damian-Iordache and Sriram V. Pemmaraju. A (2+epsilon)-approximation scheme for minimum domination on circle graphs. *J. Algorithms*, 42(2):255–276, 2002.

12 Minati De and Abhiruk Lahiri. Geometric dominating set and set cover via local search. *CoRR*, abs/1605.02499, 2016. `arXiv:1605.02499`.

13 Mark de Berg and Amirali Khosravi. Optimal binary space partitions for segments in the plane. *Int. J. Comput. Geometry Appl.*, 22(3):187–206, 2012.

14 Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(1):439–485, 2005.

15 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 624–633, 2014.

16 Thomas Erlebach and Erik Jan van Leeuwen. Domination in geometric intersection graphs. In *LATIN 2008, Búzios, Brazil, April 7-11, 2008, Proceedings*, pages 747–758, 2008.

17 Matt Gibson and Imran A. Pirwani. Algorithms for dominating set in disk graphs: Breaking the log$n$ barrier - (extended abstract). In *ESA 2010, UK*, pages 243–254, 2010.

18 Martin Charles Golumbic, Marina Lipshteyn, and Michal Stern. Edge intersection graphs of single bend paths on a grid. *Networks*, 54(3):130–138, 2009.

19 Sathish Govindarajan, Rajiv Raman, Saurabh Ray, and Aniket Basu Roy. Packing and covering with non-piercing regions. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 47:1–47:17, 2016.

20 Daniel Heldt, Kolja B. Knauer, and Torsten Ueckerdt. Edge-intersection graphs of grid paths: The bend-number. *Discrete Applied Mathematics*, 167:144–162, 2014.

21 Joseph D. Horton and Kyriakos Kilakos. Minimum edge dominating sets. *SIAM J. Discrete Math.*, 6(3):375–387, 1993.

22 J. Mark Keil, Joseph S. B. Mitchell, Dinabandhu Pradhan, and Martin Vatshelle. An algorithm for the maximum weight independent set problem on outerstring graphs. *Comput. Geom.*, 60:19–25, 2017.

23 Dániel Marx. Parameterized complexity of independence and domination on geometric graphs. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, pages 154–165, 2006.

24 Saeed Mehrabi. Approximating domination on intersection graphs of paths on a grid. In *15th International Workshop on Approximation and Online Algorithms (WAOA 2017), Vienna, Austria*, pages 76–89, 2017.

25 Apurva Mudgal and Supantha Pandit. Covering, hitting, piercing and packing rectangles intersecting an inclined line. In *COCOA 2015, Houston, TX, USA*, pages 126–137, 2015.

26 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.

27 Supantha Pandit. Dominating set of rectangles intersecting a straight line. In *CCCG 2017, Ottawa, Ontario, Canada*, pages 144–149, 2017.

28 Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 475–484, 1997.

On Efficiently Solvable Cases of Quantum k-SAT

Marco Aldi
Department of Mathematics and Applied Mathematics, Virginia Commonwealth University, Richmond, VA, USA
maldi2@vcu.edu

Niel de Beaudrap[1]
Department of Computer Science, University of Oxford, UK
niel.debeaudrap@cs.ox.ac.uk

Sevag Gharibian[2]
Department of Computer Science, University of Paderborn, Germany, and Virginia Commonwealth University, USA
sevag.gharibian@uni-paderborn.de

Seyran Saeedi
Department of Computer Science, Virginia Commonwealth University, Richmond, VA, USA
saeedis@vcu.edu

Abstract

The constraint satisfaction problems k-SAT and Quantum k-SAT (k-QSAT) are canonical NP-complete and QMA_1-complete problems (for $k \geq 3$), respectively, where QMA_1 is a quantum generalization of NP with one-sided error. Whereas k-SAT has been well-studied for special tractable cases, as well as from a parameterized complexity perspective, much less is known in similar settings for k-QSAT. Here, we study the open problem of computing satisfying assignments to k-QSAT instances which have a "matching" or "dimer covering"; this is an NP problem whose decision variant is trivial, but whose search complexity remains open.

Our results fall into three directions, all of which relate to the "matching" setting: (1) We give a polynomial-time classical algorithm for k-QSAT when all qubits occur in at most two clauses. (2) We give a parameterized algorithm for k-QSAT instances from a certain non-trivial class, which allows us to obtain exponential speedups over brute force methods in some cases by reducing the problem to solving for a single root of a single univariate polynomial. (3) We conduct a structural graph theoretic study of 3-QSAT interaction graphs which have a "matching". We remark that the results of (2), in particular, introduce a number of new tools to the study of Quantum SAT, including graph theoretic concepts such as transfer filtrations and blow-ups from algebraic geometry; we hope these prove useful elsewhere.

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

Keywords and phrases search complexity, local Hamiltonian, Quantum SAT, algebraic geometry

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.38

Related Version A full version of this paper is available at `https://arxiv.org/abs/1712.09617`.

[1] NdB acknowledges support from the EPSRC National Quantum Technology Hub in Networked Quantum Information Processing.
[2] SG acknowledges support from NSF grants CCF-1526189 and CCF-1617710.

Funding This work was partially supported by the EPSRC National Quantum Technology Hub in Networked Quantum Information Processing, and NSF grants CCF-1526189 and CCF-1617710.

Acknowledgements The first result of this project was partially completed while NdB was affiliated with the Centrum Wiskunde & Informatica; SG thanks Ronald de Wolf and Centrum Wiskunde & Informatica for their hospitality. SG thanks Howard Barnum and David Reeb regarding discussions on algebraic geometry, and David Gosset for discussions on Quantum SAT.

1 Introduction

Constraint satisfaction problems (CSPs) are cornerstones of both classical and quantum complexity theory. Indeed, CSPs such as 3-SAT and MAX-2-SAT are complete for NP [13], and their analogues Quantum 3-SAT (3-QSAT) and the 2-local Hamiltonian problem are QMA_1- and QMA-complete, respectively [3, 10, 16, 15]. (QMA is Quantum Merlin-Arthur, a quantum generalization of Merlin-Arthur, and QMA_1 is QMA with perfect completeness.) As such CSPs are intractable in the worst case, approaches such as approximation algorithms, heuristics, and exact algorithms are employed. In this paper, we focus on the latter technique, and ask: Which special cases of k-QSAT can be solved efficiently on a classical computer?

Unfortunately, this problem appears to be markedly more difficult than in the classical setting. For example, classically, if each clause c of a k-SAT instance can be matched with a unique variable v_c, then clearly the k-SAT instance is satisfiable, and finding a solution is trivial: Set variable v_c to satisfy clause c. (Note that the matching can be found efficiently via, e.g., the Ford-Fulkerson algorithm [11].) In the quantum setting, it has been known [17] since 2010 that k-QSAT instances with such "matchings" (also called a "dimer covering" in physics [17]) are also satisfiable, and moreover the satisfying assignment can be represented succinctly as a tensor product state. Yet, *finding* the satisfying assignment efficiently has proven elusive (indeed, the proof of [17] is *non-constructive*). In other words, we have a trivial NP *decision* problem whose analogous *search* version is not known to be efficiently solvable (see, e.g., [2] regarding the longstanding open question of decision versus search complexity for NP problems). This is the starting point of the present work.

Results and techniques. Our results fall under three directions, all of which are related to k-QSAT with matchings. For this, we first define Quantum k-SAT (k-QSAT) [3] and the notion of a system of distinct representatives (SDR). For k-QSAT, the input is a two-tuple $\Pi = (\{\Pi_i = |\psi_i\rangle\langle\psi_i|\}_i, \alpha)$ of rank 1 projectors or *clauses* $\Pi_i \in \mathcal{L}(\mathbb{C}^2)^{\otimes k}$, each acting non-trivially on a set of k (out of n) qubits, and non-negative real number $\alpha > 1/p(n)$ for some fixed polynomial p. The output is to decide whether there exists a satisfying assignment on n qubits $|\psi\rangle \in (\mathbb{C}^2)^{\otimes n}$, i.e. to distinguish between the cases $\Pi_i|\psi\rangle = 0$ for all i (YES case), or whether $\langle\psi| \sum_i \Pi_i |\psi\rangle \geq \alpha$ (NO case). Note that k-QSAT generalizes k-SAT. As for a *system of distinct representatives (SDR)* (see, e.g., [12]), given a set system such as a hypergraph $G = (V, E)$, an SDR is a set of vertices $V' \subseteq V$ such that each edge in $e \in E$ is paired with a distinct vertex $v_e \in V'$ such that $v_e \in e$. In previous work on QSAT, an SDR has been referred to as a "dimer covering" [17].

1. Quantum k-SAT with bounded occurrence of variables. Our first result concerns the natural restriction of limiting the number of times a variable can appear in a clause. For example, 3-SAT with at most 3 occurrences per variable is NP-hard. We complement this as follows.

▶ **Theorem 1.** *There exists a polynomial time classical algorithm which, given an instance* Π *of k-QSAT in which each variable occurs in at most two clauses, outputs a satisfying product state if* Π *is satisfiable, and otherwise rejects. Moreover, the algorithm works for clauses ranging from* 1*-local to k-local in size.*

To show this, our idea is to "partially reduce" the k-QSAT instance to a 2-QSAT instance. We then use the transfer matrix techniques of [3, 18, 4] (particularly the notion of *chain reactions* from [4]), along with a new notion of "fusing" chain reactions, to deal with the remaining clauses of locality at least 3 in the instance.

Although this setting seems unrelated to the open question of computing solutions to k-QSAT instances with SDRs, we show the following. Denote the *interaction hypergraph* $G = (V, E)$ of a k-QSAT instance as a k-uniform hypergraph (i.e. all edges have size precisely k), in which the vertices correspond to qubits, and each clause c acting on a set of k qubits S_c, is represented by a hyperedge of size k containing the vertices corresponding to S_c.

▶ **Theorem 2.** *Let $G = (V, E)$ be a hypergraph with all hyperedges of size at least 2, and such that each vertex has degree at most 2. Then, G has an SDR.*

Thus, Theorem 1 resolves the open question of [17] for k-QSAT instances with SDRs in which (1) each variable occurs in at most two clauses and (2) there are no 1-local clauses. ((2) is necessary, as allowing edges of size 1 easily makes Theorem 2 false in general.)

2. On parameterized complexity for Quantum k-SAT. Our next result, and the main contribution of this paper, gives a parameterized algorithm[3] for explicitly computing (product state) solutions for a non-trivial class of k-QSAT instances. As discussed in Section 3, this algorithm in some cases provides an *exponential* speedup over brute force diagonalization.

At the core of the algorithm is a new graph theoretic notion of *transfer filtration of type b* for a k-uniform hypergraph $G = (V, E)$, for fixed $b > 0$. Intuitively, one should think of b as denoting the size of a set of b qubits which form the hard "foundation"' of any k-QSAT instance on G. With the notion of transfer filtration in hand, our framework for attacking k-QSAT can be sketched at a high level as follows.

1. First, given a k-QSAT instance Π on G with transfer filtration of type b, we "blow-up" Π to a larger, *decoupled* instance Π^+ (Decoupling Lemma, Lemma 9). The decoupled nature of Π^+ makes it "easier" to solve (Transfer Lemma, Lemma 17), in that any assignment to the b "foundation" qubits can be *extended* to a solution to all of Π^+. This raises the question – how does one map the solution of Π^+ back to a solution of Π?
2. We next give a set of "qualifier" constraints $\{h_s\}$ (Qualifier Lemma, Lemma 19) acting on *only the b foundation qubits*, with the following strong property: If a (product state) assignment $\mathbf{v}$ to the b foundation qubits satisfies the constraints $\{h_s\}$, then not only can we extend $\mathbf{v}$ via the Transfer Lemma to a full solution for Π^+ as in Step 1 above, but we can also map this extended solution back to one for the *original* k-QSAT instance Π.

Once the framework above is developed, we show that it applies to the non-trivial family of k-QSAT instances whose k-uniform hypergraph $G = (V, E)$ has a transfer filtration of type $b = |V| - |E| + 1$. This family includes, e.g., the semi-cycle, tiling of the torus, and "fir tree" (full version). Our main result (Theorem 23) says the following: For any k-QSAT instance Π on such a G and whose constraints are generic (see Section 3), computing a (product state) solution to Π reduces to solving for a root of a *single univariate* (see Remark 25) polynomial P – *any* such root (which always exists if the field $\mathbb{K}$ is algebraically closed) can then be extended back to a full solution for Π.

The key advantage of this approach, and what makes it a parameterized algorithm, is the following – the *degree* of P, and hence the runtime of the algorithm, scale exponentially only in b and a "radius" parameter r of the transfer filtration. Thus, given a transfer filtration

[3] Roughly, parameterized complexity characterizes the complexity of computational problems with respect to specific *parameters* of interest other than just the input size (e.g. the treewidth of the input graph).

where b and r are at most logarithmic, finding a (product state) solution to k-QSAT reduces to solving for a single root over $\mathbb{C}$ for a single univariate polynomial h_1 of polynomial degree, which can be done in polynomial time [25, 24]. Indeed, in Section 3 we give a non-trivial family of k-uniform hypergraphs, denoted Crash, for which our algorithm runs in polynomial time, whereas brute force diagonalization would require exponential time.

Conveniently, even when the foundation b and radius r are superlogarithmic, our algorithm still gives a *constructive* proof that all k-QSAT instances satisfying the preconditions of Theorem 23 have a (product state) solution. In particular, in Corollary 27, we observe that such hypergraphs must have SDRs, and so we constructively reproduce the result of [17] that any 3-QSAT instance with an SDR is satisfiable (by a product state) (again, assuming the additional conditions of Theorem 23 are met).

Finally, although this result stems primarily from tools of projective algebraic geometry (AG), the presentation herein avoids any explicit mention of AG terminology (with the exception of defining the term "generic" in Section 3.3) to be accessible to readers without an AG background. A brief overview of the ideas in AG terms is given in the full version.

3. A study of 3*-uniform hypergraphs with SDRs.* Our final contribution, which we hope guides future studies on the topic, is to take steps towards understanding the structure of all 3-QSAT instances with SDRs, particularly when $|E| = |V|$. Unfortunately, this seems a difficult task (if not potentially impossible, see "finite characterization" comments below). We first give various characterizations involving intersecting families (each pair of edges has non-empty intersection). We then study *linear* hypergraphs (each pair of edges intersects in at most one vertex), which are generally more complex. (For example, the set of edge-intersection graphs of 3-uniform linear hypergraphs is known not to have a "finite" characterization in terms of a finite list of forbidden induced subgraphs [19].) We study "extreme cases" of linear hypergraphs with SDRs, such as the Fano plane and "tiling of the torus", and in contrast to these two examples, demonstrate a (somewhat involved) linear hypergraph we call the iCycle which also satisfies the *Helly property* (which generalizes the notion of "triangle-free"). A main conclusion of this study is that even with multiple additional restrictions in place (e.g. linear, Helly), the set of 3-uniform hypergraphs with SDRs remains non-trivial. To complement these results, we show how to fairly systematically construct large linear hypergraphs with $|E| = |V|$ without SDRs. We hope this work highlights the potential complexity involved in dealing with even the "simple" case of 3-QSAT with SDRs.

Discussion, previous work and open questions. Regarding our parameterized algorithm, our notions of transfer filtrations and blow-ups apply to *any* instance of k-QSAT (and thus also[4] k-SAT), including QMA_1-complete instances. (For example, every k-uniform hypergraph has a trivial foundation obtained by iteratively removing vertices until the resulting set contains no edges. A key question is how *small* the foundation and radius of the filtration can be chosen for a given hypergraph, as our algorithm's runtime scales exponentially in these parameters.) More precisely, our techniques in Section 3, up to and including the Qualifier Lemma, apply to arbitrary k-QSAT instances. The main question is when *local* solutions to the qualifier constraints (which act only on b out of n qubits) can

[4] For the special case of k-SAT, note that it is not *a priori* clear that having a transfer filtration with a small foundation suffices to solve the system trivially. This is because the genericity assumption on constraints, which k-SAT constraints do not satisfy, is required to ensure that any assignment to the foundation propagates to all bits in the instance. Thus, the brute force approach of iterating through all 2^b assignments to the foundation does not obviously succeed.

be extended to *global* solutions to the entire k-QSAT instance. We answer this question affirmatively for the non-trivial class of k-QSAT instances which satisfy the preconditions of Theorem 23 (e.g. the semi-cycle, fir tree, crash, and any k-uniform hypergraph with $b = |V| - |E| + 1$), obtaining polynomial to exponential speedups over brute force in Section 3.

Moving to previous work, Quantum k-SAT was introduced by Bravyi [3], who gave an efficient (quartic time) algorithm for 2-QSAT, and showed that 4-QSAT is QMA_1-complete. Subsequently, Gosset and Nagaj [10] showed that Quantum 3-SAT is also QMA_1-complete, and independently and concurrently, Arad, Santha, Sundaram, Zhang [1] and de Beaudrap, Gharibian [4] gave linear time algorithms for 2-QSAT. The original inspiration for this paper was the work of Laumann, Läuchli, Moessner, Scardicchio and Sondhi [17], which showed *existence* of a product state solution for any k-QSAT instance with an SDR. Thus, the decision version of k-QSAT with SDRs is in NP and trivially efficiently solvable. However, whether the search version (i.e. compute an explicit satisfying assignment) is also in P remains open. The question of whether the decision and search complexities of NP problems are the same is a longstanding open problem in complexity theory; conditional results separating the two are known (see e.g. Bellare and Goldwasser [2]).

Regarding *classical* k-SAT, as mentioned above, in contrast to k-QSAT, solutions to k-SAT instances with an SDR can be trivially computed. As for parameterized complexity, classically it is a well-established field of study (see, e.g., [5] for an overview). The parameterized complexity of SAT and #SAT, in particular, has been studied by a number of works, such as [26, 6, 23, 7, 22, 21, 8], which consider parameterizations including based on tree-width, modular tree-width, branch-width, clique-width, rank-width, and incidence graphs which are interval bipartite graphs. Regarding parameterized complexity of *Quantum* SAT, as far as we are aware, our work is the first to initiate a "formal" study of the subject; however, we should be clear that existing works in Quantum Hamiltonian Complexity [20, 9] have long implicitly used "parameterized" ideas (e.g. in tensor network contraction, the bond dimension can be viewed as a parameter constraining the complexity of the contraction).

We close with open questions. Which ideas from classical parameterized complexity be generalized to the quantum setting? We develop a number of tools for studying Quantum SAT – can these be applied in more general settings, for example beyond the families of k-QSAT instances considered in Theorem 23? The "parameter" in our results of Section 3 involves the radius of a transfer filtration – whether a transfer filtration (of a fixed type b) of *minimum* radius can be computed efficiently, however, is left open for future work. Similarly, it is not clear that given $b \in \mathbb{N}$, the problem of deciding whether a given hypergraph G has a transfer filtration of type at most b is in P. We conjecture this latter problem is NP-complete. Finally, the question of whether solutions to arbitrary instances of k-QSAT with SDRs can be computed efficiently (recall they are guaranteed to exist [17]) remains open.

Organization. Section 2 gives an efficient algorithm for 3-QSAT with bounded occurrence of variables, and introduces the notion of transfer matrices (which are generalized via transfer functions in Section 3). Our main result is given in Section 3, and concerns a new parameterized complexity-type approach for solving k-QSAT. Our structural graph theoretic study of hypergraphs with SDRs, and any omitted proofs, are deferred to the full version.

Notation and basic definitions. For complex Euclidean space $\mathcal{X}$, $\mathcal{L}(\mathcal{X})$ denotes the set of linear operators mapping $\mathcal{X}$ to itself. For unit vector $|\psi\rangle \in \mathbb{C}^2$, the unique orthogonal unit vector (up to phase) is denoted $|\psi^\perp\rangle$, i.e. $\langle\psi|\psi^\perp\rangle = 0$.

▶ **Definition 3** (Hypergraph)**.** A *hypergraph* is a pair $G = (V, E)$ of a set V (vertices), and a family E (edges) of subsets of V. If each vertex has degree d, we say G is d-regular. When convenient we use $V(G)$ and $E(G)$ to denote the vertex and edge sets of G, respectively. We say G is k-uniform if all edges have size k.

▶ **Definition 4** (Cycle, Semicycle, Chain [14])**.** A k-uniform hypergraph $G = (V, E)$ is a *cycle* if there exists a sequence $S = (v_1, v_2, ..., v_l) \in V^l$ for $l \geq n$ such that (1) $v \in S$ for all $v \in V$, (2) for all $1 \leq i \leq l$, $e_i = \{v_i, v_{i+1}, ..., v_{i+k-1}\}$ are distinct edges in E, where indices are understood modularly. The length of the cycle G is $m = l$. If instead $1 \leq i \leq l - k + 1$ and $v_1 = v_l$ ($v_1 \neq v_l$), we obtain a *semicycle* (*chain*) of length $m = l - k + 1$.

2 Quantum SAT with bounded occurrence of variables

Transfer matrices, chain reactions, and cycle matrices. To study 3-QSAT with each qubit occurring in at most two constraints, we first recall transfer matrix tools from the study of 2-QSAT [3, 18, 4]. For any rank-1 constraint $\Pi_i = |\psi\rangle\langle\psi| \in \mathcal{L}((\mathbb{C}^2)^{\otimes k})$, consider Schmidt decomposition $|\psi\rangle = \alpha|a_0\rangle|b_0\rangle + \beta|a_1\rangle|b_1\rangle$, where $|a_i\rangle \in (\mathbb{C}^2)^{\otimes(k-1)}$ lives in the Hilbert space of the first $k-1$ qubits and $|b_i\rangle \in \mathbb{C}^2$ the last qubit. Then, the *transfer* matrix $T_\psi : (\mathbb{C}^2)^{\otimes k-1} \mapsto \mathbb{C}^2$ is given by $T_\psi = \beta|b_0\rangle\langle a_1| - \alpha|b_1\rangle\langle a_0|$. In words, given any assignment $|\phi\rangle$ to the first $k-1$ qubits, if $T_\psi|\phi\rangle \in \mathbb{C}^2$ is non-zero, then it is the *unique* assignment to qubit k (given $|\phi\rangle$ on qubits 1 to $k-1$) which satisfies Π_i.

In the special case of $k = 2$, transfer matrices are particularly useful. Consider first a 2-QSAT interaction graph (which is a 2-uniform hypergraph, or just a graph) $G = (V, E)$ which is a path, i.e. a sequence of edges $e_1 = (v_1, v_2), e_2 = (v_2, v_3), \ldots, e_m = (v_{m-1}, v_m)$ for distinct $v_i \in V$, and where edge e_i corresponds to constraint $|\psi_i\rangle$. Then, any assignment $|\phi\rangle \in \mathbb{C}^2$ to qubit 1 *induces a chain reaction (CR)* in G, meaning qubit 2 is assigned $T_{\psi_1}|\phi\rangle$, qubit 3 is assigned $T_{\psi_2}T_{\psi_1}|\phi\rangle$, and so forth. If this CR terminates before all qubits labelled by V receive an assignment, which occurs if $T_{\psi_i}|\phi'\rangle = 0$ for some i, this means that constraint i (acting on qubits i and $i+1$) is satisfied by the assignment $|\phi'\rangle$ to qubit i alone, and no residual constraint is imposed on qubit $i+1$. Thus, the graph G is reduced to a path $e_{i+1}, \ldots, e_m$. In this case, we say the CR is *broken*. Note that if G is a path, then it is a satisfiable 2-QSAT instance with a product state solution.

Finally, consider a 2-QSAT instance whose interaction graph G is a cycle $C = (v_1, \ldots, v_{m+1})$ with m. Then, a CR induced on vertex v_1 with any assignment $|\psi\rangle \in \mathbb{C}^2$ will in general propagate around the cycle and impose a consistency constraint on v_1. Formally, denote $T_C = T_{\psi_m} \cdots T_{\psi_1} \in \mathcal{L}(\mathbb{C}^2)$ as the *cycle matrix* of C. Then, if the cycle matrix is not the zero matrix, it be shown that the satisfying assignments for the cycle are precisely the eigenvectors of T_C. (If $T_C = 0$, any assignment on v_1 will only propagate partially around the cycle, thus decoupling the cycle into two paths.) Thus, if G is a cycle, it has a product state solution.

Here, when we refer to "solving the path or cycle", we mean applying the transfer matrix techniques above to efficiently compute a product state solution to the path or cycle.

***k*-QSAT with bounded occurence of variables.** We now prove Theorem 1.

Proof of Theorem 1. We begin by setting terminology. Let Π be an instance of k-QSAT with k-uniform interaction graph $G = (V, E)$. For any clause c, let Q_c denote the set of qubits acted on c, i.e. Q_c is the edge in G representing c. We say c is *stacked* if Q_c is contained in another clause $Q_{c'}$, i.e. if $\exists c' \neq c$ such that $Q_c \subseteq Q_{c'}$. For a qubit v, we use shorthand $|v\rangle$ to

denote the current assignment from $\mathbb{C}^2$ to v. For a clause c, $|c\rangle$ denotes the bad subspace of c, i.e. clause c is given by rank-1 projector $I - |c\rangle\langle c|$. The set of clauses vertex v appears in is denoted C_v. For any assignment $|v\rangle$, let $S_{|v\rangle} = \{\langle v|c\rangle \mid c \in C_v\} \subseteq \bigcup_{i=0}^{k-1} \mathbb{C}^{2^i}$, where recall c can be a clause on $1, \ldots, k$ qubits, and we assume $\langle v|$ acts as the identity on the qubits of c which are not v. Thus, $S_{|v\rangle}$ is the set of constraints we obtain by taking the clauses in C_v, and projecting down qubit v in each clause onto assignment $|v\rangle$ (i.e. clauses in $S_{|v\rangle}$ do not act on v). Our algorithm will satisfy that the only possible element of $\mathbb{C}$ in $S_{|v\rangle}$ is 0, obtained by projecting a constraint $|c\rangle \in \mathbb{C}^2$ onto its orthogonal complement to satisfy it; thus, assume without loss of generality that $S_{|v\rangle} \subseteq \bigcup_{i=1}^{k-1} \mathbb{C}^{2^i}$. Finally, two 1-local clauses $|c\rangle, |c'\rangle \in \mathbb{C}^2$ *conflict* if $|c\rangle$ and $|c'\rangle$ are linearly independent.

Algorithm A. Let Π satisfy the conditions of our claim. We repeatedly "partially reduce" Π to a 2-QSAT instance, and use the transfer matrix techniques outlined above to solve this subproblem. Combining this with a new notion of fusing CRs, the technique can be applied iteratively to reduce k-local constraints to 2-local ones until the entire instance is solved. Note: If a CR on a path is broken by a transfer matrix T_ψ on edge (u, v), i.e. $T_\psi|u\rangle = 0$, we implicitly continue by choosing assignment $|0\rangle$ on v to induce a new CR on the path.

1. While there exists a 1-local constraint c acting on some qubit v:
   - **a.** If c conflicts with another 1-local clause on v, reject. Else, set $|v\rangle = |c^\perp\rangle \in \mathbb{C}^2$. Set[5] $C_v = S_{|v\rangle}$, and remove v from Π.
2. While there exists a qubit v appearing only in clauses of size at least $k' \geq 3$:
   - **a.** Set $|v\rangle = |0\rangle$ and $C_v = S_{|v\rangle}$. Remove v from Π.
3. While there exists a 2-local clause:
   - **a.** If there exists a stacked 2-local clause c, i.e. $c' \neq c$ such that $Q_c \subseteq Q_{c'}$:
     - **i.** If $Q_c = Q_{c'}$, remove the qubits c acts on, and set their values to satisfy c and c'.
     - **ii.** Else, $Q_c \subset Q_{c'}$. Thus, c' is k'-local for $3 \leq k' \leq k$. Set the values of the qubits in Q_c so as to satisfy c. This collapses c' to a $(k'-2)$-local constraint on $Q_{c'} \setminus Q_c$.
       - **A.** If $k' - 2 = 1$, then c' has been collapsed to a 1-local constraint on some vertex $v \in Q_{c'} \setminus Q_c$, creating a path rooted at v. Set v so as to satisfy c', and use a CR to solve the resulting path until either the path ends, or a k''-local constraint is hit for $3 \leq k'' \leq k'$. In the latter case (Figure 1, Left), the k''-local constraint is reduced to a $(k''-1)$-local constraint and we return to the beginning of Step 3.
   - **b.** Else, pick an arbitrary 2-local clause c acting on variables v_1 and v_2. Then, v_1 (v_2) is the start of a path h_1 (h_2) (e.g., Figure 1, Middle).
     - **i.** If the path forms a cycle from v_1 to v_2, use the cycle matrix to solve the cycle. Remove the corresponding qubits and clauses from Π.
     - **ii.** Else, set v_1 and v_2 so as to satisfy c. Solve the resulting paths h_1 (h_2) until a k'-local (k''-local) constraint l_1 (l_2) is hit for $3 \leq k' \leq k$ ($3 \leq k'' \leq k$). If both l_1 and l_2 are found:
       - **A.** If $l_1 = l_2$ (i.e. $k' = k''$) and $k' - 2 = 1$, then fuse the paths h_1 and h_2 into a new path beginning at the qubit in l_1 which is not in h_1 or h_2 (Figure 1, Right). Iteratively solve the resulting path until a k'-local constraint is hit for $3 \leq k' \leq k$.
4. If any qubits are unassigned, set their values to $|0\rangle$.

In the full version, we prove correctness, run algorithm A on a sample input, and discuss its general applicability to an entire family of non-trivial 3-QSAT instances. ◀

[5] Note there is one "global copy" of each clause c that is "shared" by all C_v.

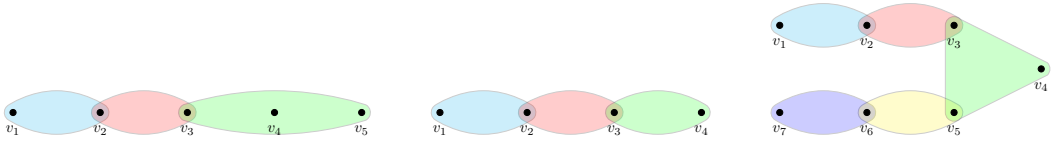


Figure 1 (Left) Solving the path rooted at v_1 via CR satisfies clauses (v_1, v_2) and (v_2, v_3), and projects clause (v_3, v_4, v_5) onto a 2-local residual clause on (v_4, v_5). The CR then stops. (Middle) Letting c denote the clause on (v_2, v_3), v_2 is the start of path $(v_2, v_1, \ldots)$, and v_3 is the start of path $(v_3, v_4, \ldots)$. (Right) Inducing CRs on v_1 and v_7, we assign values to v_3 and v_5. This collapses 3-local clause (v_3, v_4, v_5) into a 1-local clause on v_4 with a unique satisfying assignment, which induces a new CR starting at v_4. Thus, two CR's are "fused" into one CR.

3 Quantum SAT and parameterized algorithms

We next develop a parameterized algorithm for computing an explicit (product state) solution to a non-trivial class of k-QSAT instances (Theorem 23). Although the inspiration stems from algebraic geometry (AG), we generally avoid AG terminology to increase accessibility (see the full version for an overview in AG terms).

3.1 The transfer type of a hypergraph

▶ **Definition 5.** A hypergraph $G = (V, E)$ is of *transfer type* b if there exists a chain of subhypergraphs (denoted a *transfer filtration of type* b) $G_0 \subseteq G_1 \subseteq \cdots \subseteq G_m = G$ and an ordering of the edges $E(G) = \{E_1, \ldots, E_m\}$ such that

1. $E(G_i) = \{E_1, \ldots, E_i\}$ for each $i \in \{0, \ldots, m\}$,
2. $|V(G_i)| \leq |V(G_{i-1})| + 1$ for each $i \in \{1, \ldots, m\}$,
3. if $|V(G_i)| = |V(G_{i-1})| + 1$, then $V(G_i) \setminus V(G_{i-1}) \subseteq E_i$,
4. $|V(G_0)| = b$, where we call $V(G_0)$ the *foundation*,
5. and each edge of G has at least one vertex not in $V(G_0)$.

In other words, a transfer filtration of type b builds up G iteratively by choosing b vertices as a "foundation", and in each iteration adding precisely one new edge E_i and *at most* one new vertex. If a new vertex is added in iteration i, condition (3) says it must be in edge E_i added in iteration i.

▶ **Example 6** (Running example)**.** We introduce a hypergraph G to serve as a running example in this section. Let $V(G) = \{1, 2, 3, 4\}$ with edges $E_1 = \{1, 2, 3\}$, $E_2 = \{1, 2, 4\}$, $E_3 = \{1, 3, 4\}$ and $E_4 = \{2, 3, 4\}$. By Definition 4, G is a 3-uniform cycle. Consider hypergraphs G_0, G_1, G_2, G_3 such that $V(G_0) = \{1, 2\}$, $V(G_1) = \{1, 2, 3\}$, $V(G_2) = V(G_3) = V(G_4) = V(G)$, $E(G_0) = \emptyset$ and $E(G_j) = \{E_1, \ldots, E_j\}$ for $j = 1, 2, 3$. Then $G_0 \subseteq G_1 \subseteq G_2 \subseteq G_3 \subseteq G_4 = G$ is a transfer filtration of type 2, G_2 is a chain, and G_3 is a semicycle.

▶ Remark 7. Let G be a hypergraph with transfer filtration $G_0 \subseteq G_1 \subseteq \cdots \subseteq G_m = G$ of type b. Order the edges of G so that $E(G_i) = \{E_1, \ldots, E_i\}$ $\forall i \in \{1, \ldots, m\}$. Since by construction each edge contains at least one vertex not in $V(G_0)$, there exists a function $r : \{1, \ldots, m\} \to \{0, \ldots, m-1\}$ such that $r(i) < i$ and $|E_i \setminus V(G_{r(i)})| = 1$ for all $i \in \{1, \ldots, m\}$.

▶ **Example 8** (Running example)**.** Let G be the 3-uniform cycle of Example 6. Then one can choose $r : \{1, 2, 3, 4\} \to \{0, 1, 2, 3\}$ with $r(1) = r(2) = 0$, $r(3) = 1$ and $r(4) = 1$.

As the first step in our construction, we show how to map any k-uniform hypergraph G of transfer type b to a new k-uniform hypergraph G' of transfer type b whose transfer filtration

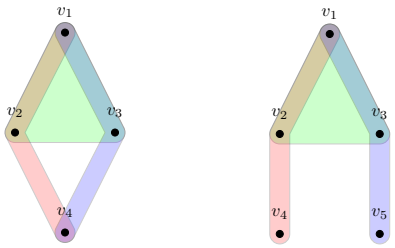


Figure 2 For the hypergraph on the left, consider the transfer filtration with foundation $G_0 = \{v_1, v_2\}$, and we iteratively add edges $\{v_1, v_2, v_3\}$, $\{v_1, v_2, v_4\}$, and $\{v_1, v_3, v_4\}$. The Decoupling Lemma maps this hypergraph to the one on the right, decoupling the intersection on vertex v_4. The surjective function p "undoes" the decoupling by mapping v_1, v_2, v_3 to themselves, and v_4, v_5 to v_4.

must add a vertex in each step (this follows directly from the relationship between $|V(G)|$ and $|E(G)|$ below). This has two effects worth noting: First, G' is guaranteed to have an SDR. Second, it *decouples* certain intersections in the hypergraph, as illustrated in Figure 2. For clarity, in the lemma below, for a function p acting on vertices, we implicitly extend its action to edges in the natural way, i.e. if $e = (v_1, v_2, v_3)$ then $p(e) = (p(v_1), p(v_2), p(v_3))$.

▶ **Lemma 9** (Decoupling lemma). *Given a k-uniform hypergraph G of transfer type b, there exists a k-uniform hypergraph $\widetilde{G}$ of transfer type b with $|E(G)| + b$ vertices and a surjective function $p : V(\widetilde{G}) \to V(G)$ such $p(\widetilde{E}) \in E(G)$ for every $\widetilde{E} \in E(\widetilde{G})$.*

Proof. (Sketch) Let $G_0 \subseteq G_1 \subseteq \cdots \subseteq G_m = G$ be a transfer filtration such that $V(G_0) = \{1, \ldots, b\}$, $E(G_i) = \{E_1, \ldots, E_i\}$ for every $i \geq 1$ and let $r : \{1, \ldots, m\} \to \{0, \ldots, m-1\}$ as in Remark 7. By Remark 7, there is a surjection $p : \{1, \ldots, m+b\} \to \{1, \ldots, n\}$ such that $p(i) = i$ for all $i \in \{1, \ldots, b\}$ and $\{p(i)\} = E_{i-b} \setminus V(G_{r(i-b)})$ for all $i \in \{b+1, \ldots, b+m\}$. For each $j \in \{1, \ldots, m+b\}$, let $\underline{j} = \min(p^{-1}(p(j)))$ and $\widetilde{E}_i = \{i+b\} \cup \{\underline{j} \mid j \in p^{-1}(E_i \setminus p(i+b))\}$ for each $i \in \{1, \ldots, m\}$. Setting $V(\widetilde{G}_i) = \{1, \ldots, b\}$ and $E(\widetilde{G}_i) = \{\widetilde{E}_1, \ldots, \widetilde{E}_i\}$ for each $i = \{0, \ldots, m\}$ we obtain a transfer filtration $\widetilde{G}_0 \subseteq \widetilde{G}_1 \subseteq \cdots \subseteq \widetilde{G}_m = \widetilde{G}$ of type b satisfying the requirements of the claim. ◀

▶ **Example 10** (Running example). Let G be the 3-uniform cycle of Example 6. The proof of Lemma 9 (full version) produces a 3-uniform hypergraph $\widetilde{G}$ with vertices $\{1, 2, 3, 4, 5, 6\}$ and edges $\widetilde{E}_1 = \{1, 2, 3\}$, $\widetilde{E}_2 = \{1, 2, 4\}$, $\widetilde{E}_3 = \{1, 3, 5\}$, $\widetilde{E}_4 = \{2, 3, 6\}$, and surjective function $p : \{1, 2, 3, 4, 5, 6\} \to \{1, 2, 3, 4\}$ defined by $p(1) = 1$, $p(2) = 2$, $p(3) = 3$, $p(4) = p(5) = p(6) = 4$. This choice is not unique: setting $\widetilde{E}_4 = \{2, 4, 6\}$ and $p(6) = 3$ also satisfies Lemma 9.

One of the "parameters" in our parameterized approach will be the *radius* of a transfer filtration, defined next. The concept is reminiscent of radii of graphs, and roughly measures "how far" an edge is from the foundation of b vertices with respect to the filtration.

▶ **Definition 11** (Radius of transfer filtration). Let G be a hypergraph admitting a transfer filtration $G_0 \subseteq \cdots \subseteq G_m = G$ of type b. Consider the function (whose existence is guaranteed by Remark 7) $r : \{0, \ldots, m\} \to \{0, \ldots, m-1\}$ such that $r(0) = 0$ and $r(i)$ is the smallest integer such that $|E_i \setminus V(G_{r(i)})| = 1$ $\forall i \in \{1, \ldots, m\}$. The *radius of the transfer filtration $G_0 \subseteq \cdots \subseteq G_m = G$ of type b* is the smallest integer β such that $r^\beta(i) = 0$ for all $i \in \{1, \ldots, m\}$ (r^β denotes composition of r with itself β times). The *type b radius of G* is the minimum value $\rho(G, b)$ of β over the set of all possible transfer filtrations of type b on G.

▶ **Example 12** (Running example). For G the 4-cycle from Example 6, since function r described in Example 8 is non-constant and $r(r(i)) = 0$ for all $i \in \{1, 2, 3, 4\}$, the transfer filtration of Example 6 has radius $\beta = 2$.

3.2 The main construction

Let W be a two dimensional vector space over a field $\mathbb{K}$. To discuss k-local constraints and product state solutions to k-QSAT instances, we now set up somewhat more general terminology than is standard in the literature. While this level of generality is natural given the geometric nature of our construction, for simplicity one may set $\mathbb{K} = \mathbb{C}$ and identify W with $\mathbb{C}^2$ if desired.

▶ **Definition 13.** A function $H_i : W^n \to \mathbb{K}$ is *k-local* if there exists a subset $E_i = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$ and a nonzero functional $H_i^* : W^{\otimes k} \to \mathbb{K}$ such that $H_i(v_1, \ldots, v_n) = H_i^*(v_{i_1} \otimes \cdots \otimes v_{i_k})$ for all $v_1, \ldots, v_n \in W$, i.e. H_i acts non-trivially only on a subset of k indices. A collection $H = (H_1, \ldots, H_m)$ of k-local functions $H_1, \ldots, H_n : W^n \to \mathbb{K}$ is *k-local*. The corresponding subsets E_i (i.e. on which H_i acts non-trivially) are the edges of a hypergraph G_H with vertices $\{1, \ldots, n\}$, the *interaction graph of H*. The *product satisfiability set* of k-local collection H is the set $\mathcal{S}_H$ of all $(v_1, \ldots, v_n) \in W^n$ such that $v_i \neq 0$ for all $i \in \{1, \ldots, n\}$ and $H_j(v_1, \ldots, v_n) = 0$ for all $j \in \{1, \ldots, m\}$.

▶ Remark 14. Consider an isomorphism $\sharp$ between W and its dual $W^\vee$ that to each $v \in W$ assigns a functional $v^\sharp \in W^\vee$ such that $v^\sharp(v) = 0$. For instance, if a basis $\{w_1, w_2\}$ for W is chosen then we may define $\sharp$ by setting $((a_1w_1 + a_2w_2)^\sharp)(b_1w_1 + b_2w_2) = a_1b_2 - a_2b_1$ for all $a_1, a_2, b_1, b_2 \in \mathbb{K}$. Given any $v_1, v_2 \in W$, $v_1^\sharp(v_2) = 0$ if and only if $\exists \lambda \in \mathbb{K}$ such that $\lambda v_2 = v_1$.

▶ **Definition 15.** For $N \in \mathbb{Z}^+$, the *Fibonacci numbers of order N* are the entries of the sequence $(F_r^{(N)})$ such that $F_r^{(N)} = F_{r-1}^{(N)} + \ldots + F_{r-N}^{(N)}$ for all $r \geq N$, $F_{N-1}^{(N)} = 1$ and $F_r^{(N)} = 0$ for all $r \leq N - 2$. Note that there exists [27] a monotonically increasing sequence (ψ_N) with values in the real interval $[1, 2)$ such that, for each $N \geq 1$, $F_r^{(N)} \sim \psi_N^r$ as $r \to +\infty$.

▶ **Definition 16.** A function f on W^l with values in a $\mathbb{K}$-vector space has *degree* $(d_1, \ldots, d_l)$ if $f(\lambda_1 v_1, \ldots, \lambda_l v_l) = \lambda_1^{d_1} \cdots \lambda_l^{d_l} f(v_1, \ldots, v_l)$ for every $\lambda_1, \ldots, \lambda_l \in \mathbb{K}$ and every $v_1, \ldots, v_l \in W$.

Applying the Decoupling Lemma to an input k-uniform hypergraph G with transfer type b, we obtain a k-uniform hypergraph $\widetilde{G}$ of type b with $m = n - b$, for m and n the number of edges and vertices, respectively. The next lemma shows that $\widetilde{G}$ is "easier to solve", in that any global (product) solution to the k-QSAT system can be derived from a set of assignments to the b foundation vertices, and conversely, any (product) assignment to the latter can be extended to a global (product) solution.

▶ **Lemma 17** (Transfer Lemma). *Let $H = (H_1, \ldots, H_{n-b})$ be a k-local collection of functions $H_i : W^n \to \mathbb{K}$ whose interaction graph is a k-uniform hypergraph of transfer type b. There exist non-zero (non-constant) functions, "transfer functions" $g_1, \ldots, g_n : W^b \to W$, s.t.:*

1. *(Global to local assignments) If $(v_1, \ldots, v_n) \in \mathcal{S}_H$ (recall $v_i \neq 0$ by definition of $\mathcal{S}_H$) there exist nonzero $\lambda_1, \ldots, \lambda_n \in \mathbb{K}$ such that, $\forall i \in \{1, \ldots, n\}$, $\lambda_i v_i = g_i(v_1, \ldots, v_b)$.*
2. *(Local to global assignments) For any nonzero $v_1, \ldots, v_b \in W$ there exist $v_{b+1}, \ldots, v_n \in W$ such that $(v_1, \ldots, v_n) \in \mathcal{S}_H$ and $v_i = g_i(v_1, \ldots, v_b)$ for every i such that $g_i(v_1, \ldots, v_b) \neq 0$.*
3. *(Degree bounds) g_i has degree $(d_{i1}, \ldots, d_{ib})$ such that $d_{ij} \leq F_i^{(b)}$ for all $j \in \{1, \ldots, b\}$.*

Proof. (Sketch) We sketch the proof in the case $b = 2$ and $k = 3$. Define $g_1(v_1, v_2) = v_1$ and $g_2(v_1, v_2) = v_2$. Assume $G_0 \subseteq G_1 \subseteq \cdots \subseteq G_{n-2} = G_H$ is a transfer filtration of type b, $V(G_i) = \{1, \ldots, i + 2\}$ for all $i \in \{1, \ldots, n - 2\}$. Assume $E(G_i) = \{E_1, \ldots, E_i\}$ with

$E_i = \{i, i_1, i_2\}$ for some $i_1, i_2 < i$. We construct transfer functions inductively as follows. First define $(g_i^\sharp(v_1, v_2))(v) = H^*_{i-2}(g_{i_1}(v_1, v_2) \otimes g_{i_2}(v_1, v_2) \otimes v)$ for all $v_1, v_2, v \in W$. Then, given an isomorphism $\sharp$ between W and $W^\vee$ as in Remark 14, define $g_i : W^2 \to W$ such that $(g_i(v_1, v_2))^\sharp = g_i^\sharp(v_1, v_b)$ for all $v_1, v_2 \in W$. The properties of transfer functions stated in the lemma are proved by straightforward induction. We leave the details to the reader. ◀

▶ **Example 18** (Running example). Let $H = (H_1, H_2, H_3, H_4)$ be a 3-local collection of functions $H_i : W^6 \to \mathbb{K}$ whose interaction graph is the 3-uniform chain $\widetilde{G}$ described in Example 10 (obtained by plugging the 4-cycle G of Example 6 into the Decoupling Lemma). For clarity, H_i is defined on hyperedge $\widetilde{E}_i$, where the order of vertices in each edge is fixed by the transfer filtration chosen; in particular, use ordering $\widetilde{E}_1 = (1, 2, 3)$, $\widetilde{E}_2 = (1, 2, 4)$, $\widetilde{E}_3 = (1, 3, 5)$, $\widetilde{E}_4 = (2, 4, 6)$ with foundation $\{1, 2\}$. The proof of Lemma 17 constructs transfer functions $g_1, \ldots, g_6 : W^2 \to W$ which give assignments to qubits 1 through 6, respectively, as follows. Fixing a basis $\{w_1, w_2\}$ of W: $g_1(v_1, v_2) = v_1, g_2(v_1, v_2) = v_2, g_3(v_1, v_2) = H_1^*(v_1 \otimes v_2 \otimes w_2)w_1 - H_1^*(v_1 \otimes v_2 \otimes w_1)w_2, g_4(v_1, v_2) = H_2^*(v_1 \otimes v_2 \otimes w_2)w_1 - H_2^*(v_1 \otimes v_2 \otimes w_1)w_2, g_5(v_1, v_2) = H_3^*(v_1 \otimes g_3(v_1, v_2) \otimes w_2)w_1 - H_3^*(v_1 \otimes g_3(v_1, v_2) \otimes w_1)w_2, g_6(v_1, v_2) = H_4^*(v_2 \otimes g_4(v_1, v_2) \otimes w_2)w_1 - H_4^*(v_2 \otimes g_4(v_1, v_2) \otimes w_1)w_2$.

Thus far, we have seen how combining the Decoupling and Transfer Lemmas "blows up" an input k-QSAT system Π to a larger "decoupled" system Π^+ which is easier to solve due to its decoupled property. Now we wish to relate the solutions of Π^+ *back* to Π. This is accomplished by the next lemma, which introduces a set of "qualifier" constraints $\{h_s\}$ with the key property: Any solution to $\{h_s\}$ can be extended to one for Π^+, and then *mapped back* to a solution for Π. Importantly, the qualifier constraints *act only* on the b foundation vertices, as opposed to all n vertices!

▶ **Lemma 19** (Qualifier Lemma). *Let $H = (H_1, \ldots, H_m)$ be a k-local collection of functions $H_i : W^n \to \mathbb{K}$ whose interaction graph is a k-uniform hypergraph of transfer type b such that $m > n - b$. Then there exist non-zero (non-constant) functions, called* qualifiers, *$h_1, \ldots, h_{m-n+b} : W^b \to \mathbb{K}$ and $\pi : W^n \to W^b$ such that*

1. *$h_s(\pi(\mathcal{S}_H)) = 0$ for all $s \in \{1, \ldots, m - n + b\}$;*
2. *h_s has degree $(d_{s1}, \ldots, d_{sb})$ with $d_{sr} \leq 2F^{(b)}_{\rho(G,b)+b+1}$ $\forall s \in [m + b]$ and $\forall r \in [b]$.*

Proof. (Sketch) We sketch the proof in the case $b = 2$. Given a transfer filtration $G_0 \subseteq \cdots \subseteq G_m = G_H$ of type 2 and radius $\rho(G_H, 2)$, the Decoupling Lemma yields a hypergraph $\widetilde{G_H}$ and a surjection p. Note that $\widetilde{G_H}$ is the interaction graph of a k-local collection $\widetilde{H} = (\widetilde{H}_1, \ldots, \widetilde{H}_m)$ of functions $\widetilde{H}_i : W^{m+2} \to \mathbb{K}$ such that $\widetilde{H}_i^* = H_i^*$ for each $i \in \{1, \ldots, m\}$. Let $\Delta : W^n \to W^{m+2}$ be such that $\Delta(v_1, \ldots, v_n) = (\widetilde{v}_1, \ldots, \widetilde{v}_{m+2})$, where $\widetilde{v}_i = v_{p(i)}$ for all $i \in \{1, \ldots, m + 2\}$. In particular $(v_1, \ldots, v_n) \in \mathcal{S}_H$ if and only if $\Delta(v_1, \ldots, v_n) \in \mathcal{S}_{\widetilde{H}}$. Applying the Transfer Lemma to $\widetilde{G_H}$ yields transfer functions $g_1, \ldots, g_{m+2} : W^2 \to W$. Borrowing notation from the proof of Lemma 9, let $\{i_1, \ldots, i_{m-n+2}\}$ be the subset of all $i \in \{1, \ldots, m+2\}$ such that $\underline{i} < i$. For each $s \in \{1, \ldots, m-n+2\}$, define qualifier $h_s : W^2 \to \mathbb{K}$ such that $h_s(v_1, v_2) = (g_{i_s}^\sharp(v_1, v_2))(g_{\underline{i_s}}(v_1, v_2))$ for all $v_1, v_2 \in W$. If $(v_1, \ldots, v_n) \in \mathcal{S}_H$, then for every $s \in \{1, \ldots, m - n + 2\}$ there exists $\lambda_{i_s}, \lambda_{\underline{i_s}} \in \mathbb{K}$ such that $\lambda_{i_s} v_{p(i_s)} = g_{i_s}(v_1, v_2)$ and $\lambda_{\underline{i_s}} v_{p(i_s)} = g_{\underline{i_s}}(v_1, v_2)$. Therefore $h_s(v_1, v_2) = \lambda_{i_s}\lambda_{\underline{i_s}} v^\sharp_{p(i_s)}(v_{p(i_s)}) = 0$ for every $s \in \{1, \ldots, m - n + 2\}$. Upon defining π as the composition of Δ with the projection onto the first two entries, this proves the first statement of the lemma. The second statement follows from the Transfer Lemma. ◀

▶ Remark 20. To recap, the construction in the proof of Lemma 19 implies that to solve the k-QSAT instance Π, we: (1) Apply the Decoupling Lemma to blow up Π to decoupled instance Π^+. (2) Apply the transfer functions from the Transfer Lemma to $v_1, \ldots, v_b$ to

obtain a solution on all $m+b$ vertices for Π^+. Crucially, the qualifier constraints ensure that all decoupled copies of a vertex v receive the *same* assignment. (3) Map this solution back to one on n vertices for Π by "merging" decoupled copies of vertices.

▶ **Example 21** (Running example)**.** Let $H = (H_1, H_2, H_3, H_4)$ be a 3-local collection of functions $H_i : W^4 \to \mathbb{K}$ whose interaction graph is the 3-uniform cycle of transfer type 2 from Example 6. If p is chosen as in Example 10, then the two qualifier functions are $h_1(v_1, v_2) = (g_5^\sharp(v_1, v_2))(g_4(v_1, v_2))$ of degree $(3, 2)$ and $h_2(v_1, v_2) = (g_6^\sharp(v_1, v_2))(g_3(v_1, v_2))$ of degree $(2, 3)$, where g_3, g_4, g_5, g_6 so that $d_{sr} \leq 3 \leq 10 = 2F_5^{(2)}$ for each $s, r \in \{1, 2\}$.

3.3 Generic constraints

Remark 20 outlined the high-level strategy for computing a (product-state) solution to an input k-QSAT system Π. For this strategy to work, however, we require an assignment to the foundation of the transfer filtration which (1) satisfies the qualifier functions from the Qualifier Lemma, and (2) causes the transfer functions g_i from the Transfer Lemma to output non-zero vectors. When are (1) and (2) possible? We now answer this question affirmatively for a non-trivial class of k-QSAT instances, assuming constraints are chosen generically.

▶ Remark 22 (Generic constraints). The set of k-local constraints H on k-uniform interaction hypergraph G is canonically identified with the projective variety $\mathcal{X}_G(\mathbb{K}) = (\mathbb{P}^{2^k-1}(\mathbb{K}))^m$. (See also [17].) We say a property holds for the *generic constraint with interaction graph* G if it holds for every k-local constraint on a Zariski open set of $\mathcal{X}_G(\mathbb{K})$. In the important case $\mathbb{K} = \mathbb{C}$, this implies in particular that such a property holds for almost all choices of constraints (with respect to the natural measure on $\mathcal{X}_G(\mathcal{C})$ induced by the Fubini-Study metric).

We now show the main theorem of this section (whose proof requires a few other definitions and a Surjectivity Lemma; see full version). The theorem applies to k-uniform hypergraphs of transfer type $b = n - m + 1$, which includes non-trivial instances such as the semi-cycle and the "fir tree" (full version). In words, the theorem says that for any k-uniform hypergraph of transfer type $b = n - m + 1$ (i.e. there is one qualifier function h_1), if the constraints are chosen generically, then any zero of h_1 is the image under the map π (defined in Qualifier Lemma) of a satisfying assignment to the corresponding k-QSAT instance. The key advantage to this approach is simple: To solve the k-QSAT instance, instead of solving a system of equations, we are reduced to solving for the roots of just one polynomial – h_1. Moreover, if both the foundation size b and the radius of the transfer filtration of G are at most logarithmic in m and n, then h_1 has polynomial degree in m and n.

▶ **Theorem 23.** *Let $\mathbb{K}$ be algebraically closed, and let $\mathcal{F}$ denote the set of k-uniform hypergraphs with n vertices, m edges, and transfer type $b = n - m + 1$. If H is a generic k-local constraint with interaction graph $G \in \mathcal{F}$ and h_1 and π are as in the Qualifier Lemma (Lemma 19), then $(h_1 \circ \pi)^{-1}(0) \cap \mathcal{S}_H$ is nonempty.*

▶ **Example 24** (Running example)**.** We illustrate the proof of Theorem 23 by specializing the construction to the 3-uniform semicycle G_3 from Example 6. Then $\widetilde{G}_3$ is the hypergraph with vertices $\{1, 2, 3, 4, 5\}$ and edges $\widetilde{E}_1 = \{1, 2, 3\}$, $\widetilde{E}_2 = \{1, 2, 4\}$, $\widetilde{E}_3 = \{1, 3, 5\}$. Moreover, the transfer functions $g_1, \ldots, g_5 : W^2 \to W$ can be chosen as in Example 18. Let h_1 be as in Example 21 and suppose $v_1, v_2 \in W$ are such that $h_1(v_1, v_2) = 0$. If none of the $g_i(v_1, v_2)$ are zero, then a solution of the form (v_1, v_2, v_3, v_4) can be found by Remark 20. Else, suppose (say) $g_3(v_1, v_2) = 0$ (generically, only one $g_i(v_1, v_2)$ will be zero in this case) so that v_3 is not constrained by v_1 and v_2. With respect to a fixed basis $\{w', w''\}$ of W, we need to show that v_3

can be chosen in such a way that (v_1, v_2, v_3, v_4), where (according the Transfer Lemma) $v_4 = H_2^*(v_1 \otimes v_2 \otimes w'')w' - H_2^*(v_1 \otimes v_2 \otimes w')w''$, is a solution. The idea is to modify $\widetilde{G_3}$ by removing the edge $\widetilde{E_1}$ and adding the vertex labeled by 3 to the foundation. With this modification, the Transfer Lemma yields $g_5(v_1, v_2, v_3) = H_3^*(v_1 \otimes v_3 \otimes w'')w' - H_3^*(v_1 \otimes v_3 \otimes w')w''$. By the Qualifier Lemma, we conclude that $g_5(v_1, v_2, v_3)$ is a non-zero multiple of v_4 if and only if $H_3^*(v_1 \otimes v_3 \otimes w')H_2^*(v_1 \otimes v_2 \otimes w'') - H_3^*(v_1 \otimes v_3 \otimes w'')H_2^*(v_1 \otimes v_2 \otimes w') = 0$. Introducing a coordinate $v_3 = w' + xw''$, this last condition is equivalent to the vanishing of a polynomial in x. While this particular example the polynomial is linear, it is in general of high degree and the assumption that $\mathbb{K}$ is algebraically closed is required in order to guarantee the existence of a root.

▶ Remark 25 (Reduction to univariate polynomials). Theorem 23 reduces us to solving a single polynomial equation, $h_1(v_1, \ldots, v_b) = 0$, which is multi-variate. In this case, we can reduce it further to a univariate polynomial by fixing arbitrary vectors $w_1, \ldots, w_b \in W$ and $w_b' \in W$ linearly independent from w_b. Then $P(x) = h_1(w_1, \ldots, w_b + xw_b')$ is a *univariate* polynomial in $\mathbb{K}[x]$, which has a root $x \in \mathbb{K}$ since $\mathbb{K}$ is algebraically closed.

▶ Remark 26 (Runtimes, and complexity of solving for roots). By Theorem 23 and Remark 25, solving k-QSAT instances on hypergraphs in $\mathcal{F}$ with generic constraints reduces to solving for the roots of a single univariate polynomial, $P(x) \in \mathbb{K}[x]$. This can be accomplished by combining Theorem 2.7 of [25] and the algorithm of Schönhage [24] (Section 3.4 therein), which yields numerical approximations to all the roots of P within additive inverse exponential error in time exponential only in r and b. More specifically, in the full version, we give an explicit statement of the algorithm and a formal runtime analysis. We find k-QSAT instances with generic constraints and $b = n - m + 1$ require total time at most (for radius r, foundation size b, degree $d \leq 2^{r+b+2}$, m the number of constraints, n the number of qubits, $k \in \Theta(1)$ the locality of the constraints, and p a fixed polynomial which determines the additive accuracy $2^{-p(n)}$ to which we solve for roots of polynomials)

$$O(mn) + O\left(2^{2kb(r+b)}\right) + O\left(d^3 \log d + d^2 \log\left(9^d 2^{p(n)d}\right)\right) + O((r+b)2^{b(r+b+2)}). \qquad (1)$$

Thus, the algorithm is polynomial in m, n, and p, and exponential in k (the locality of the constraints), r (the radius), and b (foundation size).

Before discussing exponential speedups, we tie Theorem 23 back to SDRs:

▶ **Corollary 27.** *If G is a k-uniform hypergraph of transfer type $b = |V(G)| - |E(G)| + 1$, then G has an SDR.*

Thus, Theorem 23 constructively recovers the result of [17] (that any k-QSAT instance with an SDR has a (product-state) solution) in the case when the additional conditions of Theorem 23 are met (recall [17] works on all graphs with an SDR, but is not constructive). More generally, we can prove

▶ **Theorem 28.** *If G is a k-uniform hypergraph of transfer type $b \leq |V(G)| - |E(G)| + k - 1$, then G has an SDR.*

On exponential speedups via Theorem 23. Recall Theorem 23 applies to k-uniform hypergraphs of transfer type $b = n - m + 1$, such as the semicycle. From a parameterized complexity perspective, however, most interesting are hypergraphs for which the foundation size b and filtration radius r satisfy $b, r \in o(n + m)$, for which we might obtain an asymptotic speedup over brute force diagonalization of the Quantum SAT system (note the semicycle has $b \in \Theta(k), r \in \Theta(n)$). In the full version, we discuss the triangular tiling of the torus and

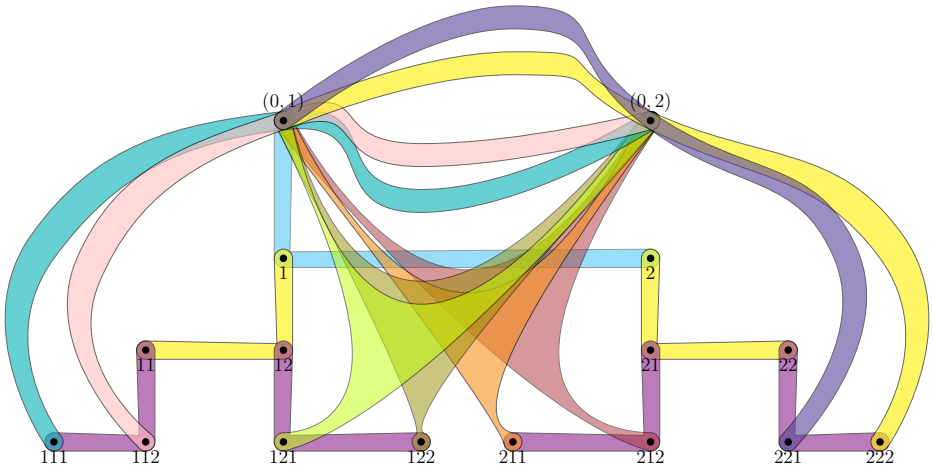


Figure 3 Depiction of 3-uniform crash hypergraph $C_{3,3}$. Generally, $C_{t,k}$ has an exponential separation between the filtration radius and foundation size versus number of vertices and edges.

the fir tree as examples with a quadratic separation $b, r \in \Theta(\sqrt{n} + \sqrt{m})$. (Note that for the runtime of Equation (1), however, a quadratic separation is unfortunately not enough for an asymptotic speedup.) Here, however, we give a hypergraph with a stronger, *exponential*, separation. Namely, we introduce the hypergraph *Crash* (Figure 3), with $r \in \Theta(t)$ and $b \in O(k)$, but $n, m \in \Theta((k-1)^t)$ for $k \geq 3$. On such hypergraphs, our parameterized algorithm hence runs in polynomial time, whereas brute force diagonalization would require time exponential in m and n.

We define k-uniform hypergraph family *Crash*, denoted $C_{t,k}$, as follows. For $k \geq 2$, let $\Sigma = \{1, 2, \ldots, k-1\}$. For $t \geq 1$, $C_{t,k}$ has vertices $V(C_{t,k}) = \bigcup_{j=0}^{t} V_j$ where $V_0 = \{(0, x) \,|\, x \in \Sigma\}$ and $V_j = \Sigma^{t-j+1}$ for all $1 \leq j \leq s$. The edge set of $C_{t,k}$ is the union of all edges of the following three forms:

1. For every $x \in V_1$, $E_x = \{x\} \cup V_0$;
2. for every $2 \leq j \leq t$ and every $x \in V_j$, $E_x = \{x\} \cup \{xa \,|, a \in \Sigma\}$;
3. $E_0 = \{(0, 1)\} \cup V_t$.

Then $C_{t,k}$ has a transfer filtration with foundation V_0 obtained by first adding all the edges E_x with $x \in V_1$, then adding all the edges E_x with $x \in V_2$, etc, with E_0 added last. This transfer filtration has radius t and type $k - 1 = |V(C_{n,k})| - |E(C_{n,k})| + 1$, whereas $|V(C_{n,k})|, |E(C_{n,k})| \in \Theta((k-1)^t)$ (full version).

References

1 Itai Arad, Miklos Santha, Aarthi Sundaram, and Shengyu Zhang. Linear Time Algorithm for Quantum 2SAT. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

2 M. Bellare and S. Goldwasser. The complexity of decision versus search. *SIAM J. Comput.*, 23(1):97–119, 1994.

3 S. Bravyi. Efficient algorithm for a quantum analogue of 2-SAT. Available at arXiv.org e-Print quant-ph/0602108v1, 2006.

4 Niel de Beaudrap and Sevag Gharibian. A Linear Time Algorithm for Quantum 2-SAT. In Ran Raz, editor, *31st Conference on Computational Complexity (CCC 2016)*, volume 50 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:21, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

5 Rodney G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer Publishing Company, Incorporated, 2012.

6 E. Fischer, J.A. Makowsky, and E.V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 156(4):511–529, 2008. Third Haifa Workshop on Interdisciplinary Applications of Graph Theory, Combinatorics & Algorithm.

7 Robert Ganian, Petr Hliněný, and Jan Obdržálek. Better algorithms for satisfiability problems for formulas of bounded rank-width. *Fundam. Inf.*, 123(1):59–76, 2013.

8 Serge Gaspers, Christos H. Papadimitriou, Sigve Hortemo Sæther, and Jan Arne Telle. On satisfiability problems with a linear structure. In *IPEC*, 2016.

9 Sevag Gharibian, Yichen Huang, Zeph Landau, and Seung Woo Shin. Quantum Hamiltonian complexity. *Foundations and Trends® in Theoretical Computer Science*, 10(3):159–282, 2014.

10 D. Gosset and D. Nagaj. Quantum 3-SAT is QMA1-complete. In *Proceedings of the 54th IEEE Symposium on Foundations of Computer Science (FOCS 2013)*, pages 756–765, 2013.

11 L. R. Ford Jr. and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

12 S. Jukna. *Extremal Combinatorics With Applications in Computer Science*. Springer, second edition, 2011.

13 R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. New York: Plenum, 1972.

14 Gyula Y. Katona and Péter G.N. Szabó. Bounds on the number of edges in hypertrees. *Discrete Mathematics*, 339(7):1884–1891, 2016. 7th Cracow Conference on Graph Theory, Rytro 2014.

15 J. Kempe, A. Kitaev, and O. Regev. The complexity of the local Hamiltonian problem. *SIAM Journal on Computing*, 35(5):1070–1097, 2006.

16 A. Kitaev, A. Shen, and M. Vyalyi. *Classical and Quantum Computation*. American Mathematical Society, 2002.

17 C. R. Laumann, A. M. Läuchli, R. Moessner, A. Scardicchio, and S. L. Sondhi. Product, generic, and random generic quantum satisfiability. *Physical Review A*, 81:062345, 2010.

18 C. R. Laumann, R. Moessner, A. Scardicchio, and S. L. Sondhi. Phase transitions and random quantum satisfiability. *Quantum Information & Computation*, 10:1–15, 2010.

19 Ranjan N. Naik, S.B. Rao, S.S. Shrikhande, and N.M. Singhi. Intersection graphs of k-uniform linear hypergraphs. *European Journal of Combinatorics*, 3(2):159–172, 1982.

20 T. J. Osborne. Hamiltonian complexity. *Reports on Progress in Physics*, 75(2):022001, 2012.

21 Daniel Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model counting for cnf formulas of bounded modular treewidth. *Algorithmica*, 76(1):168–194, 2016.

22 Sigve Hortemo Sæther, Jan Arne Telle, and Martin Vatshelle. Solving #sat and maxsat by dynamic programming. *J. Artif. Int. Res.*, 54(1):59–82, sep 2015.

23 Marko Samer and Stefan Szeider. Algorithms for propositional model counting. *Journal of Discrete Algorithms*, 8(1):50–64, 2010.

24 A. Schönhage. Equation solving in terms of computational complexity. In *Proceedings of the International Congress of Mathematicians*, pages 131–153, 1986.

25 Arnold Schönhage. Quasi-GCD computations. *Journal of Complexity*, 1(1):118–137, 1985.

26 Stefan Szeider. *On Fixed-Parameter Tractable Parameterizations of SAT*, pages 188–202. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

27 D. A. Wolfram. Solving generalized fibonacci recurrences. *Fibonacci Quart.*, 36(2):129–145, 1998.

Balanced Connected Partitioning of Unweighted Grid Graphs

Cedric Berenger
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Peter Niebert
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Kevin Perrot
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Abstract

We consider a partitioning problem for grid graphs with special constraints: a (square) grid graph as well as a number of colors is given, a solution is a coloring approximatively assigning the same number of vertices to each color and such that the induced subgraph for each color is connected. In a "rooted" variant, a vertex to be included in the coloring for each color is specified as well. This problem has a concrete motivation in multimedia streaming applications.

We show that the general problem is NP-complete. On the other hand, we define a reasonable easy subclass of grid graphs for which solutions always exist and can be computed by a greedy algorithm.

2012 ACM Subject Classification Mathematics of computing → Paths and connectivity problems, Hardware → Partitioning and floorplanning

Keywords and phrases grid graphs, connected partitioning, NP-completeness, graph algorithm

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.39

Acknowledgements We would like to thank Jérémy Chalopin and Victor Chepoi for their comments on earlier versions of this work.

1 Introduction

We study a particular partitioning problem of (square) grid graphs. Consider a finite grid (a rectangle) and a subset of squares present (shown in black on the second leftmost image of Figure 1). The present squares are vertices of a graph where the edges are implicit by the neighboring relation (leftmost image).

The problem we study is to color/partition such a graph with a given number of colors so that the induced subgraph for each color is connected and that the partition is balanced, i.e. the number of vertices for each color is (almost) the same. In the example of Figure 1, we choose to color with three colors. The third image shows a coloring satisfying both constraints. In the "rooted variant", we additionally specify for each color a root node that has to take that color (fourth image). For this choice, the third image is not a solution, but the fifth image is. Of course, the fifth image is also a solution of the unrooted problem.

The practical motivation for this particular problem concerns (broadcast) streaming in physical networks in multimedia applications (where each square is a screen tile and there may be holes and irregular borders), but also routing problems in NoC [10] (network on a chip) with horizontal and vertical communication links and dead vertices (due to manufacturing

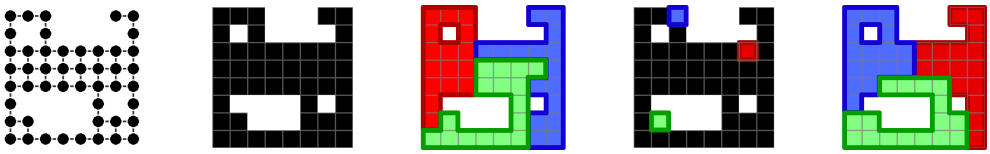


Figure 1 Grid graph and balanced connected coloring.

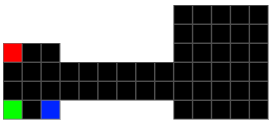


Figure 2 A rooted problem with 3 colors stuck in a corridor of width 2.

issues). For these systems, the limited bandwidth of links can be compensated by partitioning and injecting the data into the network from several "roots". Each root is thus responsible for a subset of vertices and the broadcast messages from distinct roots do not cross.

For the general case, we show that finding a balanced connected partition is NP-complete for both the rooted and the unrooted case (Theorems 1, 2). We achieve the NP-hardness result by a reduction from the Hamiltonian circuit problem on hexagonal graphs [7].

Intuitively, the connectivity of each colored zone is subject to conflicts when there are fewer connections than colors that need to pass. In our reduction, we exploit this by binary conflicts on connections that can only be used by a single color. Beyond binary conflicts, taking the point of view of our grid as a maze, several parallel and adjacent lines of vertices form a corridor of a limited capacity for colors to pass. For instance, in the example of Figure 2, the three roots are connected by a corridor of width 2, hence only two colors can connect to the big zone and obviously, no balanced connected partition exists. In order to define "easy" subclasses of graphs called *q-square connected graphs*, we formalize the idea of a maze with all corridors, whether straight or "diagonal" (zigzaging), large enough to let all colors pass. The condition of q-square connectedness can be verified on bounded windows of the graph and thus decided in linear time. It guarantees that a solution exists. Indeed, we give a greedy polynomial time algorithm for the unrooted case that always finds a solution on such topologies (Theorem 6). We also claim that this simple greedy algorithm can be extended to the rooted case and improved to compute solutions in linear time (for any fixed q).

The definition, intuitively similar but not equivalent to the classical definition of k-connectedness of graphs, is not trivial and is the result of fine-tuning the conditions in order to obtain a correctness proof. However, it is intuitive and may have an interest in itself. We have implemented the algorithm for simulation where it proves to be well behaved (it produces partitions that are "compact" and not unnecessarily intertwined).

Related work. To the best of our knowledge, the hardness of the precise problem studied here, as well as the easy subclass and the greedy algorithm, are unknown.

The much studied Balanced Connected q-Partition Problem (BCP_q), related to our work, is the problem of partitioning a weighted graph $G = (V, E)$ into q connected subgraphs of similar size/weight. In this work, we study this problem for unweighted grid graphs.

The unweighted BCP_q is shown NP-Hard in [4] for bipartite graphs for $q \geq 2$, but the result does not cover grid graphs. It has been proven in [1] that the BCP_q is NP-Hard in $G_{m\times n}$ grid graphs for any $n >= 3$, but this result requires weights on the vertices. That work also gives approximation algorithms considering the relative error under some hypothesis. In

[3] it is shown that finding a solution with an absolute error less than $|V|^{1-\epsilon}$ is NP-Hard even for the unweighted variant of the BCP_q, but on general graphs only. Our NP-hardness proof requires novel ideas and a different construction from previous works.

In [8, 9], polynomial algorithms for BCP_q are given for unweighted q-connected graphs. However grid graphs are 2-connected only. There exist also polynomial approximation results under certain restrictions. In [3] a 4/3-approximation algorithm is shown to exist for BCP_2 without other assumptions. More recently, [11] shows a polynomial time 7/6 approximation algorithm for BCP_2 in grid graphs. In [2], it is shown that there is no approximation algorithm with ratio smaller than 6/5 for arbitrary graphs. The same article also shows that BCP_q is strongly NP-Hard even on q-connected graphs with weights, as well as an inapproximability result for (BCP_2).

With motivations from VLSI design, a lot of work focuses on minimizing the number of transversal edges between parts, not necessarily keeping them connected, e.g. [5], [6].

Outline. The remainder of this article is structured as follows: Section 2 formalizes the model and the problem statement. In Section 3, we recall the Hamiltonian circuit problem for hexagonal graphs and develop the reduction from this problem to show NP-hardness for the general case. Essentially the same reduction can be used for the unrooted and for the rooted case. NP-completeness follows trivially as it is straightforward to verify a solution by a linear algorithm. In Section 4, we define the subclasses of q-square connected graphs by two alternative definitions (one intuitive, the other for algorithmic purposes) and show their equivalence. Then we define a rule based greedy algorithm for the unrooted case for coloring a graph one color after the other. We prove termination and correctness of the algorithm. Then we discuss how this algorithm can be extended to deal with the rooted case. In Section 5, we conclude and give an outlook on future work.

2 Model

We now formalize the notions of square grid graph and the partitioning problem. We simplify the presentation by assuming all square grid graphs embedded in the plane of pairs of integers.

For any finite set of pairs $V \subset \mathbb{Z}^2$ and the induced set of neighboring edges $E = \{((x_1, y_1), (x_2, y_2)) \in V \times V \mid |x_2 - x_1| + |y_2 - y_1| = 1\}$, we call $G_V = (V, E)$ a *square grid graph*. By $|G_V| := |V|$ we denote the *size* of a square grid graph

For simplicity, by grid graphs we will mean square grid graphs in the following (as opposed to hexagonal grid graphs, for instance).

For a subset of vertices $V' \subset V$ of a (grid) graph $G_V = (V, E)$, the graph (V', E') with $E' = E \cap (V' \times V')$ is the induced subgraph of G_V restricted to vertex set V'. Obviously, for a grid graph $G_V = (V, E)$ and a subset of vertices V', the induced subgraph (V', E') equals the grid graph $G_{V'}$.

A graph (V, E) is *connected* if for each pair $u, v \in V$ there exists a sequence $u = v_0, v_1, \ldots, v_l = v$ for some $l \geq 0$ and $(v_i, v_{i+1}) \in E$ for all $0 \leq i < l$.

We will represent grid graphs and induced subgraphs by colored 2D matrices, see Figure 1. An obvious data structure to represent grid graphs is a list of vertices. Alternatively, the *containing rectangle* of a grid graph G_V is defined as $R(V) = \{(x, y) \mid \exists (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4) \in V : x_1 - 1 \leq x \leq x_2 + 1 \text{ and } y_3 - 1 \leq y \leq y_4 + 1\}$, a full matrix containing V and an external border without vertices. The containing rectangle allows for matrix representations of grid graphs d.

A *connected part* $G_{V'}$ of a grid graph G_V is an induced subgraph of G_V that is connected. A *q-partition* $\mathcal{P}$ of a grid graph G_V is a set of q parts G_{V_i} with $1 \leq i \leq q$ such that : $V = \bigcup_{G_{V'} \in \mathcal{P}} V'$ and for any $G_{V_i}, G_{V_j} \in \mathcal{P}$ we have that $i \neq j$ implies $V_i \cap V_j = \emptyset$. We call a partition *connected* if all parts are connected.

For a subset of vertices $R = \{r_1, \ldots, r_q\} \subseteq V$ of a grid graph V_G called *roots*, an *R-rooted* partition of G_V is a partition $\{G_{V_i} \mid r_i \in V_i\}$.

A partition $\mathcal{P}$ of a grid graph G is *perfectly balanced* if and only if for all $G_{V_i} \in \mathcal{P}, |V_i| = \frac{|V|}{|\mathcal{P}|}$. Additionnaly, considering a partition $\mathcal{P}$ of a grid graph G, we say a part $G_{V_i} \in \mathcal{P}$ is *perfectly saturated* if it fits the requirement for P to be perfectly balanced, *i.e.* $|V_i| = \frac{|V|}{|\mathcal{P}|}$. If $|V_i|$ is below/above the *saturation threshold* $\frac{|V|}{|\mathcal{P}|}$, we respectively say that the part is *under/over-saturated*.

Perfect balance may often be an undesirably restrictive condition when it comes to balancing load in networks. In order to relax it, we may specify an error tolerance ratio $0 \leq r \leq 1$ with perfect balance corresponding to $r = 0$. For $0 \leq r \leq 1$ we say that a q-partition $\mathcal{P}$ of a grid graph G_V is *r-balanced* if and only if for each part $G_{V_i} \in \mathcal{P}$ we have $\frac{|G_V|}{q} \cdot (1-r) \leq |G_{V_i}| \leq \frac{|G_V|}{q} \cdot (1+r)$. The same way, we say that a part G_{V_i} of a q-partition $\mathcal{P}$ of a grid graph G_V is *r-saturated* if and only if $\frac{|G_V|}{q} \cdot (1-r) \leq |G_{V_i}| \leq \frac{|G_V|}{q} \cdot (1+r)$. If $|V_i|$ sits outside of the *saturation interval* $\left[\frac{|G_V|}{q} \cdot (1-r), \frac{|G_V|}{q} \cdot (1+r)\right]$, we respectively say that the part is *under/over-r-saturated*. When the ratio r is clear from the context, we will simply say that an area is balanced instead of r-balanced, and that a part is (under/over)saturated instead of (under/over)r-saturated.

The *r-balanced connected partitioning problem* (rBCP) is the problem of finding, for any grid graph G_V and number q, an r-balanced connected partition with q parts.

The *rooted r-balanced connected partitioning problem* (RrBCP) is the problem of finding, for any grid graph G_V and set of roots $R \subseteq V$, an R-rooted r-balanced connected partition.

3 NP-Hardness

In this section we present a polynomial time reduction from the Hamiltonian cycle problem (HCP) on hexagonal grid graphs to the rooted r-balanced connected partitioning problem (rRBCP) on square grid graphs, for $r \leq \frac{1}{7}$. We then argue that the reduction also applies to r-balanced connected partitioning problem (rBCP).

For any finite set $V \subset \mathbb{Z}^2$ of vertices, we construct the *hexagonal grid graph* $H_V = (V, E)$ with the induced set of edges $E = \{((x_1, y_1), (x_2, y_2)) \in V \times V \mid (|x_2 - x_1| = 1 \text{ and } |y_2 - y_1| = 0) \text{ or } (x_2 - x_1 = 0 \text{ and } y_2 - y_1 = 1 \text{ and } x_1 + y_1 \mod 2 \equiv 0)\}$. By $|H_V| := |V|$ we denote the *size* of the hexagonal grid graph. An example is shown on Figure 5 (left).

The *Hamiltonian cycle problem* (HCP) is the following decision problem: given a graph G, does there exist a cycle containing exactly once each vertex of G? This problem has been proven to be NP-complete when restricted to hexagonal grid graphs [7].

Given a hexagonal grid graph for the HCP, we will construct a square grid graph and set of roots for rRBCP. The section is split in four parts. In the first part we give intuitions on some simple rRBCP instances. Then we describe the reduction and concepts at stake. Finally, we demonstrate the correctness of the reduction. The last part explains how to apply the same idea for a reduction to rBCP.

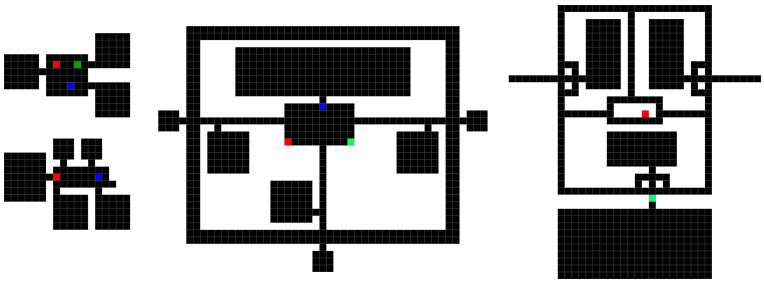


Figure 3 Intuition on the construction: tanks and roots/colors interaction.

3.1 Intuitions on the construction

Let us introduce step by step some straightforward "tricks" used in the reduction.

Consider a large connected region of a square grid graph, uncolored and accessible only via a single vertex or a single path, then this region will own a single color. This is exemplified on Figure 3 (left top): any partitioning must assign the same color to each of the three large outer regions (possibly three different colors, but a single color for each). We call such a region a "tank". In contrast, lines of vertices of width 1 are called "*paths*".

The basis of the construction will be to connect tanks via paths, and choose sizes of tanks such that the total number of vertices used in paths is negligible compared to a single tank.

If the tolerance ratio and the size of tanks are tightly chosen, then coloring a given tank may result in saturating the part corresponding to that color, i.e. a color tied to a tank may not color other tanks without oversaturating. For example, on Figure 3 (left bottom), the red root must color the large tank on the left in any partitioning, nearly saturating that color. Hence, red cannot own any of the two bottom tanks without oversaturating.

Some instances can "trap" parts/colors inside a restricted area. On Figure 3 (middle), blue is tied to a very large tank and must color it nearly saturating that color (for some suitable ratio r). But blue cannot reach the outer ring, because otherwise it would have to cover at least one of the three remaining *inner tanks* and oversaturate.

A color can prevent another color from creating a fork. In Figure 3 at right, the upper rectangle of paths has three access points (left, right, bottom). Green has to color the lower tank, and we suppose that adding any of the three inner tanks would lead to oversaturation. On the other hand, red requires the three inner tanks to saturate. Suppose that green was connected to the two exit points left and right by paths. Whichever the choice of paths taken, red could no longer reach all three inner tanks. In other words, this construction forces a decision for green between the left and the right exit point. This "gadget" will be used as a *macrocell* representing a vertex of the hexagonal grid, and the green color can only connect two of the three exits of any macrocell in order for the partition to be balanced, forcing green on a Hamiltonian path.

3.2 Reduction construction up to calculations

For a given hexagonal grid graph (instance of HCP), we build a square grid graph and set of roots (instance of rRBCP). This instance will admit a solution if and only if the HCP instance admits a solution: the square grid graph mimics the structure of the hexagonal grid, and the part of a specific color will be forced to correspond to a Hamiltonian cycle of the hexagonal grid graph (if one exists, otherwise no r-balanced partioning exists). Our construction will exclusively be composed of tanks and paths.

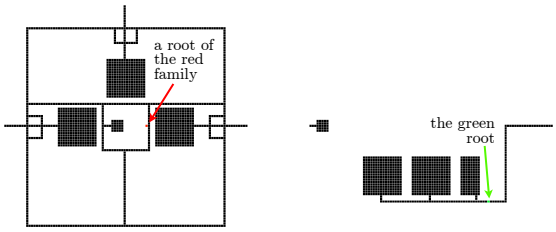


Figure 4 Macrocells with their root vertices for red (left) and green (right) parts.

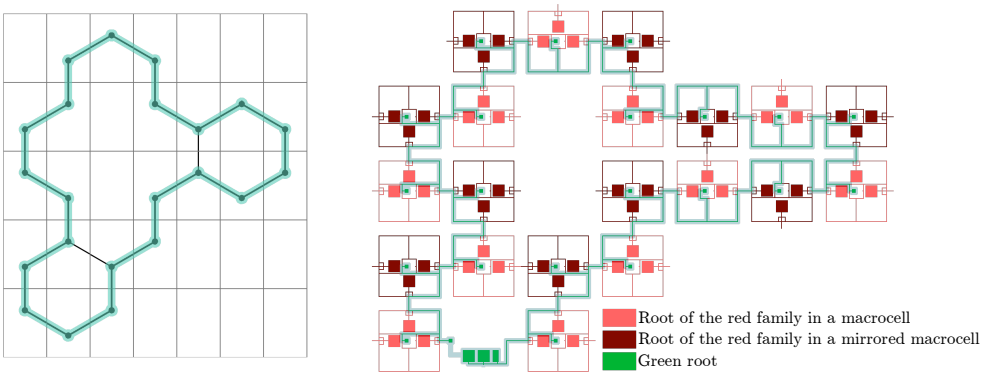


Figure 5 An hexagonal grid graph (left), and the corresponding grid graph construction (right).

Consider a macrocell as Figure 4 (left), which recalls the construction in Figure 3 (right) with an additional tank in the center. As the sizes of the tanks will depend on the hexagonal grid size and on the tolerance ratio r, it is crucial that we can scale up macrocells. Let $T \times T$ be the size of a containing rectangle of such macrocells (such that the three entry paths lie on the center of three sides), let t_i, t_c be the respective sizes of the three inner tanks and of the center tank, and let p be the total number of path vertices (not tank) in one macrocell. It is straightforward to scale up a macrocell so that the number of paths vertices becomes negligible: p grows linearly with T, whereas t_i, t_c grow quadratically.

Let us distinguish a vertex of degree two of the hexagonal grid graph, which we call *primary root* (there always exists one). By using copies of this macrocell or its mirror for each vertex distinct from the primary root, we can mimic the layout of the hexagonal grid graph, as shown on Figure 5 (right). Let m be the number of vertices in the hexagonal grid graph, and let us call *red family* the set of m colors used for roots in these macrocells.

For the primary root vertex (of degree two in the hexagonal grid graph), we replace it with a special macrocell containing (see Figure 4 (right)): on one entry path a tank of size $t_g \approx 3 \cdot \frac{5}{6} t_i$ called *green tank* with a *green* root tied to it, and on the other entry path a tank of size $t_c \approx \frac{t_i}{6m}$. The main idea here is that the green root misses approximately $3 \cdot \frac{1}{6} t_i$ vertices to be balanced (compared to the family of red colors each taking three tanks of size t_i), and will have to take the *central tanks* of size t_c in all macrocells (m times t_c) in order to enter the saturation interval and be r-balanced, hence creating a Hamiltonian path.

A complete construction is shown on Figure 5 (right).

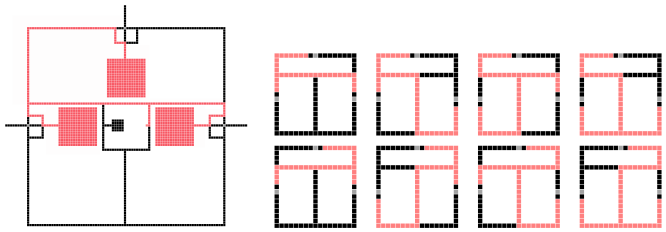


Figure 6 If another color goes across a macrocell, then the red color enforces it to take exactly two out of the three entry paths. In any case red takes the three inner tanks. Detailed schematic of the first possibility on the left, simplified schematics on the right for the eight possibilities.

3.3 Reduction correctness and calculations

Let us recall that $r < \frac{1}{7}$. We now argue about the correctness of a series of facts, leading to the correctness of the reduction.

Figure 6 shows a full case disjunction regarding the fact that macrocells owned by colors in the red family prevent green from creating forks (corresponding to degree three vertices).

▶ **Fact 1.** *If green goes across a macrocell, then it must take exactly two entry paths.*

In other words, green part is constrained by the adversary family of red colors to follow a Hamiltonian cycle.

Now, let us see how to calculate the sizes of tanks and macrocells such that the number of path vertices is negligible compared to the size of any single tank, and that green must take exactly all central tanks of size t_c in each macrocell (and therefore runs across each one).

We introduce variables to specify the construction precisely:

- m: total number of vertices in the hexagonal grid graph,
- n: total number of vertices in the square grid graph,
- k: scaling factor of macrocells,
- t_i: size of each of the three inner tank in each macrocell,
- t_c: size of the central tank in each macrocell (plus one in the primary root macrocell),
- t_g: size of the tank tied to the green root in the primary root macrocell,
- p: number of path vertices (not tank) in one macrocell,
- $s = \frac{n}{m}$: perfect saturation threshold.

By definition, we have the saturation interval ratio $I_s = [s(1-r), s(1+r)]$. We have two kinds of roots (the red family and the green), and four kinds of vertices (path and tanks t_i, t_c, t_g). This gives three groups of inequations to consider. Recall that $t_g \approx 3 \cdot \frac{5}{6} t_i$ and $t_c \approx \frac{t_i}{6m}$.

Path vertices. The *unitary macrocell* shown in Figure 4 (left) permits tanks of maximum size c_t with a minimum of c_p path vertices. By multiplying the dimensions of the unitary macrocell by an integer factor k (except for path wideness), we get a *k-scaled macrocell* permitting tanks of maximum size $k^2 \times c_t$ with less than $k \times c_p$ path vertices (the needed path length to link tanks together grows linearly with k whereas maximum possible tank size grows quadraticaly) :

$$\exists k \in \mathbb{N} : \quad p \geq k \times c_p \quad \text{and} \quad t_i, t_c \leq k^2 \times c_t \quad \text{with } c_p = 639 \text{ and } c_t = 256 \tag{1}$$

We want paths to be negligible and dominated by tanks for balance, meaning that modifying the number of path vertices without modifying the number of tanks in a part has no impact on its saturation. The total number of path vertices is mp.

Red family and inner tanks. We want that each root of the red family reaches the saturation interval with exactly the three inner tanks of its macrocell, *i.e.*

$$2t_i + mp < s(1-r) < 3t_i \quad \text{and} \quad 3t_i + mp < s(1+r) < 4t_i. \tag{2}$$

Red family and green tank. In any partitioning, roots of the red family cannot take the green tank.

Green and all tanks. Green always owns the green tank, and we want the green root to reach the saturation interval with exactly the green tank plus all m central tanks, *i.e.*

$$t_g + (m-1)t_c + mp < s(1-r) \leq t_g + mt_c \text{ and } t_g + mt_c + mp < s(1+r) < t_g + t_i. \tag{3}$$

Red family and central tanks. As we constrained green to take all central tanks, Reds can't take any central tank.

▶ **Fact 2.** *For any rational ratio $r = \frac{a}{b} < \frac{1}{7}$ and any $m > 3 \in \mathbb{N}$, if*

$$t_g = \tfrac{5}{6}s \qquad t_c = \tfrac{s(\frac{1}{6}-\frac{a}{b})}{m} \qquad t_i = ms - (t_g + m(t_c+p)) = \tfrac{(-1+12m^2+12m(-1+\frac{a}{b})+6\frac{a}{b})s}{36(m-1)m}$$
$$p = \tfrac{t_c}{2m} = \tfrac{s(\frac{1}{6}-\frac{a}{b})}{2m^2} \qquad s = 36 \cdot 71 \cdot 10^6 \cdot b \cdot (m-1) \cdot m^3 \qquad k = \tfrac{p}{c_p} = s\tfrac{\frac{1}{6}-\frac{a}{b}}{2 \cdot 639 \cdot m^2}$$

then in any r-balanced connected partitioning green must take exactly all m central tanks (one in each macrocell plus one in the primary root macrocell).

Proof. The statement satisfies Relations (1-3), implying the result (see Appendix A.1). ◀

From Fact 2 in any r-balanced partitioning the green root must go across every macrocell, and from Fact 1 it can do so only via a connected "macropath" of degree two among all m macrocells, which corresponds exactly to a Hamiltonian cycle in the HCP instance. Conversely, a Hamiltonian cycle gives a solution for the green root to do so. In both cases each root of the red family will own the three inner tanks of its macrocell.

This reduction can straightforwardly be done in polynomial time (choose a vertex of degree 2, compute the equations given in Fact 2, and map each vertex of the hexagonal grid graph to its corresponding macrocell) which gives the result.

▶ **Theorem 1.** *rRBCP is NP-hard for any rational ratio $r < \frac{1}{7}$.*

3.4 r-balanced connected partitioning problem (rBCP)

The previously described construction also applies to rBCP (without fixed roots). Indeed, it is enough to notice that when roots are removed, then there are no other solutions to the rBCP instance than those of the rRBCP instance. This is rather straightforward.

The part owning the green tank (previously green, let us now call it *yellow*) must behave the same as green and take all central tanks (it verifies Fact 2). Then, given that yellow must not own any inner tank (Relation (3), these inner tanks must be owned by other parts (previously the red family, let us now call them the *orange family*). It is now enough to notice that each orange part must own three inner tanks belonging to a single macrocell: as green runs across each macrocell to take all central tanks, there is only one remaining entry path in each macrocell, therefore the inner tanks of a macrocell cannot be shared between two parts of the orange family without letting at least one of them under-saturated (Relations (2)). As a consequence Fact 1 also holds and the result follows.

▶ **Theorem 2.** *rBCP is NP-hard for any rational ratio $r < \frac{1}{7}$.*

4 Partitionning q-square Connected Grids

Intuitively, the difficulty of the balanced connected partitioning problem is a consequence of conflicts: if large areas are acessible from a very limited number of paths, you will have to carefully choose which part will cover which area to avoid a part blocking a crucial access.

One might consider as solution to remove these conflicts by assuring that there is always a distinct path for any part to access any area, but the usual graph theoretic notion of k-connectivity is not well suited for square grids where the degree of vertices is limited to 4 in general and which always include vertices with degree less than 2 (e.g. in the corner). Moreover, k-connectivity is a global property of a graph and difficult to verify locally. Hence the need for a new definition.

In the following, we define for each q a subclass of grid graphs called *q-square connected*, which is easy to verify and for which connected balanced q-partitions always exist and which can be efficiently computed. The idea is to make corridors (Figure 2) sufficiently large for all colors to pass. Then a greedy algorithm can expand a connected part (from a root) until it is saturated (reaches $\frac{|G_V|}{q}$ vertices), in such a way that the remaining uncolored graph remains $(q-1)$-square connected. However, while "width" seems obvious for straight "corridors", it is less evident for angles, branching, etc.

4.1 Two definitions of q-square connected graphs

We represent a "square" of a given side length by its lower left corner: for $(x,y) \in \mathbb{Z}^2$, $q > 0$, let $sq(x,y,q) = \{(x+i, y+j) \mid 0 \leq i,j < q\}$ denote *the q-square at (x,y)*. A grid graph G_V is said to *contain* $sq(x,y,q)$ iff $sq(x,y,q) \subseteq V$. A grid graph G_V is *covered by q-squares* if for each $(x,y) \in V$ there exist x', y' such that $sq(x',y',q)$ is contained in G_V and $(x,y) \in sq(x',y',q)$.

Two q-squares $sq(x,y,q), sq(x',y',q)$ are *adjacent* iff $|x-x'| + |y-y'| = 1$. For a set $V \subseteq \mathbb{Z}^2$, let $Sq(V,q) = (V', E')$ such that V' the set of q-squares in V, E' the set of adjacent pairs of q-squares in V, denote the *induced graph of q-squares*. The *distance of two q-squares $sq(x,y,q), sq(x',y',q)$ in $Sq(V,q)$* is denoted by $dist((x,y),(x',y'),V,q)$.

A grid graph G_V covered by q-squares is *q-connected* if for $sq(x,y,q), sq(x',y',q)$ in $Sq(V,q)$ with $\max\{|x-x'|, |y-y'|\} \leq q$ and $\min\{|x-x'|, |y-y'|\} < q$ we have $dist((x,y),(x',y'),V,q) = dist((x,y),(x',y'),\mathbb{Z}^2,q) = |x-x'| + |y-y'|$.

The condition for q-connectedness considers square that are overlapping (their intersection is non-empty) or touching (at least one vertex in one square is adjacent to a vertex in the other square). For such squares, we require the existence of a path of squares that somehow zigzags from one square to the other without changing the direction on either axis.

▶ **Lemma 3.** *Let a connected grid graph G_V be covered by q-squares and q-connected with $q > 1$. Then G_V is also $(q-1)$-connected.*

The proof is given in the appendix. Now we consider an alternative definition that is equivalent, less intuitive, but better suited for the description of the algorithm.

A grid graph G_V is *q-wide* iff for all $v = (x,y) \in V$ there exist x', y' such that $0 \leq x - x' < q$, $0 \leq y - y' < q$ and for all $0 \leq i < q$ we have $(x'+i, y), (x, y'+i) \in V$, i.e. every vertex is part of a horizontal and a vertical segment of q vertices.

A grid graph G_V satisfies the *q-completion property* iff for every $(x,y) \in V$ and $a, b \in \{-1, 1\}$ and $c \in \{0, 1\}$ such that $(x + i \times a, y + j \times b) \in V$ with $i = 1$ and $1 \leq j \leq q$ or $j = 1$ and $1 \leq i \leq q$ as well as $(x + c \times a, y + (1-c)b) \in V$ also $(x, y+j) \in V$ for all $1 \leq j \leq q$ or $(x+i, y) \in V$ for all $1 \leq i \leq q$.

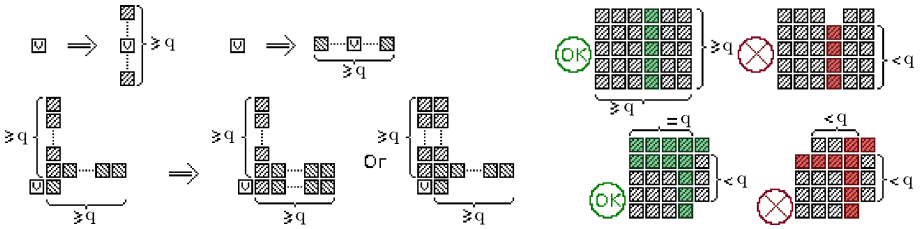


Figure 7 q-width (top) and q-completion (bottom) constraints, and examples.

The q-width and q-completion properties are depicted in Figure 7 at left ($a = b = c = 1$).

▶ **Lemma 4.** *For a connected grid graph G_V, the following two conditions are equivalent:*

1. *G_V is q-square covered and q-connected.*
2. *G_V is q-wide and satisfies the q-completion property.*

Holes in q-square connected grid graphs require a subtle definition of connectedness : two vertices $u, v \notin V$ are *close* iff their distance in $\mathbb{Z}^2$ is at most 2. For instance, two vertices on diagonal positions are close. A subset $H \subseteq \mathbb{Z}^2 \backslash V$ is *close-closed* iff for a vertex $v \in H$ and a vertex $v' \in \mathbb{Z}^2 \backslash V$ which is close to v, also $v' \in H$. The unique maximal close-closed subset $H \subseteq \mathbb{Z}^2 \backslash V$ that is infinite is called *the outside of* G_V, a finite H is called a *hole* of G_V. Note, that holes or the outside are not connected in the neighboring sense as is the grid graph. A grid graph G_V for which there exist no non-empty holes is *solid*.

A *segment* is a straight line of vertices: for $(x, y) \in \mathbb{Z}^2$, $a \in \{0, 1\}$ and $k > 0$ the *segment* $S((x, y), a, k)$ denotes the set of vertices $S((x, y), a, k) = \{(x + ai, y + (1 - a)i) \mid 0 \leq i < k\}$.

4.2 An algorithm for the unrooted problem

We now describe an algorithm for the unrooted balanced connected partitioning problem on q-square connected grid graphs. The algorithm recursively colors a connected zone of $\frac{m}{q}$ vertices leaving $\frac{m(q-1)}{q}$ vertices forming a $(q-1)$-square connected subgraph. Then we pass to the next color and repeat the process leaving a $(q-2)$-square uncolored remainder and so forth. This process can intuitively be compared to coloring the edges of a graph with q-wide corridors leaving $(q-1)$-wide corridors and thus preserving sufficiently large access for $(q-1)$ colors.

For a given q, the coloring phase of the algorithm preserves a complex invariant, *q-compliance*: a *q-compliant coloring* of G_V is a subset $C \subset V$ such that G_C is connected and $G_{V \backslash C}$ is $(q-1)$-covered and $(q-1)$-connected and that there exists a vertex v outside G_V with a neighbor in C.

The algorithm is easier to understand on solid grid graphs, as holes introduce an additional problem. We therefore invite the reader to first try to understand the algorithm in ignoring holes and then read the text again with regard to the way we treat holes: we *cut* them.

A hole H of a grid graph G_V is *cut by coloring* C if there exists a vertex $v \in H$ with a neighbor in C. The vertices of a cut hole are considered outside of G_V with respect to C.

The coloring phase starts by coloring an external corner (a vertex $v \in V$ with two neighbors outside G_V), and then augments q-compliant colorings by three operations:

1. Add a vertex v to C, such that $v \in V \backslash C$ and v has a neighbor in C.
2. Add a segment $S = S((x, y), a, k) \subseteq V \backslash C$ such that $k < q$, (x, y) has a neighbor in C and
   a. either all vertices in $(x + ai, y + (1 - a)i)$ have a neighbor in C or outside V_G and in particular S is terminated by $(x + ak, y + (1 - a)k) \notin V \backslash C$,
   b. or $k = q - 1$, and $(x + ak, y + (1 - a)k) \in H$ for some uncut hole H.

Algorithm 1 Simplified partitioning algorithm.

```
input int q, vertexSet V, such that G_V is q-square connected
int part= |V|/q;
// compute the set of all vertices in uncut holes
vertexSet holeVertices = computeUncutHoles(V)
while(q>0){
   vertexSet C={v} where v is an external corner of V
   while(|C| < part){
      // CLAIM 1, the following extension is always possible :
      C=C∪S for some simple extension S with C∪S q-compliant
      // remove vertices of holes that are cut
      holeVertices = updateUncutHoles(holeVertices,S)
   }
   output part C
   V=V\C , q=q-1
}
```

We call an extension S satisfying one of these three conditions a *simple extension*. Our algoritm below incrementally computes such colorings for determinig a part. An illustrating execution trace is given in the appendix.

▶ **Lemma 5** (Extension lemma). *In Algorithm 1, CLAIM 1 is always true, i.e. for a q-compliant coloring C of a graph G_V such that $|C| < \frac{|V|}{q}$ there exists simple extension S such that $C \cup S$ is q-compliant.*

▶ **Theorem 6.** *For a q-square connected grid graph G_V, Algorithm 1 terminates and computes an (up to $q-2$ vertices per part) balanced connected q-partition in time*[1] *$O(|R||V|q^2)$, where R is the smallest rectangle containing V, or in time $O(|V|^3q^2)$.*

Proof. The Extension lemma (proof: appendix) states that the line of CLAIM 1 is always possible. The invariant ensures that each iteration of the outer loop works on a q-compliant coloring. The termination condition of the inner loop guarantees that after the loop $\frac{|V|}{q} \leq |C| < \frac{m}{q} + q - 1$, the maximal deviation of a part. The algorithm is simplified and can, under bad circumstances, produce a final partition that lacks $(q-1)^2$ vertices, but it is easy to improve this to $q-1$ by balancing the limit between iterations of the outer loop. Moreover, a simple improvement of the algorithm could distributed the last segment attributed to a part in the inner loop between the current part and the next part, limiting the deviation from $\frac{|V|}{q}$ to at most 1 and in fact to 0, perfect balance, in the case of $|V|$ a multiple of q.

For the complexity considerations, we suppose a matrix representation of the input V as a subset of a rectangle R. This rectangle can of course be computed in $O(|R|)$ and $|R| \leq |V|^2$ since V is connected. For practical purposes, the problem can be stated such that $|R|$ itself is a measure of the input size, but for the question of Algorithm 1 being polynomial or not, this detail is of no consequence.

The functions `computeUncutHoles` and `updateUncutHoles` compute reachable sets, e.g. by depth first search, in linear time. Moreover, since `updateUncutHoles` removes vertices from the set of uncut holes and removes each vertex at most once, the total computation time for `updateUncutHoles` is linear in $|R|$.

[1] We neglect non-constant access time to the matrix R, which is of no practical consequence.

Since each iteration of the inner loop increments C and each iteration of the outer loop reduces V by C, the total number of iterations of the inner loop is limited by $|V|$. In each iteration of the inner loop, in principle, all vertices of $V \backslash C$ have to be examined for a possible simple extension. The examination of whether the extension violates the q-compliance however is local around the extension point and can be done[2] in $O(q^2)$. Note also that there is no need to test econnectedness of the uncolored part after an extension: indeed, each path leading through the extension can be replaced by a path avoiding it: width constraints imply this for extensions of type (1) and (2a), whereas an uncut hole is surrounded by a cyclic path that also allows to circumvent the cutting segment. ◀

5 Perspectives and future work

Algorithm optimization. Above, we stated the complexity of Algorithm 1 as $O(|R||V|q^2)$ or $O(|R|^2q^2)$. The complexity is quadratic in $|R|$ because of blindly sweeping R at each iteration in search of the next extension. Several options for improvement exist, for instance, tracking in parallel C and a doubly linked list of actually possible simple extensions in connection with R, we can always pick the first extension of the list and update the list in each iteration in essentially $O(q^3)$ since the impact to other extensions is local. Cutting holes may temporarily require slower updates, but their global impact remains linear in $|R|$. On the other hand, prioritizing simple extensions may allow to bound the set of extensions at any point of the algorithm execution, potentially bringing the complexity to $O(|R|q^2)$.

Towards an algorithm for the rooted problem. We conjecture that Algorithm 1 can be modified to solve the rooted variant of the BCP as follows: In the rooted variant, we start with a grid graph G an a set of q vertices that are already colored with distinct colors: $\forall 1 < i \leq n, C_i = \{v_i\}$, where the C_i designate parts we are building. Here, we cannot greadily extend colors one after the other and keep the q-connection of the remaining uncolored subgraph $G \backslash \bigcup C_i$. Indeed, while a part C_i is not saturated, this part must keep at least a single access to the uncolored part of the graph, i.e $\exists (u, v), u \in C_i, v \in G \backslash \bigcup C_i$. This greatly restricts the possibilities for extensions if we consider a single color at any step.

Instead of extending a single color until the corresponding part is saturated, we can search possible extensions of any color that keeps unsaturated part connected to the uncolored graph, i.e, we search for a set S of maximum $q-1$ vertices $v_1 ... v_q$ that we will add to a colored part C_i, such that $G \backslash \cup C_i$ is $q-1$-connected and $\forall 0 < i < q, \exists (u, v), u \in C_i, v \in G \backslash \cup C_i$. When a colored part C_i becomes saturated, we can remove it and recursively consider the smaller problem on the subgraph $G \backslash C_i$, with the remaining $q-1$ unsaturated parts still connected to the uncolored subgraph which is $(q-2)$-connected.

An important difficulty introduced in the rooted problem is the possible emergence of access conflicts that will exclude a completion of the partition. If these conflicts are not avoided, the greedy algorithm can get stuck. As an example of a simple conflict, consider two unsaturated parts C_i and C_j which both access the uncolored part of the graph by same single vertex. Obviously, such a coloring cannot be completed (c.f. Figure 8 at left for green and red). More general cases can involve up to q parts when all the k parts get their unique

[2] Whereas extensions by a single vertex can produce a violation of q-width or the q-completion property, the extensions by a segment S can only produce a violation of the q-width property.

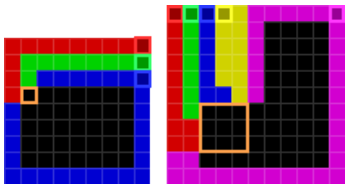


Figure 8 Binary (red and green, at left), and 4-nary (red, green, blue, yellow, at right) conflicts.

access to the uncolored subgraph into the same square of size $(k-1) \times (k-1)$: In Figure 8 (right), extending either red, green, blue, or yellow will recursively create a $(q-1)$-ary conflict between the remaining colors.

We conjecture that the Extension lemma can be modified to take into account such q-ary conflicts and avoid them, at the price of a limited imbalance when approching the saturation of all colors. In the future, we will try to extend Algorithm 1 with this reasoning.

References

1 Ronald Becker, Isabella Lari, Mario Lucertini, and Bruno Simeone. Max-min partitioning of grid graphs into connected components. *Networks: An International Journal*, 32(2):115–125, 1998.

2 Frédéric Chataigner, Liliane RB Salgado, and Yoshiko Wakabayashi. Approximation and inapproximability results on balanced connected partitions of graphs. *Discrete Mathematics and Theoretical Computer Science*, 9(1):177–192, 2007.

3 Janka Chlebíková. Approximating the maximally balanced connected partition problem in graphs. *Information Processing Letters*, 60(5):225–230, 1996.

4 Martin E Dyer and Alan M Frieze. On the complexity of partitioning graphs into connected subgraphs. *Discrete Applied Mathematics*, 10(2):139–153, 1985.

5 Andreas E Feldmann. *Balanced partitioning of grids and related graphs*. ETH Zurich, 2012.

6 Andreas Emil Feldmann. Fast balanced partitioning is hard even on grids and trees. *Theoretical Computer Science*, 485:61–68, 2013.

7 Kamrul Islam, Henk Meijer, Yurai Núñez Rodríguez, David Rappaport, and Henry Xiao. Hamilton circuits in hexagonal grid graphs. In *CCCG*, pages 85–88, 2007.

8 László Lovasz. A homology theory for spanning tress of a graph. *Acta Mathematica Hungarica*, 30(3-4):241–251, 1977.

9 Jun Ma and Shaohan Ma. Ano (k 2 n 2) algorithm to find ak-partition in ak-connected graph. *Journal of Computer Science and Technology*, 9(1):86–91, 1994.

10 Srinivasan Murali and Giovanni De Micheli. Bandwidth-constrained mapping of cores onto noc architectures. In *Proceedings of the conference on Design, automation and test in Europe-Volume 2*, page 20896. IEEE Computer Society, 2004.

11 Bang Ye Wu. A 7/6-approximation algorithm for the max-min connected bipartition problem on grid graphs. In *Computational Geometry, Graphs and Applications*, pages 188–194. Springer, 2011.

A Appendix

A.1 Idea behind the solution given in Fact 2

In Fact 2, we gave a set of dimensions for macrocells to build a square grid instance that satisfies Relations (1) to (3) for given values of tolerance ratio r and number of macrocells m. Let us present the reasoning behind these values.

As the cumulated surface of three inner tanks will be chosen close to the saturation threshold s, we first choose the size of the green tank $t_g = \frac{5}{6}s$, so green will be unable to take inner tanks (Relation (3) right).

Then we choose the size of central tanks so that green exactly enters the saturation interval by covering the green tank and all the central tanks: $t_g + mt_c = s(1-r)$ (Relation (3) left). Now we choose total path length in such a way that it is neglectible in front of tanks: $mp = \frac{t_c}{2}$, so collecting all the path vertices in the topology accounts for less that collecting a single central tank (mp in Relations (2) and (3)). The inner tanks get all the remaining vertices: $mt_i = ms - t_g - m(t_c + p)$ (Relation (2)).

Finally, the last dimension to set is the saturation threshold s. As all previously set dimensions must be integers and are multiples of s, we choose s to be a multiple of all denominators. Finally we also take s sufficiently large so that there exists $k = \frac{p}{c_p}$ such as $p \geq k \times c_p$, *i.e.* there is enough path vertices to wire all the tanks (Relation (1)).

A.2 Proofs of Lemma 3 and Lemma 4

In order to prove these properties, we first prove auxiliary observations:

▶ **Lemma 7.** *Let a connected grid graph G_V be covered by q-squares and q-connected. Then the induced graph $Sq(V, q)$ is also connected.*

Proof. We show that for any two squares $sq(x_1, y_1, q)$ and $sq(x_2, y_2, q)$ in G_V there exists a path in $Sq(V, q)$. The proof uses induction on the least distance d of pairs of vertices $(x'_1, y'_1) \in sq(x_1, y_1, q)$ and $(x'_2, y'_2) \in sq(x_2, y_2, q)$: if this distance is 0 then the squares are intersecting and the q-connected property implies that there is a path between the two squares. Otherwise, there is a first vertex (x'_3, y'_3) on the shortest path from (x'_1, y'_1) to (x'_2, y'_2) that is not in $sq(x_1, y_1, q)$. Since G_V is q-square covered, there exists $sq(x_3, y_3, q)$ in V_G with $(x'_3, y'_3) \in sq(x_3, y_3, q)$. Moreover, because of q-connectedness, there exists a path from $sq(x_1, y_1, q)$ to (x_3, y_3, q) in $Sq(V, q)$. On the other hand, the distance of (x'_3, y'_3) and (x'_2, y'_2) is less than d, hence by induction, there exists a path from $sq(x_3, y_3, q)$ to $sq(x_2, y_2, q)$ in $Sq(V, q)$ and the two paths joint connect $sq(x_1, y_1, q)$ and $sq(x_2, y_2, q)$ as desired. ◀

▶ **Lemma 8.** *Let a connected grid graph G_V be covered by q-squares and q-connected with $q > 1$. Then each $(q-1)$-square in G_V is contained in a q-square of G_V.*

Proof. Sketch: let $sq(x, y, q-1)$ be a $(q-1)$-square. In order to show the existence of a q-square in G_V containing it, we first show that there exists a q-square containing one or several lines from one side of the $(q-1)$-square and from there by induction we show that there exists a q-square containing all the $(q-1)$ lines from that side. The proof necessarily makes use of the q-connected property. ◀

Proof of Lemma 3. Suppose that two $(q-1)$-squares are touching or intersecting. Any two q-squares containing them also touch or intersect. We consider two q-squares containing them of minimal distance and consider a path between the two. A path consists of steps

"up", "left", "right" or "down" and it never contains both "up" and "down" on the one hand or "left" and "right" on the other. A $(q-1)$-square can imitate the same directions and always be included in the q-square of a path, however, it may end up on the "wrong" among the 4 $(q-1)$-squares in the final q-square and require correction. However, it is easy to see that this correction can either be achieved by skipping a step of the q-square path (thus relatively moving in the opposite direction) or by adding a step in the final q-square and we thus end up with a path of the desired distance. ◀

Proof of Lemma 4. (1) implies (2): q-width trivially follows from q-square covering. Suppose a constellation (x, y) and a, b, c concerning the q-completion property. Without loss of generality we suppose $a = b = c = 1$, other cases are very similar. Supposing the property fails to hold for this instance of parameters, then there exist vertices $(x+i, y), (x, y+j) \notin V$ with $1 \leq i, j \leq q$. We suppose i, j minimal, i.e. for all $1 \leq i' < i$ we have $(x+i', y) \in V$ and $(x, y+j') \in V$. We show the existence of q-squares in G_V that touch or overlap, yet violate the q-connected property because of the obstacles represented by $(x+i, y), (x, y+j)$. First assume $j = 1$, then the only q-square containing $(x+1, y+1)$ can be $sq(x+1, y+1, q)$. This q-square touches the q-square containing $(x+1, y)$ yet a direct path to the letter q-square starting at $sq(x+1, y+1, q)$ would have to start either down or left, but there are no edges in that direction, a contradiction. So $j > 1$. Now consider the q-squares containing vertices $(x+i', y+1)$ with $0 \leq i' <\leq i$, all of which belong to G_V. Some of them (e.g. $(x, y+1)$) must be below $y + j' - q + 1$, others above $y + 1$ and hence there must exist an i' such that $(x+i', y+1)$ is in a q-square below $y + j' - q + 1$ and $(x+i'+1, y+1)$ in a q-square above $y+1$ and these q-squares touch or intersect, yet there is no direct path between them because of the two obstacle vertices. Another contradiction. Hence, we cannot at the same time have both $(x+i, y), (x, y+j) \notin V$, hence the constellation satisfies the q-completion property.

(2) implies (1):

First we show that the grid graph is q-square covered. We do this by a construction of coloring a part of the vertices of the graph incrementally in such a way, that the colored zone is invariantly a union of q-squares. We show that this process can be continued until all vertices are colored. The construction starts by choosing a vertex (x, y) with x minimal. Since the graph is q-wide this gives us q vertices $(x, y_1)...(x, y_q)$ with $y = y_i$, all with x minimal. Again because of q-width, it follows that $sq(x, y_1, q) \subseteq V$ and this is the first square we color. Now suppose that there are uncolored vertices left. Since the graph is connected, at least one such vertex is adjacent to a colored square. Without loss of generality, let us suppose $sq(x, y+1, q)$ fully colored and $(x+i, y)$ with $0 \leq i < q$ not yet colored. Because of q-width, there exists x' such that $x' + k = x + i$ with $0 \leq k < q$ and for all $0 \leq l < q$ we have $(x'+l, y) \in V$. We suppose furthermore x, y, i, x' chosen with these conditions such that $|x - x'|$ is minimal. Then either $x = x'$, but then $sq(x, y, q) \subseteq$ and we can color $(x, y)...(x+q-1, y)$ and continue. Or $x! = x'$ and let us suppose that $x' < x$. Then the preconditions for q-completion are satisfied for $(x-1, y)$ and $a = b = c = 1$ and either completion contradicts the assumption of having chosen x, y, i, x' such that $|x - x'|$ is minimal.

Now we want to show that G_V is q-connected. Supposing it is not, then there are two q-squares $sq(x, y, q), sq(x', y', q)$ such that $\max\{|x-x'|, |y-y'|\} \leq q$ and $\min\{|x-x'|, |y-y'|\} < q$ but their distance in $Sq(V, q)$ is greater than $|x - x'| + |y - y'|$. Assume $|x - x'| + |y - y'|$ minimal for such a pair of q-squares. Obviously, $\min\{|x - x'|, |y - y'|\} > 0$ otherwise a straight path between the two q-squares obviously exists contradicting assumptions. The non-trivially overlapping or touching squares give rise to a constellation of the q-completion property which in turn yields a third square at distance 1 from one of the two squares and 1

unit closer to the other square. Since we assumed $|x - x'| + |y - y'|$ minimal, this results in an intermediate q-square on a path of length $|x - x'| + |y - y'|$ between the original q-squares, contradicting assumptions. ◀

A.3 Proof of the Extension Lemma

The proof of the Extension lemma is based on a (constructive) recursive analysis of a q-compliant coloring: it is based on the identification of potential extension points, and if the latter induces a conflict to q-compliance, there is a way to subdivide the area of extension points until necessarily we reach an area too small to contain a conflict.

In order to formalize the search area, we introduce the notion of *coloring border* as the edge between the outside and the colored part of V_G on the one hand and the uncolored part of V_G on the other hand: Visually, this border is between vertices and we formalize it as couples: let (u, v) be a pair of neighboring vertices such that $v \in V \backslash C$ and $u \in C$ or $u \in H$ for some cut hole of V_G or such that u is outside of V_G *coloring border candidate*. If $u \in C$ we call it a *colored edge*, in the case of $u \in H$ we call it an *uncolored edge*. Coloring border candidates $(u, v), (u', v')$ are *connected* iff u is a neighbor of u' and v is a neighbor of v' or $u = u'$ and $v = (x, y), v' = (x', y')$ satisfy $|x - x'| + |y - y'| = 2$, i.e. they are on diagonal positions, or conversely $v = v'$ and u, u' are on diagonal positions. Graphically, connected coloring border candidates can be drawn without lifting the pen and share a common point. An extension point is a coloring border candidate (u, v) with $u \in C$ and v the first vertex of a simple extension S. A *coloring border* is a connected set of coloring border candidates.

A coloring border B is *promising* if

- it contains colored segments;
- for any simple extension $S(x, y, a, k)$ with (x, y) having a colored neighbor on the border its addition to C either preserves q-compliance or it causes a conflict with an uncut hole or with a colored edge that is either on the coloring border or directly connected to it;
- one of the following cases holds :
  1. It contains uncolored segments $S(x, y, a, q - 1)$ such that all vertices are connected to coloring border candidates and such that $(x - a, y - (1 - a)), (x + a(q - 1), y + (1 - a)(q - 1)) \notin V \backslash C$.
  2. It does not satisfy (1), there exists a vertex $v \in V \backslash C$ having one neighbor $v_1 \in C$ and another neighbor $v_2 \in \mathbb{Z}^2 \backslash V$ (a corner with one side colored, the other outside V), with (v, v_1) in the coloring border.
  3. It does not satisfy (1) or (2), but it contains edges (v, v') with $v' \in \mathbb{Z}^2 \backslash V$.
  4. It does not satisfy (1) or (2) or (3), but it contains two "colored corners", i.e. pairs of edges (v_i, v_i') and (v_i, v_i'')such that $v_i \in V \backslash C$ and $v_i', v_i'' \in C$ (concave corners with color on both sides), and such that no convex corner (with color on both sides) exists between them (such pairs of concave corners naturally contain a segment all along the edges between the two corners).

Proof of Lemma 5 (Extension lemma). By induction, we claim that a promising coloring border contains an extension that is q-compliant. Then, since the complete border of the partially colored graph is trivially promising, the result follows.

Now let B be a promising coloring border. We examine the cases and show that for each case either a q-compliant simple extension can be identified, or that a smaller promising coloring border can be identified. Case (1) allows immediate coloring, as such a segment cannot break $(q - 1)$-completion and the only way it could break the $(q - 1)$-width would be if $|V \backslash C| = (q - 1)^2$, which contradicts the assumptions $|C| < \frac{|V|}{q}$.

Suppose case (2): adding a single vertex in the corner cannot break $(q-1)$-width, because this contradicts that (1) is not satisfied. Without loss of generality, let us assume $v = (x, y)$ such that $(x-1, y) \in C$ and $(x, y-1) \notin V$ (other cases are symmetric). In order to generate a conflict to $(q-1)$-completion, we must have $(x+q-1, y), (x+q-2, y)...(x+q-2, y+q-1) \in V \backslash C$ but that $(x', y') \notin V \backslash C$ for an $(x', y') \in \{(x+q-1, y+1), ..., (x+q-1, y+q-1)\}$. Since V_G is q-square connected, it is not difficult to see that $\{(x+q-1, y+1), ..., (x+q-1, y+q-1)\} \subseteq V$ hence $(x', y') \in C$. With $(x'-1, y') \in V \backslash C$ it forms a coloring border candidate and since the coloring border is promising, it is connected to the corner. Considering the shortest path between the corner and the conflicting edge, then by removing the edge linking the corner to the colored vertex, we obtain a new coloring border. We claim that it is promising : it contains concave corners because of the orientation of the edges at the two vertices, and moreover, if an extension of any of the types (1)-(4) on this sub-border have conflicts, they must occur within the sub-border.

For the case (3), let us suppose (other cases are symmetric) that $v = (x, y), (x+1, y) \in V \backslash C$ and moreover $(x, y-1) \in C$ and $(x+1, y-1) \notin V$. If adding v produces a conflict to q-compliance, it is either a conflict to $(q-1)$-width or to $(q-1)$-completion.

Let us first consider the case of an $(q-1)$-width conflict. Such a conflict can concern $(x, y+q-2)$, $(x-k, y)$ or $(x+k, y)$ for some $1 < k < q-1$. In the first case, since G_V is q-square connected, it is easy to see that $(x, y+q-2) \in V$, and consequently, it is in C. By removing the edges $(x, y), (x, y-1)$ and $((x, y+q-2), (x, y+q-3))$ from the border coloring and by keeping a path between these two edges connecting them and containing colored edges, we obtain a coloring border of reduced size, which we claim to be promising : for example, the two removed edges are parallel and the connecting path necessarily contains concave corners (otherwise, we end up with a contradiction to the assumption that (1) does not hold), and finally, given that the remaining coloring border is "almost closed", a conflict with respect to some extension point cannot be situated beyond the two removed edges. In the second case, we consider the simple extension $(x-k+1, y)$. It cannot have an $(q-1)$-width conflict (otherwise we would have (1), but we assume that we did not) and because of $(x+1, y-1) \notin V$ we find that a conflicting vertex with respect to $(q-1)$-completion must be $(x-k+q-1, y+q-2) \in C$ we apply a similar reasoning as above in order to identify a sub coloring border. Finally, In the third case (and excluding the second case), the segment $\{(x, y), ..., (x+k-1, y)\}$ is a simple extension that cannot introduce a conflict to $(q-1)$-width and as before, if there is a conflict to $(q-1)$-completion, this conflict is between the edge below (x, y) and some colored edge and again, we apply a reduction of the coloring border. In any case, either the extension preserves q-compliance, or the reduced coloring border is promising and contains by induction an extension preserving q-compliance, that extends obviously to the complete bordering color.

For the case (4), we suppose (up to symmetry) corners $(x, y), (x+k, y)$ with $(x-1, y), (x+k+1, y) \in C$ and $(x, y-1), ..., (x+k, y-1) \in C$. If adding (x, y) to C causes a conflict, it is either due to a colored vertex and we proceed by induction as above. If, on the other hand, there exists a conflicting vertex $v' \notin V$ then this vertex cannot be on the coloring border (assumption of case (4)), hence it must be part of an uncut hole at position $(x+q-1, y+q-1)$. We claim that in this case, the simple extension $S = \{(x+q-1, y), ..., (x+q-1, y+q-2)\}$ preserves q-compliance. First, it cannot introduce a conflict with $(q-1)$-completion, but also, it cannot introduce a conflict with $(q-1)$-width. Also observe that the uncut hole is surrounded by some closed path in $V \backslash C$, cutting it cannot cause $V \backslash (C \cup S)$ to become disconnected. ◀

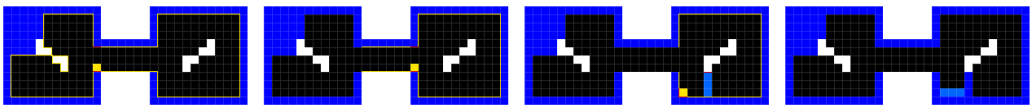
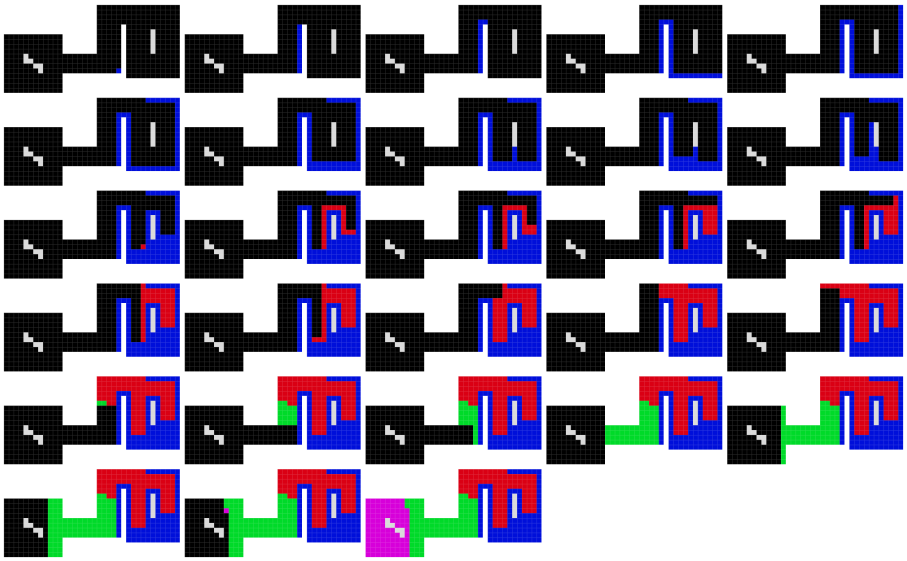


Figure 9 Promising coloring border and extension search.

The recursive reasoning of the proof is illustrated in Figure 9 for the case $q = 4$, from left to right: (a) A partial coloring and a promising coloring border (initially the inner contour) is given and a possible simple extension (Case 3) is shown in yellow, but it induces a conflict concerning $(q-1)$-width, the edges cut are marked red, we keep the path connecting these edges at right. A second attempt of Case 3 is shown in the second picture. In the third picture, there are no more uncolored edges (Case 4) in the coloring border so we try a concave corner, but there is a conflict with an uncut hole, so we cut it instead, which conclude the search for the extension. The last picture shows an application of Case 1.

A.4 Example computation of our coloring algorithm

Below, we show a trace of the partitioning by Algorithm 1 for illustration.



Concurrent Games and Semi-Random Determinacy

Stéphane Le Roux
Darmstadt Technical University, Department of Mathematics, Darmstadt, Germany
leroux@mathematik.tu-darmstadt.de

Abstract

Consider concurrent, infinite duration, two-player win/lose games played on graphs. If the winning condition satisfies some simple requirement, the existence of Player 1 winning (finite-memory) strategies is equivalent to the existence of winning (finite-memory) strategies in finitely many derived one-player games. Several classical winning conditions satisfy this simple requirement.

Under an additional requirement on the winning condition, the non-existence of Player 1 winning strategies from all vertices is equivalent to the existence of Player 2 stochastic strategies almost-sure winning from all vertices. Only few classical winning conditions satisfy this additional requirement, but a fairness variant of omega-regular languages does.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory, Theory of computation → Verification by model checking, Software and its engineering → Software verification, Software and its engineering → Model checking

Keywords and phrases Two-player win/lose, graph, infinite duration, abstract winning condition

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.40

Acknowledgements Referees (of several conferences) and Sasha Rubin made helpful comments. A simplification of the proof of Lemma 11 was triggered by a conversation with Arno Pauly.

1 Introduction

Computer science models systems interacting concurrently with their environment via infinite duration two-player win/lose games played on graphs: a play starts at a *state* of the graph, where the players *concurrently* choose one *action* each and thus induce the next state, and so on for infinitely many rounds. The *winning condition* is a given subset W of the infinite sequences of states, and Player 1 wins the play iff the sequence of visited states belongs to W. A *strategy* of a player prescribes one action depending on what has been played so far, and a *winning strategy* is a strategy ensuring victory regardless of the opponent strategy.

There are games where neither of the players has a winning strategy, but Borel determinacy [25] guarantees the existence of a winning strategy in games where the players play alternately and the winning condition is a Borel set. Under Borel condition again, Blackwell determinacy [26] guarantees a weaker conclusion when the players play concurrently: there exists a value $v \in [0, 1]$ such that for all $\epsilon > 0$ the players have stochastic strategies guaranteeing victory with probability $v - \epsilon$ and $1 - v - \epsilon$, respectively.

In the special case of concurrent games played on finite graphs with ω-regular winning conditions, [11] designed algorithms to decide the existence of (stochastic) strategies that are winning, winning with probability one, and winning with probability $1 - \epsilon$ for all $\epsilon > 0$. [11] also mentions a three-state game where only the latter exist, which exemplifies the complexity of the concurrent ω-regular games on finite graphs. Then [6] studied concurrent

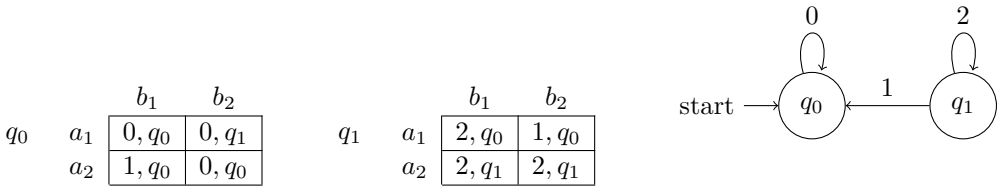


Figure 1 To the left, a concurrent game with states q_0, q_1, colors $0, 1, 2$, and two actions per player. To the right, a one-player game derived by using the delayed response $[(0, q_0)(0, q_0)]; [(1, q_0)(2, q_1)]$.

prefix independent winning conditions, which is strictly more general than the ω-regular conditions, and [13] further improved upon some results. Some of these results were extended recently to multi-player multi-outcome games, see e.g. [3], [15].

The new games. This article studies slightly different games: when the players concurrently choose one action each, it also produces a *color*; the winning condition is now a given subset W of the infinite sequences of colors; and Player 1 wins the play iff the produced sequence of colors belongs to W. There are two differences between the classical games and the new games. First, the winning condition does not involve the visited states but the transitions instead; second it does so indirectly, via colors labeling the transitions. E.g. in the game on the left-hand side of Figure 1, starting at q_0, the action sequence $(a_1, b_1)(a_1, b_2)(a_1, b_1)$ yields the state sequence $q_0q_0q_1q_0$ and the color sequence 002.

There are several reasons why these new games are interesting.

- The classical games can be encoded easily into the new ones by using state names as colors. Variants such as the games with colored states, or the colorless games with winning condition on the transitions can also be encoded easily into the new games.
- The converse encoding may increase the state space (to infinity for games with infinitely many actions). Note that the transition-versus-state issue was already studied in the turned-based setting in [10]. Likewise, colorless games are encoded easily in games with colors without size increase, and colors usually lead to more succinct winning conditions.
- Colors are widely used in turn-based games, and for all games they help to study the winning conditions independently from the game structure, and thus to approximate or even characterize nice winning conditions for classes of games (usually simple to check) rather than for single games (usually more accurate but harder to check). This is exemplified by the difference between Theorems 5 and 7 in [27].
- Whereas classical one-state games are trivial, the new one-state games are fairly complex and constitute a nice intermediate object towards the understanding of the more complex general games. Likewise, some one-state (aka stateless) objects from the literature are interesting in their own right: [1] studied one-state multi-objective Markov decision processes; vector addition systems (VAS, [17]) are still studied despite the vector addition systems with states (VASS, [16]); the Minkowski games [24] defined with finite sets are a special case of the one-state games from this article.

The main results.

- If W is closed under *interleaving* and *prefix removal*, and if states and colors are finitely many, the existence of a Player 1 winning (finite-memory) strategy is equivalent to the existence of winning (finite-memory) strategies in finitely many derived one-player games.

- If, in addition, W is *factor-prefix complete* and there are finitely many actions, either Player 1 has a winning strategy from one state, or every Player 2 constant (stronger than positional!), positive, stochastic strategy is almost-sure winning from all vertices. This is *semi-random determinacy*.
- One-state games enjoy a stronger conclusion than in the previous item under somewhat weaker assumptions: if the winning condition is *factor-set complete* and closed under interleaving, if Player 2 has finitely many actions, either Player 1 has a winning strategy, or every Player 2 constant, positive stochastic strategy is almost-sure winning.

The finitary flavor of the above characterizations yields decidability and memory sufficiency, in the rough range of double exponentials in the number of states times the number of colors.

In the context of semi-random determinacy, a neutral, random Player 2 is therefore as bad for Player 1 as a hostile environment. Also, the victory is clear-cut in the above results: no need for approximate optimal strategies, no need for the notion of value, etc. This is due to the assumptions, and it is legitimate to wonder how restrictive they are.

Several classical winning conditions from computer science are closed under interleaving, see Section 5. The Muller condition is not, but the parity condition is, so the first characterization result extends to the concurrent Muller games via the Last Appearance Record (LAR), as done in [28]. So, closedness under interleaving is not as restrictive as it may seem.

Fewer classical winning conditions are factor-prefix complete (defined in Section 3.2), but the boundedness condition from [24] and a variant of the ω-regular languages are both closed under interleaving and factor-prefix complete. The variant is as follows: each produced color requests some combinations of colors to occur in the future. In winning plays, the number of currently unsatisfied requests should be uniformly bounded over time. It may be relevant even as a business model: at every time unit the system can pay penalties for every currently unsatisfied request, which may be covered by greater, albeit bounded, instantaneous income.

The above variant relates to the notion of *fairness*, which requires that co-finitely many requests are eventually satisfied. The *finitary fairness* [2] additionally requires uniformly bounded response time. This idea was used in [12] to study temporal logic, and in [9] to study finitary parity games. Requiring uniformly bounded response time (or variants thereof) to study games has been further used later, e.g. in [5]. However, these notions of fairness do not enjoy closedness under interleaving and factor-prefix completeness. (Details in Section 5.)

Related works. The semi-random determinacy implies the *bounded limit-one property* from [11] for the new games: if one state has positive value, one state has value one.

Corollary 4 generalizes the nice Theorem 4 from [18]. Note that the convexity of winning conditions defined in [18] is a essentially the same as the interleaving closedness defined here.

This article also shares similarities with [14]: both use abstract winning conditions, and both characterize the existence of winning strategies in two-player games by the existence of winning strategies in finitely many derived one-player games. Several articles adopted a similar approach: [19] and [20] reduce multi-player multi-outcome Borel games to simpler two-player win/lose Borel games, and characterize the preferences and structures that guarantee the existence of Nash equilibrium in infinite tree-games; [21] does the same to characterize the preferences that guarantee the existence of subgame perfect equilibrium (at low levels of the Borel hierarchy); [23] and [27] do the same to almost characterize the existence of finite-memory Nash equilibrium in games on finite graphs; [22] reduces one-shot concurrent two-player multi-outcome games to simpler one-shot concurrent two-player win/lose games, with applications to generalized Muller games and generalized "parity" games.

One of the benefits of abstraction is that it leads to more general results: e.g. [23] noted that the lexicographic product of mean-payoff and reachability objectives cannot be encoded into real-valued payoffs, and [27] proved it.

Structure of the article. Section 2 gives basic definitions. Section 3 presents the main results and additional definitions. Section 4 discusses the key elements of the proofs. Section 5 presents applications.

2 Definitions

The folklore Observation 1 below will be used extensively to lift properties from finite words to infinite words. It will be first explicitly invoked, and then only implicitly used.

▶ **Observation 1.** *Let $f : S^* \to T^*$ be such that $u \sqsubseteq v \Rightarrow f(u) \sqsubseteq f(v)$, where $\sqsubseteq$ is the prefix relation. Then f can be uniquely extended to $S^* \cup S^\omega \to T^* \cup T^\omega$ such that $f(\rho_{\leq n}) \sqsubseteq f(\rho)$ for all $n \in \mathbb{N}$ and $\rho \in S^\omega$.*

Games. A *game (with colors and states)* is a tuple $\langle A_1, A_2, Q, q_0, \delta, C, \mathrm{col}, W\rangle$ such that
- A_1 and A_2 are non-empty sets (of actions for Player 1 and Player 2),
- Q is a non-empty set (of states),
- $q_0 \in Q$ (is the initial state),
- $\delta : Q \times A_1 \times A_2 \to Q$ (is the state update function).
- C is a non-empty set (of colors),
- $\mathrm{col} : Q \times A_1 \times A_2 \to C$ (is a color trace),
- $W \subseteq C^\omega$ (is the winning condition for Player 1)

Histories. The *full histories* (*full runs*) of such a game are the finite (infinite) words over $A_1 \times A_2$, the Player 2 *histories* (Player 2 *runs*) are the finite (infinite) words over A_2, and the Player 1 *histories* (Player 1 *runs*) are the finite (infinite) words over A_1.

Strategies. A Player 1 strategy is a function from A_2^* to A_1. Informally, it requires Player 1 to remember exactly how Player 2 has played so far, and it tells Player 1 how to play.

Induced histories. The function h is defined inductively below. As arguments it expects a strategy and a Player 2 history in A_2^*, and it returns a full history: the very full history that, morally, should happen if Player 1 followed the given strategy while Player 2 played the given Player 2 history. Namely, $h(s, \epsilon) := \epsilon$ and $h(s, \beta \cdot b) := h(s, \beta) \cdot (s(\beta), b)$.

By Observation 1 the function h is extended to expect opponents runs in A_2^ω and return full runs: $h(s, \boldsymbol{\beta})$ is the only action run whose prefixes are the $h(s, \boldsymbol{\beta}_{\leq n})$ for $n \in \mathbb{N}$.

Extending the update and trace functions. The state update function δ is extended to $\Delta : (A_1 \times A_2)^* \to Q$ inductively: $\Delta(\epsilon) := q_0$ and $\Delta(\rho \cdot (a, b)) := \delta(\Delta(\rho), a, b)$. Using Δ, the trace function col is naturally lifted to full histories by induction: $\mathrm{col}(\epsilon) := \epsilon$ and $\mathrm{col}(\rho \cdot (a, b)) := \mathrm{col}(\rho) \cdot \mathrm{col}(\Delta(\rho), a, b)$. The trace function is further extended to full runs by Observation 1. When considering several games, indices may be added to the corresponding Δ and col.

Winning strategies. A Player 1 strategy s is winning if $\mathrm{col} \circ h(s, \boldsymbol{\beta}) \in W$ for all $\boldsymbol{\beta} \in A_2^\omega$. If there is a Player 1 winning strategy in a game, one says that Player 1 wins the game.

Memory. A Player 1 strategy s is said be implementable with memory M, or memory size $\log_2 |M|$, if there exist a set M and $m_0 \in M$, and two functions $\sigma : Q \times M \to A_1$ and $\mu : Q \times M \times A_2 \to M$ such that $s(\beta) = \sigma(\Delta \circ h(s, \beta), m(\beta))$, where m is defined inductively by $m(\epsilon) := m_0$ and $m(\beta b) := \mu(\Delta \circ h(s, \beta), m(\beta), b)$. If M is finite, s is called a finite-memory strategy. Note that every Player 1 strategy is implementable with memory A_2^ω.

One-player games. Intuitively, a *one-player game (with colors and states)* amounts to a game where Player 2 has only one strategy available, i.e. $|A_2| = 1$. Formally, it is a tuple $\langle A_1, Q, q_0, \delta, C, \mathrm{col}, W\rangle$ such that A_1, Q, and C are non-empty sets, $q_0 \in Q$, $\delta : Q \times A_1 \to Q$, $\mathrm{col} : Q \times A_1 \to C$, and $W \subseteq C^\omega$. In this context, the *full histories* (*full runs*) of such a game are the finite (infinite) words over A_1, and the Player 2 *histories* of Player 1 are the natural numbers (telling how many rounds have been played). There is only one Player 2 *run*, namely ω. Then, a Player 1 strategy is a function from $\mathbb{N}$ to A_1, and the notation for the induced full histories is overloaded: $h(s, 0) := \epsilon$ and $h(s, n + 1) := h(s, n) \cdot s(n)$. By Observation 1 the function h is (again) extended: $h(s, \omega)$ is the only action run whose prefixes are the $h(s, n)$ for $n \in \mathbb{N}$. A Player 1 strategy s is winning if $\mathrm{col} \circ h(s, \omega) \in W$.

Prefix removal. A set of infinite sequences is closed under prefix removal if the tails of the sequences from the set are again in the set. Formally, $W \subseteq C^\omega$ is closed under prefix removal if the following holds: $\forall(\gamma, \boldsymbol{\gamma}) \in C^* \times C^\omega$, $\gamma \cdot \boldsymbol{\gamma} \in W \Rightarrow \boldsymbol{\gamma} \in W$. Note that closedness under prefix removal is weaker than the prefix independence assumed in [6], [13], and [18].

Interleaving. Interleaving two infinite sequences consists in enumerating sequentially (prefixes of) the two sequences to produce a new infinite sequence. For example, interleaving $(2n)_{n\in\mathbb{N}}$ and $(2n + 1)_{n\in\mathbb{N}}$ can produce the sequences $(n)_{n\in\mathbb{N}}$ (perfect alternation), $1 \cdot 0 \cdot 3 \cdot 5 \cdot 2 \cdot 7 \cdot 4 \cdot 6 \cdot (n + 8)_{n\in\mathbb{N}}$, and $(2n)_{n\in\mathbb{N}}$ (by enumerating the first sequence only), but not the sequences $(4n)_{n\in\mathbb{N}}$ or $0 \cdot 1 \cdot 4 \cdot 3 \ldots$.

Delayed response. Consider a game $g = \langle A_1, A_2, Q, q_0, \delta, C, \mathrm{col}, W\rangle$ with finite Q and C. For every $q \in Q$ let $E_1^q, \ldots, E_{k_q}^q$ be the elements of $\{(\mathrm{col}, \delta)(q, a, A_2) \mid a \in A_1\}$, where $(\mathrm{col}, \delta)(q, a, A_2) := \{(\mathrm{col}(q, a, b), \delta(q, a, b)) \mid b \in A_2\}$ for all $a \in A_1$. The elements of $\otimes_{q\in Q, i\leq k_q} E_i^q$ are called the Player 2 *delayed responses*. Intuitively, a Player 2 delayed response amounts to a Player 2 positional strategy in (and only in) a sequentialized version of the game. In every round of this version, Player 1 chooses an action first, then Player 2 chooses an action (or more precisely some color and state among the pairs he could induce by choosing an action). E.g. $[(0, q_0)(0, q_0)]; [(1, q_0)(2, q_1)]$ is a delayed response for Figure 1. It means that at state q_0, Player 2 selects $(0, q_0)$ for both actions of Player 1, and at state q_1 it selects $(1, q_0)$ if Player 1 chooses action a_1. Note that delayed responses are not Player 2 (positional) strategies in the concurrent game, e.g. as $[(0, q_0)(0, q_0)]$ is not achievable in any column.

Derived one-player games. Let t be a Player 2 delayed response. The one-player game $g(t) := \langle A_1, Q, q_0, \delta_t, C, \mathrm{col}_t, W\rangle$ is defined by $(\mathrm{col}_t, \delta_t)(q, a) := t_{q,(\mathrm{col},\delta)(q,a,A_2)}$, the projection of t on the (q, E_i^q)-component such that $E_i^q = (\mathrm{col}, \delta)(q, a, A_2)$. Intuitively, $g(t)$ is the game obtained by letting Player 2 fix his strategy (to realize) t in the sequentialized version of g. For example, the game on the left-hand side of Figure 1 applied to the delayed response $[(0, q_0)(0, q_0)]; [(1, q_0)(2, q_1)]$ yields the game on the right-hand side of Figure 1.

3 Main results

Section 3.1 characterizes the existence of Player 1 winning strategies and gives a complexity result. Section 3.2 defines additional concepts and uses the above characterization to characterize the existence of Player 2 everywhere-winning stochastic strategies. Section 3.3 studies the special case of one-state games and presents the semi-random determinacy.

3.1 Existence of Player 1 winning strategies

Theorem 2 below characterizes the existence of Player 1 winning strategies in a game via the existence of winning strategies in finitely many derived one-player games. Theorem 3 afterwards drops the assumption on closedness under prefix removal from Theorem 2, but at the cost of a universal quantification over the starting state of the game. In Theorems 2 and 3, the finiteness and the closedness assumptions are used only to prove the $2 \Rightarrow 1$ implications.

▶ **Theorem 2.** *Consider a game $g = \langle A_1, A_2, Q, q_0, \delta, C, \mathrm{col}, W\rangle$. If Q and C are finite, and W is closed under interleaving and prefix removal, the following are equivalent.*

1. Player 1 *wins g.*
2. Player 1 *wins $g(t)$ for all delayed responses t.*

If A_1 is finite and Player 1 *wins, she can do it with memory size $O(f(|A_1|,|Q|,|C|)\cdot(|C\times Q|)^{|Q|2^{|C\times Q|}})$, where $f(|A_1|,|Q|,|C|)$ is a sufficient memory size to win the one-player games using A_1, Q and C.*

▶ **Theorem 3.** *Consider games $g_q = \langle A_1, A_2, Q, q, \delta, C, \mathrm{col}, W\rangle$ parametrized by $q \in Q$. If A_1, A_2, and Q are finite, if W is factor-prefix complete and closed under interleaving and prefix removal, the following are equivalent.*

1. Player 1 *wins g_q for all $q \in Q$.*
2. Player 1 *wins $g_q(t)$ for all $q \in Q$ and delayed responses t.*

If the above holds, Player 1 *wins every g_q with memory size as in Theorem 2.*

In games that are (or encode) turn-based games, the delayed responses are Player 2 positional strategies. So, restricting Theorems 2 and 3 to turn-based games yields Corollaries 4 and 5, respectively. Note that Corollary 4 generalizes Theorem 4 from [18] by only assuming closedness under prefix removal instead of prefix independence. This is significant since the safety condition is closed under interleaving and prefix removal, but is not prefix independent.

▶ **Corollary 4.** *Consider a game $g = \langle A_1, A_2, Q, q_0, \delta, C, \mathrm{col}, W\rangle$ encoding a turn-based game. If Q and C are finite, and W is closed under interleaving and prefix removal, either* Player 1 *has a winning strategy or* Player 2 *has a positional winning strategy.*

▶ **Corollary 5.** *Consider games $g_q = \langle A_1, A_2, Q, q, \delta, C, \mathrm{col}, W\rangle$ parametrized by $q \in Q$ and encoding a turn-based games. If Q and C are finite, and W is closed under interleaving, either* Player 1 *wins all g_q, or* Player 2 *has a positional winning strategy for some g_q.*

The characterizations from Theorems 2 and 3 yields decidability results and rough algorithmic complexity estimates in Corollary 6 below. Note that checking all the possible strategies using memory size given by Theorems 2 and 3 would be slower than Corollary 6.

▶ **Corollary 6.** *Let $\mathcal{C} \neq \emptyset$, let $W \subseteq \mathcal{C}^\omega$ be closed under interleaving and prefix removal (resp. by interleaving), and let $f : \mathbb{N}^3 \to \mathbb{N}$ be such that for all finite $C \subseteq \mathcal{C}$ and all one-player games $\langle A_1, Q, q_0, \delta, C, \mathrm{col}, W\rangle$, it takes at most $f(|A_1|,|Q|,|C|)$ computation steps to decide*

the existence of a (finite-memory) winning strategy in the game. Then for all finite games $g_{q_0} = \langle A_1, A_2, Q, q_0, \delta, C, \mathrm{col}, W\rangle$ *it takes at most*

$$f(|A_1|, |Q|, |C|) \cdot (|C \times Q|)^{|Q|2^{|C\times Q|}} + |Q||A_1||A_2|$$

computation steps to decide whether Player 1 *wins* g_{q_0} *(with finite memory).*

(resp. $|Q| \cdot f(|A_1|, |Q|, |C|) \cdot (|C \times Q|)^{|Q|2^{|C\times Q|}} + |Q||A_1||A_2|$ *computation steps to decide whether* Player 1 *wins* g_q *(with finite memory) for all* $q \in Q$*.)*

3.2 Existence of Player 2 almost-sure winning random strategies

Consider a game $\langle A_1, A_2, Q, q_0, \delta, C, \mathrm{col}, W\rangle$.

Probability distribution. A probability distribution on a finite set E is a function $f : E \to [0, 1]$ such that $\sum_{e\in E} f(e) = 1$. Let us call $D(E)$ the set of the probability distributions on E.

Stochastic strategies. A Player 1 (Player 2) stochastic strategy is a function $\sigma : (A_1 \times A_2)^* \to D(A_1)$ ($\tau : (A_1 \times A_2)^* \to D(A_2)$).

Induced stochastic histories. The function H is defined inductively below. As arguments it expects stochastic strategies σ and τ for Player 1 and a Player 2, respectively, and it returns a function from $(A_1 \times A_2)^*$ to $\mathbb{R}$. Namely, $H(\sigma, \tau)(\epsilon) := 1$, and $H(\sigma, \tau)(\rho \cdot (a, b)) := H(\sigma, \tau)(\rho) \cdot \sigma(\rho)(a) \cdot \tau(\rho)(b)$. It is easy to check that $H(\sigma, \tau)(\rho) \geq 0$ for all $\rho \in (A_1 \times A_2)^*$, and that $\sum_{|\rho|=n} H(\sigma, \tau)(\rho) = 1$ for all $n \in \mathbb{N}$.

Induced probability measure. For every pair $(\sigma, \tau) \in D(A_1)^{(A_1\times A_2)^*} \times D(A_2)^{(A_1\times A_2)^*}$ one defines a probability measure $\lambda(\sigma, \tau)$ on $(A_1 \times A_2)^\omega$ by setting $\lambda(\sigma, \tau)(\rho \cdot (A_1 \times A_2)^\omega) := H(\sigma, \tau)(\rho)$ for all $\rho \in (A_1 \times A_2)^*$. (It is then extended uniquely to measurable sets.)

Almost-sure winning stochastic strategies. A Player 2 stochastic strategy τ is said to be almost-sure winning if $\lambda(\sigma, \tau)(\mathrm{col}^{-1}[W]) = 0$ for all $\sigma \in D(A_1)^{(A_1\times A_2)^*}$. (Recall that $\mathrm{col} : (A_1 \times A_2)^\omega \to C^\omega$ is an extension of $\mathrm{col} : Q \times A_1 \times A_2 \to C$ with notation overload.)

Factor-prefix completeness. Informally, W is factor-prefix complete if the following holds: if the prefixes of an infinite sequence occur as factors arbitrarily far in the tail of a second sequence in W, the first sequence is also in W. (A factor, aka substring, is a subsequence of consecutive elements.) Formally, $W \subseteq C^\omega$ is factor-prefix complete if the following holds: $\forall \boldsymbol{\gamma} \in C^\omega, (\exists \boldsymbol{\gamma}' \in W, \forall n, m \in \mathbb{N}, \exists k \in \mathbb{N}, \boldsymbol{\gamma}_{\leq n} = \boldsymbol{\gamma}'_{m+k} \cdots \boldsymbol{\gamma}'_{m+k+n}) \Rightarrow \boldsymbol{\gamma} \in W$.

In Theorem 7 below, a distribution is said to be positive if it assigns only positive masses. A (stochastic) strategy is said to be constant if it is a constant function, i.e. it returns always the same distribution, which is stronger than being Markovian (aka memoryless, positional).

▶ **Theorem 7** (semi-random determinacy). *Consider games* $g_q = \langle A_1, A_2, Q, q, \delta, C, \mathrm{col}, W\rangle$ *parametrized by* $q \in Q$*. If* A_1 *and* A_2 *are finite, if* W *is factor-prefix complete and closed under interleaving and prefix removal, the following are equivalent.*

1. *for all* $q \in Q$, Player 1 *has no winning strategies in* g_q*.*
2. *for all* $q \in Q$, Player 2 *has a constant, positive, stochastic strategy almost-sure winning* g_q*.*
3. *for all* $q \in Q$ *every* Player 2 *stochastic strategy involving probabilities bounded away from* 0 *(i.e. with positive infimum) almost-sure wins* g_q*.*

So in the setting of Theorem 7, either Player 1 has a winning strategy for some g_q, or every constant, positive strategy is almost-sure winning, hence the determinacy. Also note that semi-random determinacy implies the *bounded limit-one property* from [11] for the new games: if one state has positive value, one state has value one.

3.3 The special case of stateless (i.e. one-state) games

Stateless games. Intuitively, a *stateless game (with colors)* amounts to a game with only one state, i.e. $|Q| = 1$. Formally, it is a tuple $\langle A_1, A_2, C, \mathrm{col}, W\rangle$ such that A_1, A_2, and C are non-empty sets, $\mathrm{col} : A_1 \times A_2 \to C$ (as opposed to $\mathrm{col} : Q \times A_1 \times A_2 \to C$ in the general case), and $W \subseteq C^\omega$. Histories, runs, strategies, and induced histories are defined as in the general case . It is easier to extend the trace function in this context: $\mathrm{col}(\epsilon) := \epsilon$ and $\mathrm{col}(\rho \cdot (a, b)) = \mathrm{col}(\rho) \cdot \mathrm{col}(a, b)$.

Restricting Theorem 3 to stateless games yields a simpler Corollary 8 below. (Note that restricting Theorem 2 would yield a weaker variant of Corollary 8, i.e. additionally assuming closedness under prefix removal.) Memory size and algorithmic complexity estimates could be obtained essentially by replacing $|Q|$ with 1 in Theorem 3 and Corollary 6.

▶ **Corollary 8.** *Consider a game $\langle A_1, A_2, C, \mathrm{col}, W\rangle$ with finite C and interleaving-closed W. Let $C_1, \ldots, C_k$ be the elements of $\{\mathrm{col}(a, A_2) \mid a \in A_1\}$. The following are equivalent.*

1. Player 1 *has a winning strategy (resp. finite-memory winning strategy).*
2. $\forall (c_1, \ldots, c_k) \in C_1 \times \cdots \times C_k$, $W \cap \{c_1, \ldots, c_k\}^\omega \neq \emptyset$ *(resp. $W \cap \{c_1, \ldots, c_k\}^\omega \cap \mathrm{reg}_C \neq \emptyset$), where reg_C are the regular infinite sequences over C.*

Restricting Theorem 7 to stateless games cancels the universal quantification over states, but an even stronger version can be obtained: finiteness of A_1 and prefix removal closedness are dropped, and the assumption on factor-prefix completeness is weakened to factor-set completeness, as below.

Factor-set completeness. A language of infinite sequences is called *factor-set complete* if the following holds: if a sequence in the language has factors of unbounded length over some C_0, the language has a sequence over C_0. This is formally defined by contraposition: $W \subseteq C^\omega$ is factor-set complete if for all $C_0 \subseteq C$ and for all $\rho \in W$, we have $W \cap C_0^\omega = \emptyset \Rightarrow \forall \rho \in W, \exists m \in \mathbb{N}, \forall n \in \mathbb{N}, \exists i \in \mathbb{N}, i < m \wedge \rho_{n+i} \notin C_0$.

▶ **Observation 9.** *Factor-prefix completeness implies factor-set completeness (finite alphabets).*

▶ **Theorem 10** (Stateless semi-random determinacy)**.** *Consider a stateless game $\langle A_1, A_2, C, \mathrm{col}, W\rangle$ with finite C and A_2. Let us assume that W is interleaving-closed and factor-set complete. Then either* Player 1 *has a winning strategy, or every* Player 2 *constant, positive, stochastic strategy is almost-sure winning.*

4 The proofs

Theorems 2 and 3 characterize a concurrent game by finitely many one-player games. A natural idea would be to split their proof into two parts: first, reduce the problem to turn-based games via the well-known observation that a player has a winning strategy in a concurrent game iff she has one in the sequential version of the game where she plays first; second, use similar techniques as in [18]. For this to work, the sequential versions of the

	b_1	b_2
a_1	$1,1$	$2,1$
a_2	$1,-1$	$0,-1$
a_3	$-1,0$	$-2,0$

$1,1$	$1,1$	$1,1$	$1,1$	$2,1$
$1,-1$	$1,-1$	$0,-1$	$0,-1$	$1,-1$
$-1,0$	$-2,0$	$-1,0$	$-2,0$	$-1,0$

Figure 2 A concurrent Minkowski game and its derived games.

concurrent games must allow for colorless transitions, or a fresh color should be used for the transitions where Player 1 plays. This raises three issues: first, true colors should occur infinitely often in every run in these turn-based games, which would require a more complex notion of turn-based game (and considering only games with strict player alternation does not help, as this property is lost during the induction); second, the winning condition should be rephrased to take the fresh color into account, and so should its closedness properties; third, it would be much more difficult to obtain stronger results for the one-state concurrent games, since the one-state property may be hard to track through the translation into turn-based games. Instead, this article overcomes the concurrency directly thanks to Lemma 11.

▶ **Lemma 11.** *Let $(X_i)_{i\in I}$ be a family of sets. Then*
$\forall f : \prod_{i\in I} X_i \to I, \exists i \in I, \forall x \in X_i, \exists y \in \prod_{i\in I} X_i, y_i = x \wedge f(y) = i.$

Proof. Towards a contradiction, let us assume the negation of the claim, i.e. $\exists f : \prod_{i\in I} X_i \to I, \forall i \in I, \exists x \in X_i, \forall y \in \prod_{i\in I} X_i, y_i \neq x \vee f(y) \neq i$. By collecting one witness $x =: z_i$ for each i, one constructs $z \in \prod_{i\in I} X_i$ such that $\forall y \in \prod_{i\in I} X_i, y_i \neq z_i \vee f(y) \neq i$. In particular, taking $y := z$ yields $z_i \neq z_i \vee f(z) \neq i$ for all i, which contradicts the type of f. ◀

Consider the one-state game g in Figure 2 (to the left), where each cell encloses one vector of the real plane. Player 1's objective is that the sum of the outcome vectors remains bounded, which is closed under interleaving and prefix removal, so g is a concurrent version of the Minkowski games [24]. There are $2^3 = 8$ delayed responses, and five of the corresponding one-player games $g_0, \dots g_7$ are displayed to the right in Figure 2. Player 1 wins $g_0, \dots, g_7$, since for each $i \leq 7$ the vector $(0, 0)$ is in the convex hull of the three vectors defining g_j. The idea is to let Player 1 play g as if she were playing $g_0, \dots, g_7$ in parallel, more specifically in an interleaved way. Then, summing up the eight bounded trajectories yields a bounded trajectory for g.

The main difficulty to play the $g_0, \dots, g_7$ in an interleaved way is that at every stage, Player 1 should pick an action such that whichever action Player 2 chooses, the resulting vector is exactly the expected one by the (fixed) winning strategy for *some* g_j. Let $f : \{1, 2\}^3 \to \{a_1, \dots, a_3\}$ be the function that tells which action should be played currently in each of the $2^3 = 8$ one-player games. By Lemma 11 there exists an action a_i such that the following holds: if Player 2 chooses b_1, there exists g_j expecting the vector in the cell (a_i, b_1), and likewise if Player 2 chooses b_2, there exists g_k expecting the vector in the cell (a_i, b_2).

Let us now quickly mention semi-random determinacy. The proof of Theorem 7 below uses similar techniques as, e.g., a proof in [24].

Proof of 1 ⇒ 3 from Theorem 7. Let $p \in]0, \frac{1}{|A_2|}]$ and let τ be a Player 2 stochastic strategy that always assigns probability at least p to every action.

For all $q \in Q$, by contraposition of Theorem 2 let t_q be a delayed response (in g_q) such that Player 1 loses the one-player game $g_q(t_q)$. For all $n \in \mathbb{N}$, anytime a play reaches the

state q, the probability that from then on Player 2 follows t_q for n rounds in a row, as if second-guessing Player 1, is greater than or equal to p^n.

Consider a play where Player 2 follows τ. Let q be a state that is visited infinitely often. (Such a state exists since Q is finite.) Thanks to the argument above, for all $n \in \mathbb{N}$, the probability that, at some point, Player 2 follows t_q for n rounds in a row from q on is one. Since the countable intersection of measure-one sets has also measure one, the probability that, for all $n \in \mathbb{N}$, at some point Player 2 follows t_q for n rounds in a row from q on is one.

Let $(\rho^n)_{n\in\mathbb{N}}$ be the corresponding full histories. Since A_1 and A_2 are finite, the tree induced by prefix closure of the $(\rho^n)_{n\in\mathbb{N}}$ is finitely branching, so by Koenig's Lemma it has an infinite path $\boldsymbol{\rho}$, which corresponds to Player 2 following t_q infinitely many rounds in a row. So $\mathrm{col}(\boldsymbol{\rho}) \notin W$. By factor-prefix closedness the original play is also losing for Player 1, i.e. winning for Player 2. ◀

5 Applications

Abstract assumptions need not only be general, they also need to be practical. Section 5.1 shows that the closedness and completeness axioms enjoy nice algebraic properties: individually, w.r.t. Boolean combination, as well as collectively via the derived closure or completion operators. Section 5.2 mentions several classical or recent winning conditions from computer science and tells which of them satisfy the closedness and completeness axioms. Section 5.3 introduces the notion of bounded residual load as an alternative to the finitary fairness [2], and uses it to define a finitary variant of the ω-regular languages that satisfies the closedness and completeness axioms.

5.1 Algebraic properties of the closedness and completeness axioms

Lemma 12 below shows how the axioms behave w.r.t. Boolean combination.

▶ **Lemma 12.**

1. *The set of the factor-set complete languages is closed under union.*
2. *The set of the interleaving-closed languages is closed under intersection.*
3. *The set of the factor-prefix complete languages is closed under intersection and union.*

The set of the interleaving-closed languages is not closed under union: $\{0^\omega\}$ and $\{1^\omega\}$ are closed under interleaving (and by prefix removal), but $\{0^\omega, 1^\omega\}$ is not. The set of the interleaving-closed languages is not closed under complementation: the interleaving of two infinite sequences that are not eventually constant is not eventually constant, but interleaving the eventually constant sequences 0^ω and 1^ω may yield $(01)^\omega$. The set of the factor-set complete languages is not closed under intersection: indeed, both two-element sets $\{0(12)0(12)^20(12)^30\ldots, (12)^\omega\}$ and $\{0(12)0(12)^20\ldots, (112)^\omega\}$ are factor-set complete, but their intersection $\{0(12)0(12)^20\ldots\}$ is not. The set of the factor-set (-prefix) complete languages is not closed under complementation: $\{1^\omega\}$ is factor-set (-prefix) complete, but $\{0,1\}^\omega \setminus \{1^\omega\}$ is not.

The closedness under interleaving and prefix removal, and the factor-prefix completeness induce closure operators. If a relevant winning condition fails to satisfy an equaly relevant axiom, such an operator conveniently constructs a (more generous, axiom satisfying) variant of the winning condition. The closure by prefix removal of a set consists in adding the tails of the sequences from the set; the closure by interleaving consists in adding sequences obtained by interleaving the sequences from the set; and the factor-prefix completion consists in adding

the sequences whose prefixes occur arbitrarily far in a sequence from the set. Note that factor-set completeness does not induce a canonical closure operator due to the existential quantifier in its definition.

Lemma 13 below shows that the operators behave as expected. This is not for granted in general, as one may need to perform the addition operation an ordinal number of times. Here, one step suffices, which is convenient if computation is of concern.

▶ **Lemma 13.**

1. *Closure by prefix removal yields sets that are closed under prefix removal.*
2. *Closure by interleaving yields sets that are closed under interleaving*
3. *Factor-prefix completion yields sets that are factor-prefix complete.*

Lemma 14 shows that the operators preserve the existing properties. (Lemma 13 is invoked as a proof technique.)

▶ **Lemma 14.**

1. *Closure by prefix removal preserves closedness under interleaving.*
2. *Closure by prefix removal preserves factor-set and factor-prefix completeness.*
3. *Closure by interleaving preserves closedness under prefix removal.*
4. *Closure by interleaving preserves factor-set and factor-prefix completeness.*
5. *Factor-prefix completion preserves closedness under prefix removal.*

5.2 Concrete winning conditions

The non-comprehensive list below displays classical or recent winning conditions from computer science. It especially shows that new winning conditions obtained by conjunction of older winning conditions have been recently studied, e.g. in [7] and [4].

Parity $C := \{0, 1, \dots n\}$ for some $n \in \mathbb{N}$. A sequence is winning iff the least number occurring infinitely many times in the sequence is even.

Muller $C := \{0, 1, \dots n\}$ for some $n \in \mathbb{N}$. Let $M \subseteq \mathcal{P}(C)$ be a set of subsets of C. A sequence is winning iff the numbers occurring infinitely many times in the sequence constitute a set in M.

Mean-payoff $C = \mathbb{R}$, and a sequence is winning iff the limit superior of the partial sums is non-negative: $(u_n)_{n\in\mathbb{N}} \in \mathbb{R}^{\mathbb{N}}$ is winning iff $\limsup_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n} u_n \geq 0$. (Variants exist with limit inferior or positivity instead of non-negativity.)

Energy $C = \mathbb{R}$, and a sequence is winning iff its partial sums are non-negative: $(u_n)_{n\in\mathbb{N}} \in \mathbb{R}^{\mathbb{N}}$ is winning iff $\forall n \in \mathbb{N}, \sum_{i=0}^{n} u_n \geq 0$.

Boundedness [24] $C = \mathbb{R}^d$, and a sequence is winning iff its partial sums are uniformly bounded: $(u_n)_{n\in\mathbb{N}} \in (\mathbb{R}^d)^{\mathbb{N}}$ is winning iff $\exists b \forall n \in \mathbb{N}, \| \sum_{i=0}^{n} u_n \| \leq b$.

Discounted sum C is a bounded subset of $\mathbb{R}$. Let $0 < \alpha < 1$ and $t \in \mathbb{R}$. A sequence $(u_n)_{n\in\mathbb{N}} \in C^{\mathbb{N}}$ is winning iff $\sum_{n=0}^{+\infty} \alpha^n u_n \geq t$.

Energy-parity [7] $C := \mathbb{R} \times \{0, 1, \dots n\}$ for some $n \in \mathbb{N}$. The winning condition is the conjunction of the energy (first component) and the parity (second component) conditions.

Average energy [4] $C = \mathbb{R}$. The objective is to maintain a non-negative energy while keeping the average level of energy below a threshold $t \in \mathbb{R}$: a sequence $(u_n)_{n\in\mathbb{N}} \in \mathbb{R}^{\mathbb{N}}$ is winning iff $(\forall n \in \mathbb{N}, \sum_{i=0}^{n} u_n \geq 0) \wedge \limsup_{n\to+\infty} \frac{1}{n} \sum_{i=0}^{n} \sum_{j=0}^{i} u_j \leq t$.

▶ **Observation 15.**

1. *The parity, mean-payoff, energy, boundedness, energy-parity, and average energy conditions are all closed under interleaving. (It uses Lemma 12.2 to deal with energy-parity and average energy.)*

2. *The Muller and discounted sum conditions are not closed under interleaving.*
3. *The boundedness condition is factor-prefix complete; the others are not.*
4. *The energy condition (thus also energy-parity and average energy) and the discounted sum condition are not closed under prefix removal; the others are.*

▶ **Corollary 16.** *The turn-based safety-mean-payoff-parity games are half-positionally determined. (By Corollary 4 and Section 5.1.)*

It may be disappointing that the Muller condition is not even closed under interleaving, but Proposition 17 below extends Theorem 2 to the concurrent Muller games. Using results from [11] is likely to yield a better algorithmic complexity, though, but the point here is mainly that Theorem 2 can be extended.

▶ **Proposition 17.** *[Similar to [11]] Consider the finite games $\langle A_1, A_2, Q, q_0, \delta, C, \mathrm{col}, W\rangle$ where W is a Muller condition. Deciding the existence of a* Player 1 *winning (finite-memory) strategy can be done in big O of*

$$(|A_1||A_2||C||C|!)^2 \cdot (|Q||C|^2|C|!)^{|Q||C||C|!\left(2^{|Q||C|^2|C|!}\right)}$$

computation steps.

5.3 Bounded residual load

Unlike Theorems 2 and 3, Theorems 7 and 10 are not likely to be extended to include ω-regular languages. Before defining a variant of the ω-regular languages that satisfies the closedness and completeness properties from this article, let us consider notions of fairness that can be defined via a predicate S on $\mathbb{N} \times \mathbb{N} \times C^\omega$. Intuitively $S(n, d, \boldsymbol{\gamma})$ is supposed to mean that the sequence $\boldsymbol{\gamma}$ has satisfied, with delay at most d, a request that was formulated in $\boldsymbol{\gamma}$ at time n.

There are several reasonable ways to express the good behavior of an infinite sequence using the $S(n, d, \boldsymbol{\gamma})$. The classical definition of *fairness* requires that all problems be eventually solved (see F below), or cofinitely many problems (see FCI below), for a usual weakening that ensures prefix independence of the condition. Arguing that this kind of fairness gives no guarantee about response time, [11] strengthened fairness into *finitary fairness*, which requires the existence of a uniform bound on the waiting time (see FF below).

Yet another variant, *bounded residual load* (BRL), is introduced below. It says that $\boldsymbol{\gamma} \in C^\omega$ satisfies S wrt bounded residual load, if the number of problems that have currently not yet been solved is uniformly bounded over time.

1. $F(\boldsymbol{\gamma}) := \forall n \in \mathbb{N}, \exists d \in \mathbb{N},\ S(n, d, \boldsymbol{\gamma})$
2. $FCI(\boldsymbol{\gamma}) := |\{n \in \mathbb{N} \mid \forall d \in \mathbb{N},\ \neg S(n, d, \boldsymbol{\gamma})\}| < \infty$
3. $FF(\boldsymbol{\gamma}) := \exists d \in \mathbb{N}, \forall n \in \mathbb{N},\ S(n, d, \boldsymbol{\gamma})$
4. $BRL(\boldsymbol{\gamma}) := \exists b \in \mathbb{N}, \forall n \in \mathbb{N},\ b \geq |\{k \in \mathbb{N} \mid k \leq n \wedge \neg S(k, n-k, \boldsymbol{\gamma})\}|$

▶ **Observation 18.**

1. $FF(\gamma) \Rightarrow F(\gamma) \quad \wedge \quad F(\gamma) \Rightarrow FCI(\gamma)$
2. $FF(\gamma) \Rightarrow BRL(\boldsymbol{\gamma}) \quad \wedge \quad BRL(\boldsymbol{\gamma}) \Rightarrow FCI(\gamma)$
3. *F and BRL are incomparable in general.*

The finitary fairness and the like may be too strict for some applications: gladly accepting to wait b time units, but categorically refusing to wait $b+1$ time units sounds unusual indeed. Instead, the system (which is responsible for solving the problems) could pay a penalty for

each problem spending each time unit unsolved. Thanks to the bounded residual load, one has then the guarantee that the amount of money to be paid per time unit is bounded.

It is possible to combine the two ideas, though: by setting an acceptable response time and an acceptable uniform bound on the number of missed deadlines. This however, turns out to be equivalent to the simple BRL, which argues for the robustness of the concept.

▶ **Observation 19.** *Let* $BRLD(\boldsymbol{\gamma}) := \exists b, d \in \mathbb{N}, \forall n \in \mathbb{N},\ b \geq |\{k \in \mathbb{N} \mid k \leq n - d \wedge \neg S(k, n-k, \boldsymbol{\gamma})\}|$*, then* $BRLD(\boldsymbol{\gamma}) \Leftrightarrow BRL(\boldsymbol{\gamma})$*.*

A second justification for the BRL is that it has nice properties that the other notions of fairness lack when $S(n, d, \boldsymbol{\gamma})$ is defined to minic ω-regular languages, as shown below. Consider a non-empty set C of colors and a function $\mathcal{C} : C \to \mathcal{P}(C^*)$. A sequence $\boldsymbol{\gamma} \in C^\omega$ is said to satisfy $\mathcal{C}$ from position n after delay d, denoted $S_{\mathcal{C}}(n, d, \boldsymbol{\gamma})$, if the following holds.

$$\exists u \in \mathcal{C}(\boldsymbol{\gamma}_n), \exists (k_1, \dots, k_{|u|}) \in \mathbb{N}^{|u|},\ n < k_1 < \cdots < k_{|u|} \leq n + d \wedge \forall i \leq |u|,\ u_i = \boldsymbol{\gamma}_{k_i}$$

Intuitively, each color is a problem or a request, and the problem may be solved in several ways, each way consisting in enumerating suitable colors quickly. (This might very well correspond to the positive fragment of some bounded-time temporal logic.) To simulate the parity condition, one can set $C := \mathbb{N}$ and $\mathcal{C}(2n) := \{\{k\} \mid k \in \mathbb{N}\}$ and $\mathcal{C}(2n+1) := \{\{2k\} \mid k \in \mathbb{N} \wedge k \leq n\}$ for all $n \in \mathbb{N}$. The corresponding $BRL_{\mathcal{C}}$ is the parity condition with bounded residual load.

Lemma 20 below says that however $\mathcal{C}$ may be instantiated, all Theorems 2, 3, 7, and 10 can be applied with the $BRL_{\mathcal{C}}$ winning condition.

▶ **Lemma 20.** *For every non-empty set* C *of colors and every function* $\mathcal{C} : C \to \mathcal{P}(\mathbb{N}^C)$*, the winning condition* $BRL_{\mathcal{C}}$ *is closed under prefix removal and interleaving, and factor-prefix complete.*

Even when $\mathcal{C}$ simulates the parity condition as above, none of the corresponding $F_{\mathcal{C}}$, $FCI_{\mathcal{C}}$, or $FF_{\mathcal{C}}$ is both closed under interleaving and factor-set complete. $FF_{\mathcal{C}}$ is not closed under interleaving: $FF_{\mathcal{C}}((01)^\omega)$ and $FF_{\mathcal{C}}((23)^\omega)$, but $\neg FF_{\mathcal{C}}(\boldsymbol{\gamma})$, where $\boldsymbol{\gamma} := (23)01(23)^2 01 \dots 01(23)^n 01 \dots$ can be obtained by interleaving $(01)^\omega$ and $(23)^\omega$. $FCI_{\mathcal{C}}$ is not factor set-complete: $FCI_{\mathcal{C}}(\boldsymbol{\gamma})$, where $\boldsymbol{\gamma} := 101^2 01^3 \dots 01^n 0 \dots$, but $\neg FCI_{\mathcal{C}}(1^\omega)$ altough factors of 1's occur with arbitrary length in $\boldsymbol{\gamma}$. $F_{\mathcal{C}}$ is neither: first, $F_{\mathcal{C}}((10)^\omega)$ and $F_{\mathcal{C}}(2^\omega)$, but $\neg F_{\mathcal{C}}(1 \cdot 2^\omega)$, altough $1 \cdot 2^\omega$ can be obtained by interleaving $(10)^\omega$ and 2^ω; second, as above for $FCI_{\mathcal{C}}$. Note that the window-parity condition [8],[5] is not closed under interleaving either, as again exemplified by $(01)^\omega$ and $(23)^\omega$.

References

1. Rajeev Alur, Marco Faella, Sampath Kannan, and Nimit Singhania. Hedging Bets in Markov Decision Processes. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:20, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2016.29.
2. Rajeev Alur and Thomas A. Henzinger. Finitary fairness. *ACM Trans. Program. Lang. Syst.*, 20(6):1171–1194, 1998. doi:10.1145/295656.295659.
3. Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels. Concurrent games with ordered objectives. In Lars Birkedal, editor, *Foundations of Software Science and Computational Structures*, pages 301–315, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

4 Patricia Bouyer, Nicolas Markey, Mickael Randour, Kim G. Larsen, and Simon Laursen. Average-energy games. *Acta Informatica*, Jul 2016. doi:10.1007/s00236-016-0274-1.

5 Véronique Bruyère, Quentin Hautem, and Mickael Randour. Window parity games: an alternative approach toward parity games with time bounds. In *Proceedings of the Seventh International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2016, Catania, Italy, 14-16 September 2016.*, pages 135–148, 2016. doi:10.4204/EPTCS.226.10.

6 Krishnendu Chatterjee. Concurrent games with tail objectives. *Theoretical Computer Science*, 388(1):181–198, 2007. doi:10.1016/j.tcs.2007.07.047.

7 Krishnendu Chatterjee and Laurent Doyen. Energy parity games. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming*, pages 599–610, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

8 Krishnendu Chatterjee, Laurent Doyen, Mickael Randour, and Jean-François Raskin. Looking at mean-payoff and total-payoff through windows. *Inf. Comput.*, 242:25–52, 2015. doi:10.1016/j.ic.2015.03.010.

9 Krishnendu Chatterjee, Thomas A. Henzinger, and Florian Horn. Finitary winning in Ω-regular games. *ACM Trans. Comput. Logic*, 11(1):1:1–1:27, nov 2009. doi:10.1145/1614431.1614432.

10 Thomas Colcombet and Damian Niwiński. On the positional determinacy of edge-labeled games. *Theoretical Computer Science*, 352(1):190–196, 2006. doi:10.1016/j.tcs.2005.10.046.

11 Luca de Alfaro and Thomas A. Henzinger. Concurrent omega-regular games. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*, pages 141–154, 2000. doi:10.1109/LICS.2000.855763.

12 Nachum Dershowitz, D. N. Jayasimha, and Seungjoon Park. *Bounded Fairness*, pages 304–317. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. doi:10.1007/978-3-540-39910-0_14.

13 Hugo Gimbert and Florian Horn. Solving simple stochastic tail games. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 847–862, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1873601.1873670.

14 Hugo Gimbert and Wiesław Zielonka. Games where you can play optimally without any memory. In *CONCUR 2005 - Concurrency Theory*, volume 3653 of *Lecture Notes in Computer Science*, pages 428–442. Springer Berlin Heidelberg, 2005.

15 Julian Gutierrez, Aniello Murano, Giuseppe Perelli, Sasha Rubin, and Michael Wooldridge. Nash equilibria in concurrent games with lexicographic preferences. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1067–1073, 2017. doi:10.24963/ijcai.2017/148.

16 John Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8(2):135–159, 1979. doi:10.1016/0304-3975(79)90041-0.

17 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.

18 Eryk Kopczyński. Half-positional determinacy of infinite games. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, pages 336–347, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

19 Stéphane Le Roux. Infinite sequential Nash equilibrium. *Logical Methods in Computer Science*, 9, 2013. Special Issue for the Conference "Computability and Complexity in Analysis" (CCA 2011).

20 Stéphane Le Roux. From winning strategy to nash equilibrium. *Math. Log. Q.*, 60(4-5):354–371, 2014. doi:10.1002/malq.201300034.

21 Stéphane Le Roux. Infinite subgame perfect equilibrium in the hausdorff difference hierarchy. In *Topics in Theoretical Computer Science - The First IFIP WG 1.8 International Conference, TTCS 2015, Tehran, Iran, August 26-28, 2015, Revised Selected Papers*, pages 147–163, 2015. doi:10.1007/978-3-319-28678-5_11.

22 Stéphane Le Roux and Arno Pauly. Infinite sequential games with real-valued payoffs. In *Proceedings of LiCS*, 2014.

23 Stéphane Le Roux and Arno Pauly. Extending finite memory determinacy to multiplayer games. In *Proceedings of the 4th International Workshop on Strategic Reasoning, SR 2016, New York City, USA, 10th July 2016.*, pages 27–40, 2016. doi:10.4204/EPTCS.218.3.

24 Stéphane Le Roux, Arno Pauly, and Jean-François Raskin. Minkowski games. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 50:1–50:13, 2017. doi:10.4230/LIPIcs.STACS.2017.50.

25 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975. URL: http://www.jstor.org/stable/1971035.

26 Donald A. Martin. The determinacy of blackwell games. *Journal of Symbolic Logic*, 63(4):1565–1581, 1998. doi:10.2307/2586667.

27 Stéphane Le Roux and Arno Pauly. Extending finite-memory determinacy to multi-player games. *Information and Computation*, pages –, 2018. doi:10.1016/j.ic.2018.02.024.

28 Wolfgang Thomas. Languages, automata, and logic. In Salomaa A. Rozenberg G., editor, *Handbook of Formal Languages*. Springer, Berlin, Heidelberg, 1997.

Low Rank Approximation of Binary Matrices: Column Subset Selection and Generalizations

Chen Dan[1]
Carnegie Mellon University, Pittsburgh, Pennsylvania, United States
cdan@cs.cmu.edu

Kristoffer Arnsfelt Hansen
Department of Computer Science, Aarhus University, Aarhus, Denmark
arnsfelt@cs.au.dk
https://orcid.org/0000-0002-1155-8072

He Jiang[1]
University of Southern California, Los Angeles, California, United States
jian567@usc.edu
https://orcid.org/0000-0002-4902-2206

Liwei Wang[2]
1. Key Laboratory of Machine Perception, MOE, School of EECS, Peking University;
2. Center for Data Science, Peking University, Beijing Institute of Big Data Research;
Beijing, China
wanglw@cis.pku.edu.cn

Yuchen Zhou[1]
Department of Statistics, University of Wisconsin-Madison, Madison, Wisconsin, United States
yuchenzhou@stat.wisc.edu

Abstract

Low rank approximation of matrices is an important tool in machine learning. Given a data matrix, low rank approximation helps to find factors, patterns, and provides concise representations for the data. Research on low rank approximation usually focuses on real matrices. However, in many applications data are binary (categorical) rather than continuous. This leads to the problem of low rank approximation of binary matrices. Here we are given a $d \times n$ binary matrix $\mathbf{A}$ and a small integer $k < d$. The goal is to find two binary matrices $\mathbf{U}$ and $\mathbf{V}$ of sizes $d \times k$ and $k \times n$ respectively, so that the Frobenius norm of $\mathbf{A} - \mathbf{UV}$ is minimized. There are two models of this problem, depending on the definition of the dot product of binary vectors: The GF(2) model and the Boolean semiring model. Unlike low rank approximation of a real matrix which can be efficiently solved by Singular Value Decomposition, we show that approximation of a binary matrix is NP-hard, even for $k = 1$.

In this paper, our main concern is the problem of Column Subset Selection (CSS), in which the low rank matrix $\mathbf{U}$ must be formed by k columns of the data matrix, and we are interested in the approximation ratio achievable by CSS for binary matrices. For the GF(2) model, we show that CSS has approximation ratio bounded by $\frac{k}{2} + 1 + \frac{k}{2(2^k-1)}$ and this is asymptotically tight. For the Boolean model, it turns out that CSS is no longer sufficient to obtain a bound. We then develop a Generalized CSS (GCSS) procedure in which the columns of $\mathbf{U}$ are generated from Boolean formulas operating bitwise on selected columns of the data matrix. We show that the approximation ratio achieved by GCSS is bounded by $2^{k-1} + 1$, and argue that an exponential dependency on k is seems inherent.

[1] Work done while at Peking University.
[2] Partially supported by National Basic Research Program of China (973 Program) (grant no. 2015CB352502).

43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018).
Editors: Igor Potapov, Paul Spirakis, and James Worrell; Article No. 41; pp. 41:1–41:16

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

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis, Theory of computation → Unsupervised learning and clustering, Computing methodologies → Factorization methods,

Keywords and phrases Approximation Algorithms, Low Rank Approximation, Binary Matrices

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.41

Related Version A full version of the paper is available at [12], `http://arxiv.org/abs/1511.01699`.

1 Introduction

Low rank approximation of matrices is a classical problem. Given a matrix $\mathbf{A}$ of size $d \times n$, the goal is to find two low rank matrices $\mathbf{U}$ and $\mathbf{V}$, such that $\mathbf{UV}$ approximates $\mathbf{A}$. Formally, the problem is to solve the equation $\min_{\mathbf{U},\mathbf{V}} \|\mathbf{A} - \mathbf{UV}\|_F^2$, where the minimum is over all matrices $\mathbf{U}, \mathbf{V}$ of sizes $d \times k$ and $k \times n$ respectively. The parameter k, typically a small integer, is the desired rank. The error is measured in terms of the Frobenius norm $\|\cdot\|_F$.

In many applications, $\mathbf{A}$ is a *data matrix*: Each column of $\mathbf{A}$ is a d-dimensional data vector, and each row of $\mathbf{A}$ corresponds to an attribute. In the literature, low rank approximation of $\mathbf{A}$ is often called factor analysis or dimensionality reduction: the k columns of the matrix $\mathbf{U}$ are the *factors* or *basis vectors* of the low dimensional space, and each column of $\mathbf{V}$ contains the combination coefficients.

If $\mathbf{A}$, $\mathbf{U}$, $\mathbf{V}$ are real matrices, low rank approximation can be efficiently solved by Singular Value Decomposition (SVD). This problem has been studied for more than a century, and is known as Principal Component Analysis (PCA) [28], Karhunen-Loève Transform [30], to name a few.

In this paper we consider low rank approximation of binary matrices. The motivation is that in many applications data are binary (categorical) rather than continuous. Indeed, nearly half of the data sets in the UCI repository contains categorical features. In the binary case, we require that the data matrix $\mathbf{A}$ *as well as* the rank-k matrices $\mathbf{U}, \mathbf{V}$ are binary. There are two natural formulations of the binary low rank approximation problem, depending on the definition of vector dot product. One formulation will be referred to as the GF(2) model, in which the dot product of two binary vectors $\mathbf{u}, \mathbf{v}$ is defined as $\mathbf{u}^T\mathbf{v} := \oplus_i u_i v_i$. The other formulation will be referred to as the *Boolean* model, in which the dot product is defined as $\mathbf{u}^T\mathbf{v} := \bigvee_i (u_i \wedge v_i)$.

The Boolean model is usually called Boolean Factor Analysis (BFA). It has found numerous applications in machine learning and data mining including latent variable analysis, topic models, association rule mining, clustering, and database tiling [3, 33, 38, 40, 44]. The GF(2) model, while being less studied, has been applied to Independent Component Analysis (ICA) over string data, attracting attention from the signal processing community [25, 35, 48].

Despite of various applications and heuristic algorithms [19, 21, 31, 33], little is known from a theoretical point of view about the binary low rank approximation problem. In fact, previously the only known result is that for the very special case of $k = 1$ (where the GF(2) and the Boolean model are equivalent) there are 2-approximation algorithms (see Section 1.1).

In this paper, we provide the *first* theoretical results for the general binary low rank approximation problem, which is formally stated as follows. Given $\mathbf{A} \in \{0,1\}^{d\times n}$, solve

$$\min_{\mathbf{U}\in\{0,1\}^{d\times k},\mathbf{V}\in\{0,1\}^{k\times n}} \|\mathbf{A}-\mathbf{UV}\|_F^2. \tag{1}$$

where the matrix product $\mathbf{UV}$ is over GF(2) or the Boolean semiring respectively.

Before stating the results, let us first consider the differences between low rank approximation of real matrices and our GF(2) and Boolean models. First, the linear space over GF(2) has a very different structure from the Euclidean space. The dot product over GF(2) is not an inner product and does not induce a norm: there exists $\mathbf{a} \neq \mathbf{0}$ such that $\mathbf{a}^T\mathbf{a} = 0$ over GF(2). An immediate consequence is that for binary matrices of the GF(2) model, there is no Singular Value Decomposition (SVD), which is the basis for low rank approximation of real matrices. The Boolean model is even more different: As it is a semiring rather than a field, we do not even have a linear space (see below for details).

Thus the methodologies from the setting of real matrices do not carry over to the setting of binary matrices. In fact, we will show that finding the exact solution of (1) is NP-hard even for $k = 1$ (see Section 4). This result was obtained independently by Gillis and Vavasis [22].

Another well-studied approach for low rank approximation of matrices is Column Subset Selection (CSS) [20, 32]. The goal of CSS is to find a subset of k columns of $\mathbf{A}$ and form the low rank basis matrix so that the residual is as small as possible. An advantage of CSS is that the result is more interpretable than that of SVD. CSS has been extensively studied for low rank approximation of real matrices [1, 5, 6, 10, 11, 13–16, 24, 36, 43, 46, 47]. Below is a formal definition of CSS over real matrices.

▶ **Definition 1** (CSS for real matrices)**.** Given a matrix $\mathbf{A} \in \mathbb{R}^{d\times n}$ and a positive integer k, pick k columns of $\mathbf{A}$ forming a matrix $\mathbf{P}_A \in \mathbb{R}^{d\times k}$ such that the residual

$$\|\mathbf{A}-\mathbf{P}_A\mathbf{Q}\|_\xi$$

is minimized over all possible $\binom{n}{k}$ choices for the matrix $\mathbf{P}_A$. Here $\mathbf{Q}$ denotes the optimal matrix of size $k \times n$ given $\mathbf{P}_A$, which can be obtained by solving a least squares problem, and $\xi = 2$ or F denotes the spectral norm or Frobenius norm.

The central problem in CSS is to determine the best function $\phi(n, k)$ of n, k satisfying

$$\|\mathbf{A}-\mathbf{P}_A\mathbf{Q}\|_\xi^2 \leq \phi(k,n)\|\mathbf{A}-\mathbf{A}_k\|_\xi^2 \ , \tag{2}$$

where $\mathbf{A}_k$ denotes the best rank-k approximation to the matrix $\mathbf{A}$ as computed with SVD. Two classical results [14, 24] shows that for real matrices we have

$$\|\mathbf{A}-\mathbf{P}_A\mathbf{Q}\|_2^2 \leq (k(n-k)+1)\|\mathbf{A}-\mathbf{A}_k\|_2^2 \ , \tag{3}$$

$$\|\mathbf{A}-\mathbf{P}_A\mathbf{Q}\|_F^2 \leq (k+1)\|\mathbf{A}-\mathbf{A}_k\|_F^2 \ . \tag{4}$$

There is extensive work on developing efficient algorithms for CSS with approximation ratio close to the above bounds, possibly using more than k columns of $\mathbf{A}$. These include methods such as rank revealing QR [36], adaptive sampling [15], subspace sampling (leverage scores) [6, 16], efficient volume sampling [13], projection-cost preserving sketches [11] and greedy CSS [1].

In this work, we study the CSS problem for binary matrices over GF(2) and Boolean semiring respectively. We consider the central problem expressed by Eq. (2) and aiming to determine the best $\phi(k, n)$. We only consider the Frobenius norm, since the spectral norm does not exist in the GF(2) and Boolean models.

The difficulty of the CSS problem for GF(2) and Boolean semiring model is that all methods developed for CSS over real matrices rely on at least one of the following concepts which are intrinsic to the Euclidean space: SVD, volume of a simplex, Euclidean distance, orthogonal projection, and QR decomposition. However, none of these concept exists in the GF(2) or Boolean models.

In this paper, we develop new methods for the CSS problem for GF(2) and Boolean model respectively. For GF(2) model, we show that by picking the best k columns of $\mathbf{A}$ to form $\mathbf{P}_A$, we achieve the bound

$$\|\mathbf{A} - \mathbf{P}_A\mathbf{Q}\|_F^2 \leq \left(\frac{k}{2} + 1 + \frac{k}{2(2^k - 1)}\right) \|\mathbf{A} - \mathbf{A}_k\|_F^2,$$

where $\mathbf{A}_k = \mathbf{UV}$ is the optimal solution of (1). Moreover, we show that the ratio is asymptotically tight.

For Boolean model, it turns out that basic CSS is no longer sufficient for obtaining a bound, simply because the Boolean semiring is not a field. We instead propose a Generalized CSS (GCSS) procedure. In this GCSS framework, we select a larger number of columns of $\mathbf{A}$ and potential basis matrices $\mathbf{P}_A$ are generated from these using carefully designed Boolean formulas operating bitwise on the chosen columns of $\mathbf{A}$. We show that GCSS based on $(2^k - 1)$ columns of $\mathbf{A}$ achieves approximation ratio $(2^{k-1} + 1)$ relative to $\|\mathbf{A} - \mathbf{A}_k\|_F^2$. Moreover, we argue that the exponential dependence in k seems inherent with the Boolean model (see Section 3 for details).

Our work is a first step towards a good understanding of low rank approximation of matrices over GF(2) and the Boolean semiring. While our work gives approximation algorithms for low rank approximation for both the GF(2) and the Boolean model, our work is should mainly by viewed as existence results for (Generalized) CSS for binary matrices, parallel to the classical existence theorems [14, 24] for CSS of real matrices stated in Eq. (3) and (4). Moreover, as SVD does not apply to the GF(2) or Boolean model, CSS is so far the only method that obtains a low rank approximation for binary matrices with theoretical guarantees and deserves an in-depth study. Finally, it is an important future direction to develop efficient algorithms to achieve or approximately achieve the ratios obtained in this paper. We believe this requires new techniques futher exploiting the algebraic structure of GF(2) and the Boolean semiring.

The rest of this paper is organized as follows. In Section 1.1 we discuss existing results on low rank approximation of binary matrices. In Section 2 we present the information-theoretically optimal upper bound for the approximation ratio of CSS over GF(2). In Section 3 we propose the GCSS procedure and give the upper bound for the Boolean semiring model. In Section 4 we show that finding the exaction solution of the low rank binary matrix approximation problem is NP-hard even for $k = 1$. Finally we give our conclusion in Section 5.

1.1 Other Related Works

To the best of our knowledge, all known theoretical results on the low rank approximation problem are about the special case of rank-one, i.e., $k = 1$. In the rank-one case, one looks for binary vectors $\mathbf{u}$, $\mathbf{v}$ such that $\|\mathbf{A} - \mathbf{uv}^T\|_F$ is minimized, and the GF(2) and Boolean models are therefore equivalent.

Shen et al. [39] formulate the rank-one problem as an integer linear program and showed that solving its linear programming relaxation yields a 2-approximation algorithm. They also improved the efficiency by reducing the linear program to a max-flow problem using

a technique developed in [26]. Jiang et al. [29] observed that for the rank-one case, simply choosing the best column from $\mathbf{A}$ yields a 2-approximation algorithm.

In the GF(2) model, low rank approximation is related to the concept of matrix rigidity introduced by Valiant [45], as a method of proving lower bounds for linear circuits. For a matrix $\mathbf{A}$ over GF(2), the rigidity $R_{\mathbf{A}}(k)$ is the smallest number of entries of $\mathbf{A}$ that must be changed in order to bring its rank down to k. Thus for a $d \times n$ matrix $\mathbf{A}$, $R_{\mathbf{A}}(k)$ is *precisely* the minimum approximation error possible by a product of a $d \times k$ matrix $\mathbf{U}$ and a $k \times n$ matrix $\mathbf{V}$. By the results of Valiant, an $n \times n$ matrix $\mathbf{A}$ for which $R_{\mathbf{A}}(k) \geq n^{1+\varepsilon}$, for $k = O(n/\log\log n)$ and for some constant $\varepsilon > 0$ cannot be computed by a linear circuit of size $O(n)$ and depth $O(\log n)$. Such rigid matrices exists in abundance – the challenge is to come up with an explicit construction of a family of rigid matrices. For the low rank approximation problem we are however interested in the setting of $k \ll n$ and we are interested in algorithms rather than explicit matrices.

2 Column Subset Selection for Binary Matrices Over GF(2)

In this section we characterize the best possible approximation ratio of CSS in the GF(2) model. As mentioned in Section 1, the best approximation ratio of CSS for real matrices is $k+1$ under the Frobenius norm. This result is proved by the so-called volume sampling method [14]. Concretely, the volume sampling method randomly samples a set of k columns of $\mathbf{A}$ with probability proportional to the volume of the k-dimensional simplex formed by the k-columns along with the origin. Volume sampling generates an (expected) $k+1$ approximation ratio.

However, the GF(2) model does not have a notion of volume, since the dot product over GF(2) is not an inner product. Nevertheless, we develop a new approach and show the following bound.

▶ **Theorem 2.** *For any binary matrix* $\mathbf{A} \in \{0,1\}^{d\times n}$*, there exist* $\mathbf{P}_A \in \{0,1\}^{d\times k}$ *and* $\mathbf{Q} \in \{0,1\}^{k\times n}$*, where the columns of* $\mathbf{P}_A$ *are chosen from the columns of* $\mathbf{A}$*, such that*

$$\|\mathbf{A} - \mathbf{P}_A\mathbf{Q}\|_F^2 \leq \left(\frac{k}{2} + 1 + \frac{k}{2(2^k-1)}\right) \cdot \mathrm{OPT}_k,$$

where $\mathrm{OPT}_k := \|\mathbf{A} - \mathbf{A}_k\|_F^2$*, and* $\mathbf{A}_k = \mathbf{UV}$ *is the optimal solution of (1). Here all matrix operations are over* GF(2)*.*

Moreover, we show that the approximation ratio $\left(\frac{k}{2} + 1 + \frac{k}{2(2^k-1)}\right)$ is asymptotically tight.

▶ **Theorem 3.** *In the* GF(2) *model, for every* $k \geq 1$ *and every* $\epsilon > 0$*, there exists* $\mathbf{A}$ *such that*

$$\|\mathbf{A} - \mathbf{P}_A\mathbf{Q}\|_F^2 > \left(\frac{k}{2} + 1 + \frac{k}{2(2^k-1)} - \epsilon\right) \cdot \mathrm{OPT}_k,$$

for all $\mathbf{P}_A, \mathbf{Q}$*, where* $\mathbf{P}_A$ *are formed by* k *columns of* $\mathbf{A}$*.*

Below, we give a high level description of the proof of the theorems. Our method uses the structure of GF(2) and is different to the techniques developed for CSS of real matrices.

Consider the problem given by Eq. (1). Throughout this paper, we will call the matrix $\mathbf{U}$ the *basis matrix*, since its column vectors are the basis of the low dimensional space. Likewise we call the right matrix $\mathbf{V}$ the *coefficient matrix*, since its columns contain the linear combination coefficients. Let $\mathbf{U}$ and $\mathbf{V}$ be an optimal solution of Eq. (1), and let $\mathbf{u_1}, \ldots, \mathbf{u_k}$

be the k columns of $\mathbf{U}$. For each column $\mathbf{u}_i$ of the optimal basis matrix $\mathbf{U}$, consider its nearest neighbor among all the columns of $\mathbf{A}$. Let $\mathbf{a}_1, \ldots, \mathbf{a}_n$ be the n columns of $\mathbf{A}$, and denote by $\mathbf{a}_{(\mathbf{u}_i)}$ the nearest neighbor column of $\mathbf{u}_i$ in $\mathbf{A}$. Given an optimal basis matrix $\mathbf{U}$, we thus have a matrix $\mathbf{A}_{(\mathbf{U})} := (\mathbf{a}_{(\mathbf{u}_1)}, \ldots, \mathbf{a}_{(\mathbf{u}_k)})$, consisting of columns of $\mathbf{A}$. Note that the optimal solution of Eq.(1) is not unique. In fact, fixing an optimal basis matrix $\mathbf{U}$, for every matrix $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_k)$, $\mathbf{b}_i \in \{0,1\}^k$, if the rank[3] of $\mathbf{B}$ is k over GF(2), then $(\mathbf{UB}, \mathbf{B}^{-1}\mathbf{V})$ must also be an optimal solution. Each optimal basis matrix $\mathbf{UB}$ induces a nearest neighbor matrix $\mathbf{A}_{(\mathbf{UB})}$. We will show that there must exist a rank k matrix $\mathbf{B}$ such that the induced nearest neighbor matrix $\mathbf{A}_{(\mathbf{UB})}$, which when used as basis matrix, achieves an approximation error at most $(\frac{k}{2} + 1 + \frac{k}{2(2^k-1)})$ times that of the optimal solution $(\mathbf{UB}, \mathbf{B}^{-1}\mathbf{V})$. Let $\mathrm{Err}(\mathbf{b}_1, \ldots, \mathbf{b}_k)$ be the approximation error associated with the basis matrix $\mathbf{A}_{(\mathbf{UB})}$ for $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_k)$. Our goal is to bound the quantity

$$\min_{\mathbf{b}_1,\ldots,\mathbf{b}_k} \mathrm{Err}(\mathbf{b}_1, \ldots, \mathbf{b}_k) \ , \tag{5}$$

where $\mathbf{b}_i \in \{0,1\}^k$ for all $i \in [k]$.

Directly bounding Eq.(5) is prohibitive. The approach we take is to consider a sequence of $k+1$ error minimization problems. For the r-th ($0 \le r \le k$) minimization, we only optimize r vectors among $\mathbf{b}_1, \ldots, \mathbf{b}_k$ and keep the other $k-r$ vectors fixed. Given $\mathbf{b}_1, \ldots, \mathbf{b}_k$, let

$$\mathrm{Err}^{(0)}(\mathbf{b}_1, \ldots, \mathbf{b}_k) := \mathrm{Err}(\mathbf{b}_1, \ldots, \mathbf{b}_k), \tag{6}$$

$$\mathrm{Err}^{(r)}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r}) := \min_{\mathbf{b}\in\{0,1\}^k} \mathrm{Err}^{(r-1)}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r}, \mathbf{b}). \tag{7}$$

Note that $\mathrm{Err}^{(k)}()$ is exactly the quantity of Eq.(5).

Although the final goal is to bound the *ratio* between $\mathrm{Err}^{(k)}()$ and the error of the optimal solution of Eq.(1), we instead prove *additive* bounds for $\mathrm{Err}^{(r)}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r})$ for all $0 \le r \le k$. To be more precise, letting OPT_k be the error of the optimal solution of Eq.(1), we will show that $\mathrm{Err}^{(r)}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r})$ is bounded by OPT_k plus a term depending on r and $\mathbf{b}_1, \ldots, \mathbf{b}_{k-r}$ (Theorem 5). Then when $r = k$, this additive bound becomes a multiplicative bound with respect to OPT_k and gives the desired ratio. The reason for introducing $\mathrm{Err}^{(0)}, \ldots, \mathrm{Err}^{(k-1)}$ is that we make use of the relation between $\mathrm{Err}^{(r)}$ and $\mathrm{Err}^{(r-1)}$ to prove the bound. More precisely, is the additive bound proved by induction in r.

Although the relation of $\mathrm{Err}^{(r)}$ and $\mathrm{Err}^{(r-1)}$ is

$$\mathrm{Err}^{(r)}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r}) = \min_{\mathbf{b}} \mathrm{Err}^{(r-1)}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r}, \mathbf{b}),$$

directly optimizing $\mathbf{b}$ seems very difficult. Our approach is to use *weighted averaging*. Since for each $\mathbf{b} \in \{0,1\}^k$ it holds that,

$$\mathrm{Err}^{(r)}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r}) \le \mathrm{Err}^{(r-1)}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r}, \mathbf{b}),$$

we have that for any set of weights $w_{\mathbf{b}}$ such that $w_{\mathbf{b}} \ge 0$ and $\sum_{\mathbf{b}} w_{\mathbf{b}} = 1$,

$$\mathrm{Err}^{(r)}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r}) \le \sum_{\mathbf{b}} w_{\mathbf{b}} \mathrm{Err}^{(r-1)}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r}, \mathbf{b}).$$

We carefully choose the weights $w_{\mathbf{b}}$ to get a small upper bound. We perform weighted averaging in two layers. Consider the quotient space $\mathrm{GF}(2)^k/\mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r})$ and the coset $[\mathbf{b}] := \mathbf{b} + \mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r})$. In the first layer, we perform weighted averaging within

[3] Throughout this section, matrix inverse and matrix rank are all over GF(2).

each coset $[\mathbf{b}]$, and obtain a bound for $\mathrm{Err}^{(r)}$ depending on the coset. In the second layer we average over all cosets using another set of weights. We need different rules to set the weights in the two layers. Within a coset $[\mathbf{b}]$, we choose the weights as follows. Let $\mathbf{U}, \mathbf{V}$ be the already fixed optimal solution of Eq.(1). For each $\mathbf{c} \in [\mathbf{b}]$, let $n_{\mathbf{c}}$ denote the number of columns of $\mathbf{V}$ that are equal to $\mathbf{c}$. The weight we assign to $\mathbf{c}$ is proportional to $n_{\mathbf{c}}$. For the second layer, let

$$n_{[\mathbf{b}]} := \sum_{\mathbf{c} \in [\mathbf{b}]} n_{\mathbf{c}}$$

be the total number of columns of $\mathbf{V}$ that belong to the coset $[\mathbf{b}]$. We assign the weight to a coset $[\mathbf{b}]$ as follows. If

$$[\mathbf{b}] = \mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r}),$$

then the weight is set to be zero. Otherwise, we assign the weight to $[\mathbf{b}]$ proportional to

$$\frac{n_{[\mathbf{b}]}}{\sum_{[\mathbf{b}]} n_{[\mathbf{b}]} - \lambda n_{[\mathbf{b}]}},$$

where λ is a constant depending on r. Combining the two layers of averaging we obtain the additive bound and that implies the desired approximation ratio. This finishes the description of the proof of Theorem 2.

The lower bound in Theorem 3 is proved by explicit construction. We construct a matrix which is approximately low rank in the sense that it is the product of two rank-k matrix plus a very sparse matrix. The key ingredient of the proof is the construction of the two rank-k matrices, which have special structures so that the approximation ratio of column subset selection cannot be smaller than $\frac{k}{2} + 1 + \frac{k}{2(2^k-1)}$ significantly.

The additive bounds are stated in Theorem 5, which is technical. Below we first describe the notions that will appear in Theorem 5. These notions will also be frequently used in the proof as well. For clarity, we list the notions in two tables.

▶ **Definition 4.** For $1 \le r \le k$ and linear independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_r$ in $\{0,1\}^k$:

Table 1 Definitions for vector spans.

Definition	Explanation
$\mathrm{span}^c(\mathbf{b}_1, \ldots, \mathbf{b}_r) := \{0,1\}^k \setminus \mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_r)$	Complement of $\mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_r)$.
$\mathrm{span}^{\setminus i}(\mathbf{b}_1, \ldots, \mathbf{b}_r) := \mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{b}_{i+1}, \ldots, \mathbf{b}_r)$	Span of all vectors except the ith.

Let $\mathbf{A}$ be the matrix to be approximated and $(\mathbf{U}, \mathbf{V})$ be a fixed optimal solution of the problem in Eq.(1). For $\mathbf{u} \in \{0,1\}^d$, $\mathbf{c} \in \{0,1\}^k$, and $\mathcal{X} \subset \{0,1\}^k$:

Table 2 Definitions for errors and nearest neighbors.

Definition	Explanation
$\mathbf{a}_{(\mathbf{u})}$	The nearest neighbor of $\mathbf{u}$ among the columns of $\mathbf{A}$ (If more than one nearest neighbor, choose one arbitrarily.)
$\mathcal{J}_{\mathbf{c}} := \{j \in [n] : \mathbf{V}_j = \mathbf{c}\}$	The set of columns of $\mathbf{V}$ that are equal to vector $\mathbf{c}$.
$n_{\mathbf{c}} := \lvert\mathcal{J}_{\mathbf{c}}\rvert$	The number of columns of $\mathbf{V}$ that are equal to $\mathbf{c}$.
$L_{\mathbf{c}} := \sum_{j \in \mathcal{J}_{\mathbf{c}}} \lvert\mathbf{a}_j - \mathbf{U}\mathbf{c}\rvert$	The total approximation error of those columns in $\mathcal{J}_{\mathbf{c}}$.
$N_{\mathcal{X}} := \sum_{\mathbf{c} \in \mathcal{X}} n_{\mathbf{c}}$	The total number of columns of $\mathbf{V}$ that belong to set $\mathcal{X}$.
$M_{\mathbf{c}} = \begin{cases} \frac{L_{\mathbf{c}}}{n_{\mathbf{c}}} & n_{\mathbf{c}} > 0 \\ d & n_{\mathbf{c}} = 0 \end{cases}$	Upper bound of the average error of the columns in $\mathcal{J}_{\mathbf{c}}$.

Now we can state the additive bounds.

▶ **Theorem 5.** *Let* $\mathbf{b}_1, \ldots, \mathbf{b}_k$ *be* k *linear independent vectors in* $\{0,1\}^k$*. Then for each* $0 \leq r \leq k$*,*

$$\mathrm{Err}^r(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r}) \leq \mathrm{OPT}_k + \lambda_r \cdot \sum_{\mathbf{c} \in \mathrm{span}^c(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r})} L_{\mathbf{c}} + \sum_{i=1}^{k-r} f_i(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r}) M_{\mathbf{b}_i}, \quad (8)$$

where $M_{\mathbf{b}_i}$ *has been defined in Definition 4, and*

$$\lambda_r = \begin{cases} 0 & r = 0 \\ \frac{r}{2}\left(1 + \frac{1}{2^r - 1}\right), & 1 \leq r \leq k \end{cases}$$

and

$$f_i(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r}) = N_{\mathcal{X}} + \frac{1}{2} N_{\mathcal{Y}}, \quad (9)$$

here $\mathcal{X} = \mathbf{b}_i + \mathrm{span}^{\setminus i}(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r})$*, and* $\mathcal{Y} = \mathrm{span}^c(\mathbf{b}_1, \ldots, \mathbf{b}_{k-r})$*.*

The formal proof of Theorem 5 is lengthy and can be found in the full version of the paper [12]. Theorem 2 follows from Theorem 5 immediately.

Proof of Theorem 2. Let $r = k$ in Theorem 5. Then the last term in the RHS of Eq.(8) vanishes. The second term in the RHS of Eq.(8) becomes $\lambda_k \cdot \sum_{\mathbf{c} \in \{0,1\}^k} L_{\mathbf{c}}$. Observe that

$$\sum_{\mathbf{c} \in \{0,1\}^k} L_{\mathbf{c}} = \mathrm{OPT}_k,$$

and

$$1 + \lambda_k = \frac{k}{2} + 1 + \frac{k}{2(2^k - 1)},$$

the theorem follows. ◀

3 Generalized CSS Over Boolean Semiring

It is not difficult to see that the method developed for GF(2) model in the previous section does not apply to the Boolean model, simply because the Boolean semiring does not have a field structure. It turns out that, somewhat surprisingly, CSS is not sufficient to yield a bound relative to the optimal low rank solution in the Boolean model.

Here, we propose a Generalized CSS (GCSS) procedure. In GCSS, instead of using the columns of $\mathbf{A}$ directly to form $\mathbf{P}_A$, we apply carefully designed Boolean formulas (bitwise) to a predefined number of columns of $\mathbf{A}$ to form $\mathbf{P}_A$.

To illustrate the ideas, we first give an informal high level description of GCSS. We can capture our GCSS by the following framework, which we denote as an *oblivious basis generation scheme with advice*. Let $f(k)$ and $g(k)$ be functions of k. An oblivious basis generation scheme with *advice size* $f(k)$ and *column dependence size* $g(k)$ operates as follows. Given as input an advice string $\mathbf{o} \in \{0,1\}^{f(k)}$ the scheme outputs k Boolean formulas $\Phi_1, \ldots, \Phi_k$ each of $g(k)$ bits. Given $g(k)$ columns $\mathbf{a}_{i_1}, \ldots, \mathbf{a}_{i_{g(k)}}$ of the matrix $\mathbf{A}$, the k basis vectors $\mathbf{u}_1, \ldots, \mathbf{u}_k$ of $\mathbf{P}_A$ are constructed as

$$\mathbf{u}_j = \Phi_j(\mathbf{a}_{i_1}, \ldots, \mathbf{a}_{i_{g(k)}}),$$

where the Boolean function Φ_j is applied *entry-wise*. From such a basis generation scheme we immediately obtain an approximation result by iterating over all possible selections of $g(k)$ columns of $\mathbf{A}$ as well as all possible advice strings $\mathbf{o} \in \{0,1\}^{f(k)}$. We stress that the amount of information about $\mathbf{A}$ that can be supplied to the algorithm using the advice string is *independent* of the actual size of $\mathbf{A}$. Our construction of GCSS will have column dependence size $2^k - 1$ and advice size $O(k2^k)$ in which we encode an *ordering* of the given $2^k - 1$ columns. This results in an approximation ratio of $2^{k-1} + 1$.

To give a precise description of GCSS, it is more convenient to use sets instead of vectors as the representation. For a column $\mathbf{a}_i$ of $\mathbf{A}$, let

$$\mathcal{A}_i := \{j \in [d] : (\mathbf{a}_i)_j = 1\},$$

i.e., $\mathbf{a}_i$ is the characteristic vector of $\mathcal{A}_i$. Similarly, for an optimal solution $(\mathbf{U}, \mathbf{V})$ of the Boolean low rank approximation problem, let

$$\mathcal{U}_i := \{j \in [d] : (\mathbf{u}_i)_j = 1\},$$

and

$$\mathcal{V}_i := \{j \in [k] : v_{ij} = 1\}.$$

Thus in this section we will always think of a column of $\mathbf{A}$, $\mathbf{U}$ or $\mathbf{V}$ as a set. Given a set $\mathcal{S} \subset [k]$, let

$$\mathcal{J}_\mathcal{S} := \{j \in [n] : \mathcal{V}_j = \mathcal{S}\},$$

and $n_\mathcal{S} := |\mathcal{J}_\mathcal{S}|$. Using these notions, the Boolean product of $\mathbf{U}$ and a vector which is the characteristic vector of $\mathcal{S}$ will be denoted by $\mathcal{U}_\mathcal{S} := \bigcup_{i\in\mathcal{S}} \mathcal{U}_i$. Abusing the notion slightly, we shall still use $\mathcal{U}_i$ instead of $\mathcal{U}_{\{i\}}$ from now on. Like in the previous section, the nearest neighbor column of $\mathcal{U}_\mathcal{S}$ in $\mathbf{A}$ is defined by $\mathbf{a}_{(\mathcal{U}_\mathcal{S})}$. As we use set representation in this section, for notational simplicity we let $\mathcal{D}_\mathcal{S} \subset [d]$ be the set corresponding to this nearest neighbor column $\mathbf{a}_{(\mathcal{U}_\mathcal{S})}$, i.e.,

$$\mathcal{D}_\mathcal{S} := \{i \in [d] : \mathbf{a}_{(\mathcal{U}_\mathcal{S})_i} = 1\}$$

We are going to construct a rank-k solution $\mathcal{B}_1, \ldots, \mathcal{B}_k$, where $\mathcal{B}_i \subset [d]$ is the set representation of the column of the basis matrix. Once the basis matrix is obtained, the coefficient matrix can be calculated in the same way as in the previous section. The concrete GCSS procedure is described in Algorithm 1.

Now we can state the main result of this section.

▶ **Theorem 6.** *GCSS (as described above) has approximation ratio 2^k relative to the optimal solution of (1) over Boolean semiring.*

We now give the very high level idea of the proof. Fix a bijection π that satisfies $n_{S_1} \leq \cdots \leq n_{S_{2^k-1}}$. By construction the set $\mathcal{D}_{S_\ell}$ is the best approximation to $\mathcal{U}_{\mathcal{S}_\ell}$ given by a column of $\mathbf{A}$. Ideally the sets $\mathcal{B}_1, \ldots, \mathcal{B}_k$ should be such that $\bigcup_{i\in S_\ell} \mathcal{B}_i$ is a comparable substitute for all ℓ. What we instead will be able to achieve is that for all $\ell \in [2^k - 1]$

$$\mathcal{U}_{\mathcal{S}_\ell} \triangle \left(\bigcup_{i\in S_\ell} \mathcal{B}_i \right) \subseteq \left(\bigcup_{\ell' \geq \ell} (\mathcal{U}_{\mathcal{S}_{\ell'}} \triangle \mathcal{D}_{\mathcal{S}_{\ell'}}) \right) \tag{10}$$

where as seen from the algorithm the sets $\mathcal{B}_i$ are Boolean combinations of the sets $\mathcal{D}_{\mathcal{S}_\ell}$. Intuitively, we give more importance to approximating the columns of $\mathbf{A}$ from $\mathcal{J}_{\mathcal{S}_\ell}$ as ℓ increases. As the sizes $n_{\mathcal{S}_\ell}$ of these sets of columns also increase this means that we can account for the extra cost of possible poor approximation of the sets $\mathcal{U}_{\mathcal{S}_\ell}$ for smaller ℓ in terms of the approximation error of the sets $\mathcal{D}_{\mathcal{S}_{\ell'}}$ to $\mathcal{U}_{\mathcal{S}_{\ell'}}$ for larger $\ell' \geq \ell$.

Intuitively we should attempt to approximate all the sets $\mathcal{D}_{\mathcal{S}_\ell}$ simultaneously by $\bigcup_{i\in\mathcal{S}_\ell} \mathcal{B}_i$. But since we work over a semiring we will have to work with under-approximations. So for every ℓ we instead approximate the under-approximation $\bigcup_{i\in\mathcal{S}_\ell} \mathcal{E}_i^\ell$ of $\mathcal{D}_{\mathcal{S}_\ell}$. We do this by initially letting $\mathcal{B}_i = \mathcal{E}_i^1$ and then for each $\ell \in [2^k - 1]$ adding $\left(\bigcup_{i\in\mathcal{S}_\ell} \mathcal{E}_i^\ell\right) \setminus \left(\bigcup_{i\in\mathcal{S}_\ell} \mathcal{E}_i^1\right)$ to $\bigcup_{i\in S_\ell} \mathcal{B}_i$. This last step has to be done carefully piece by piece using the ordering of the sets $\mathcal{S}_1, \ldots, \mathcal{S}_{2^k-1}$. In the algorithm this is done using the sets $\mathcal{F}_i^{\ell_1,\ell_2}$.

The approximation ratio of GCSS over the Boolean semiring is $O(2^k)$, and thus much larger than that of GF(2). However we shall argue that this exponential dependency on k is not an artifact of proof technique, it seems inherent to the model.

Let k be even and let $n = 2^{k/2}$. We define the $n \times n$ matrix $\mathbf{A} = (a_{\alpha,\beta})$ indexed by strings $\alpha, \beta \in \{0,1\}^{k/2}$ by $a_{\alpha,\beta} = 1$ if and only if $\alpha \neq \beta$. Thus $\mathbf{A}$ is just the negation of the $n \times n$ identity matrix. It is well-known that the Boolean rank of $\mathbf{A}$ is equal to k. In particular, we can write $\mathbf{A}$ as the Boolean product of $\mathbf{U}$ and $\mathbf{V}$, where the columns of $\mathbf{U}$ and the rows of $\mathbf{V}$ are indexed by pairs (i, b) where $i \in [k/2]$ and $b \in \{0, 1\}$ and entry $(\alpha, (i, b))$ of $\mathbf{U}$ is 1 if and only if $\alpha_i = b$ and entry $((i, b), \beta)$ of $\mathbf{V}$ is 1 if and only if $\beta_i \neq b$. We note that the columns of $\mathbf{U}$ can be written as Boolean formulas applied entry-wise to (all of) the columns of $\mathbf{A}$. Since we consider approximation algorithms with *multiplicative error*, when supplied with input A and k our algorithm is required to compute an *exact factorization* of $\mathbf{A}$ into $n \times k$ and $k \times n$ matrices $\mathbf{U}$ and $\mathbf{V}$. If the underlying basis generation algorithm receives, say, only half of the columns of $\mathbf{A}$ it does not seem possible to compute such a factorization. It therefore seems that column dependence size at least $2^{k/2-1}$ is necessary, which is about the square-root of the column dependence size of our algorithm.

▶ Remark. Using the technique of weighted averaging developed for the GF(2) model, we can actually improve the approximation ratio to $2^{k-1} + 1$. We omit the details of the proof.

The proof of Theorem 6 can be found in the full version of the paper [12].

4 Hardness of Low Rank Approximation of Binary Matrices

Prior to our work, the computational complexity of the low rank approximation problem was not fully understood. For the rank-1 case, Tan showed that the equivalent problem Maximum Edge Weight Biclique for $\{-1, 1\}$-matrices is NP-hard under *randomized* reductions [42]. In the case when the rank k is unrestricted (i.e. part of the input) deciding whether there exist $\mathbf{U}$ and $\mathbf{V}$ such that $\mathbf{A} = \mathbf{UV}$ in the Boolean semiring model is precisely the NP-complete Minimal Set Basis problem [41], and that immediately implies that the approximation problem is NP-hard to approximate within *any* factor, as noted by Miettinen et al. [34]. On the other hand, this does not imply hardness when $k \ll d, n$. Indeed, the Minimal Set Basis problem is fixed-parameter tractable with parameter k, by a simple kernelization algorithm [17]. Note also that in the GF(2) model, deciding the existence of $\mathbf{U}$ and $\mathbf{V}$ such that $\mathbf{A} = \mathbf{UV}$ is efficiently solvable using Gaussian elimination, regardless of the rank k being unrestricted.

In this section we show the rank-1 Binary Matrix Approximation problem is NP-hard under normal polynomial time reduction. We first define two related problems. Let H be a *complete* bipartite graph with edge weight, and let $\mathbf{W} = (w_{ij})$ be the $d \times n$ matrix consisting

Algorithm 1 Generalized Column Subset Selection.

for all selection of $2^k - 1$ column vectors $\mathcal{A}_{j_1}, \mathcal{A}_{j_2}, \ldots, \mathcal{A}_{j_{2^k-1}}$ in $\mathbf{A}$ **do**
  for all bijections $\pi : [2^k - 1] \to (2^{[k]} \setminus \{\emptyset\})$ **do**
    Let $S_\ell = \pi(\ell)$ for $\ell \in [2^k - 1]$
    for $i \in [k]$ and $\ell \in [2^k - 1]$ **do**
      Compute

$$\mathcal{E}_i^\ell := \bigcap_{\substack{\ell' \geq \ell: \\ i \in \mathcal{S}_{\ell'}}} D_{S_l}$$

      where $D_{\mathcal{S}} = \mathcal{A}_{j_{\pi^{-1}(\mathcal{S})}}$ for $\emptyset \neq \mathcal{S} \subseteq [k]$.
    end for
    for $1 \leq \ell_1 < \ell_2 \leq 2^k - 1$ such that $i \in \mathcal{S}_{\ell_1} \cap \mathcal{S}_{l_2}$ **do**
      Compute

$$\mathcal{F}_i^{\ell_1,\ell_2} := \mathcal{E}_i^{\ell_1+1} \setminus \left[\bigcup_{i' \in \mathcal{S}_{\ell_2}} \mathcal{E}_{i'}^{\ell_1} \right].$$

    end for
    for $i \in [k]$ **do**
      Compute solution vector $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k\}$by

$$\mathcal{B}_i := \mathcal{E}_i^1 \cup \left(\bigcup_{\substack{\ell_1 < \ell_2: \\ i \in \mathcal{S}_{\ell_1} \cap \mathcal{S}_{\ell_2}}} \mathcal{F}_i^{\ell_1,\ell_2} \right).$$

    end for
  end for
  Compute the approximation error using the solution vector.
  if the approximation error is optimal **then**
    Save $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k\}$ as the output.
  end if
end for

of these edge weights. The Maximum Edge Weight Biclique problem is to find a biclique subgraph of H with maximizing total edge weight. As an optimization problem: maximize $\mathbf{x}^\mathsf{T}\mathbf{W}\mathbf{y}$, where $\mathbf{x} \in \{0,1\}^d$ and $\mathbf{y} \in \{0,1\}^n$. The Bipartite Max-Cut problem is to find a cut of the vertices of H maximum the total weight of the edges cut. As an optimization problem: maximize $\mathbf{x}^\mathsf{T}\mathbf{W}\mathbf{y}$, where $\mathbf{x} \in \{-1,1\}^d$ and $\mathbf{y} \in \{-1,1\}^n$. Note that these two problems differ only in the domain from which $\mathbf{x}$ and $\mathbf{y}$ are chosen.

Shen, Ji, and Ye [39] observed that the rank-1 Binary Matrix Approximation problem is equivalent to Maximum Edge Weight Biclique when all edge weights are chosen from $\{-1,1\}$. Namely, if $\mathbf{A}$ is a $d \times n$ Boolean matrix, $\mathbf{u} \in \{0,1\}^d$, and $\mathbf{v} \in \{0,1\}^n$, and let $\mathbf{J}_{d,n}$

denote the $d \times n$ all-1 matrix, we have

$$\begin{aligned}\|\mathbf{A} - \mathbf{u}\mathbf{v}^\mathsf{T}\|_F^2 &= \|\mathbf{A}\|_F^2 - 2\mathbf{u}^\mathsf{T}\mathbf{A}\mathbf{v} + \|\mathbf{u}\mathbf{v}^\mathsf{T}\|_F^2 \\ &= \|\mathbf{A}\|_F^2 - \mathbf{u}^\mathsf{T}(2\mathbf{A} - \mathbf{J}_{d,n})\mathbf{v}.\end{aligned}$$

Therefore, minimizing $\|\mathbf{A}-\mathbf{u}\mathbf{v}^\mathsf{T}\|_F^2$ is equivalent of maximizing $\mathbf{u}^\mathsf{T}(2\mathbf{A}-\mathbf{J}_{d,n})\mathbf{v}$. Also note that $(2\mathbf{A} - \mathbf{J}_{d,n})$ is a $\{-1,1\}$-matrix. Thus NP-hardness of Maximum Edge Weight Biclique with $\{-1,1\}$ edge weights implies NP-hardness of rank-1 Binary Matrix Approximation. To show the NP-hardness of Maximum Edge Weight Biclique, we consider reduction from the Bipartite Max-Cut problem.

Roth and Viswanathan showed that Bipartite Max-Cut is NP-hard even when all weights are chosen from the set $\{-1,1\}$ [37]. This is done by first showing NP-hardness when the weights are chosen from $\{-1,0,1\}$ and then reducing to the case of weights from $\{-1,1\}$.

Tan showed that Maximum Edge Weight Biclique is NP-hard [42] when weights are chosen from $\{-1,0,1\}$, and shows NP-hardness under randomized reductions when weights are chosen from $\{-1,1\}$. He leaves it as an open problem to obtain NP-hardness under normal polynomial time reductions. The complexity of this problem was also stated as an open problem by Amit [2]

The reduction from weights chosen from $\{-1,0,1\}$ to $\{-1,1\}$ by Roth and Viswanathan and by Tan is similar. The idea is to transform the $n \times n$ $\{-1,0,1\}$-weight matrix $\mathbf{W}$ into a new $nm \times nm$ $\{-1,1\}$-weight matrix $\mathbf{W}'$, where $\mathbf{W}'$ consists of $m \times m$ blocks corresponding to each entry of $\mathbf{W}$. A (-1)-entry is transformed into the all-(-1) $m \times m$ matrix, and similarly is a 1-entry transformed into the all 1 $m \times m$ matrix. But where Tan transforms a 0-entry to a *random* $m \times m$ $\{-1,1\}$-matrix, Roth and Viswanathan instead transforms a 0-entry into a $m \times m$ *Hadamard* matrix. We will show that this transformation into *Hadamard* matrix also work in the setting of the Maximum Edge Weight Biclique problem, thereby properly establishing its NP-hardness.

▶ **Theorem 7.** *The rank-*1 Binary Matrix Approximation *problem is* NP*-hard.*

We give a polynomial time many-one reduction from Maximum Edge Weight Biclique with weights from $\{-1,0,1\}$ to Maximum Edge Weight Biclique with weights from $\{0,1\}$, thereby showing the theorem. The proof is based on the following three lemmas.

The lemma below is an adaptation of [37, Lemma 4.2] from the $\{-1,1\}$ case to the $\{0,1\}$ case.

▶ **Lemma 8.** *Let* $\mathbf{W}$ *be an* $n \times n$ *matrix and let* $m \geq 1$*, and define* $\mathbf{W}' = \mathbf{W} \otimes \mathbf{J}_m$*, where* $\mathbf{J}_m := \mathbf{J}_{m,m}$*. Then*

$$\max_{\mathbf{u},\mathbf{v}} \mathbf{u}^\mathsf{T}\mathbf{W}'\mathbf{v} = m^2 \cdot \max_{\mathbf{x},\mathbf{y}} \mathbf{x}^\mathsf{T}\mathbf{W}\mathbf{y} \ ,$$

where $\mathbf{u},\mathbf{v} \in \{0,1\}^{mn}$ *and* $\mathbf{x},\mathbf{y} \in \{0,1\}^n$*, respectively. Furthermore, if* $\mathbf{x}$ *and* $\mathbf{y}$ *maximize* $\mathbf{x}^\mathsf{T}\mathbf{W}\mathbf{y}$*, then* $\mathbf{u} = \mathbf{x} \otimes \mathbf{1}_m$ *and* $\mathbf{v} = \mathbf{y} \otimes \mathbf{1}_m$ *maximize* $\mathbf{u}^\mathsf{T}\mathbf{W}'\mathbf{v}$*.*

Proof. Consider first $\mathbf{u} = \mathbf{x} \otimes \mathbf{1}_d$ and $\mathbf{v} = \mathbf{y} \otimes \mathbf{1}_m$. Then

$$\begin{aligned}\mathbf{u}^\mathsf{T}(\mathbf{W} \otimes \mathbf{J}_m)\mathbf{v} &= (\mathbf{x} \otimes \mathbf{1}_m)^\mathsf{T}(\mathbf{W} \otimes \mathbf{J}_m)(\mathbf{y} \otimes \mathbf{1}_m) \\ &= (\mathbf{x}^\mathsf{T}\mathbf{W}\mathbf{y}) \otimes (\mathbf{1}_m^\mathsf{T}\mathbf{J}_m\mathbf{1}_m) = m^2 \cdot (\mathbf{x}^\mathsf{T}\mathbf{W}\mathbf{y}) \ .\end{aligned}$$

Next, take $\mathbf{u}$ and $\mathbf{v}$ maximizing $\mathbf{u}^\mathsf{T}\mathbf{W}'\mathbf{v}$. We show that $\mathbf{u}$ and $\mathbf{v}$ can be brought to the form $\mathbf{u} = \mathbf{x} \otimes \mathbf{1}_m$ and $\mathbf{v} = \mathbf{y} \otimes \mathbf{1}_m$ without decreasing the value of $\mathbf{u}^\mathsf{T}\mathbf{W}'\mathbf{v}$. We first fix $\mathbf{v}$ and bring $\mathbf{u}$ to the desired form, and then similarly bring $\mathbf{v}$ to the desired form.

So fix $\mathbf{v}$, and let $\mathbf{z} = \mathbf{W}'\mathbf{v}$. Note that $\mathbf{u}$ maximizing $\mathbf{u}^\mathsf{T}\mathbf{z}$ must satisfy $\mathbf{u}_i = 1$ when $\mathbf{z}_i > 0$ and $\mathbf{u}_i = 0$ when $\mathbf{z}_i < 0$. Since $\mathbf{W}' = \mathbf{W} \otimes \mathbf{J}_m$ we have that $\mathbf{z}_{jm+1} = \mathbf{z}_{jm+2} = \cdots = \mathbf{z}_{(j+1)m}$ for all $j = 0, 1, \ldots, n-1$. Hence we can choose a maximizing $\mathbf{u}$ satisfying $\mathbf{u}_{jm+1} = \mathbf{u}_{jm+2} = \cdots = \mathbf{u}_{(j+1)m}$ for all $j = 0, 1, \ldots, n-1$ as well, meaning $\mathbf{u} = \mathbf{x} \otimes \mathbf{1}_m$ for suitable $\mathbf{x} \in \{0,1\}^n$. We can now fix $\mathbf{u}$ and in a similar way bring $\mathbf{v}$ to the form $\mathbf{v} = \mathbf{y} \otimes \mathbf{1}_m$ for suitable $\mathbf{y} \in \{0,1\}^n$. ◀

The following lemma, which is the $\{0,1\}$ analogue of [37, Lemma 4.3], is a direct consequence of Lindsey's Lemma. We state the proof for completeness.

▶ **Lemma 9.** *Let $\mathbf{H}$ be a $m \times m$ Hadamard matrix. For every $\mathbf{x}, \mathbf{y} \in \{0,1\}^m$, $|\mathbf{x}^\mathsf{T}\mathbf{H}\mathbf{y}| \leq m^{3/2}$.*

Proof. First note

$$\|\mathbf{H}\mathbf{y}\|^2 = \mathbf{y}^\mathsf{T}(\mathbf{H}^\mathsf{T}\mathbf{H})\mathbf{y} = \mathbf{y}^\mathsf{T}(m\mathbf{I})\mathbf{y} = m \cdot \|\mathbf{y}\|^2 \ .$$

We can then complete the proof by the Cauchy-Schwartz inequality,

$$|\mathbf{x}^\mathsf{T}\mathbf{H}\mathbf{y}| \leq \|\mathbf{x}^\mathsf{T}\| \cdot \|\mathbf{H}\mathbf{y}\| = \sqrt{m} \cdot \|\mathbf{x}\| \cdot \|\mathbf{y}\| \leq m^{3/2} \ . \qquad ◀$$

▶ **Lemma 10.** *Let $\mathbf{W} = (w_{ij})$ be a $n \times n$ $\{-1,0,1\}$-matrix and let $\mathbf{H}$ be a $m \times m$ Hadamard matrix. Define the $(mn) \times (mn)$ $\{-1,1\}$-block matrix $\widetilde{\mathbf{W}} = (\widetilde{\mathbf{W}}_{ij})$, where block $\widetilde{\mathbf{W}}_{ij}$ is given by*

$$\widetilde{\mathbf{W}}_{ij} = \begin{cases} w_{ij}\mathbf{J}_m & \text{if } w_{ij} \neq 0 \\ \mathbf{H} & \text{if } w_{ij} = 0 \end{cases} \ .$$

Let $\mathbf{W}' = \mathbf{W} \otimes \mathbf{J}_m$. Then for all $\mathbf{u}, \mathbf{v} \in \{0,1\}^{mn}$, $\left|\mathbf{u}^\mathsf{T}\widetilde{\mathbf{W}}\mathbf{v} - \mathbf{u}^\mathsf{T}\mathbf{W}'\mathbf{v}\right| \leq n^2 \cdot m^{3/2}$.

Proof. This is by simple estimation.

$$\begin{aligned} \left|\mathbf{u}^\mathsf{T}\widetilde{\mathbf{W}}\mathbf{v} - \mathbf{u}^\mathsf{T}\mathbf{W}'\mathbf{v}\right| &= \left|\mathbf{u}^\mathsf{T}(\widetilde{\mathbf{W}} - \mathbf{W}')\mathbf{v}\right| \\ &\leq n^2 \cdot \max_{\mathbf{x},\mathbf{y} \in \{0,1\}^m} \left|\mathbf{x}^\mathsf{T}\mathbf{H}\mathbf{y}\right| \\ &\leq n^2 \cdot m^{3/2} \ , \end{aligned}$$

where the last inequality follows from Lemma 9. ◀

Proof. of Theorem 7 Suppose now that $\mathbf{W}$ is an $n \times n$ $\{-1,0,1\}$-matrix. Let $m = 2^\ell$ be the smallest power of 2 that is greater than $4n^4$, and let $\mathbf{H}$ be the $m \times m$ Sylvester Hadamard matrix. We then define $\widetilde{\mathbf{W}}$ and $\mathbf{W}'$ as in Lemma 10. Then

$$\begin{aligned} &\left| \max_{\mathbf{u},\mathbf{v} \in \{0,1\}^{mn}} \mathbf{u}^\mathsf{T}\widetilde{\mathbf{W}}\mathbf{v} - m^2 \cdot \max_{\mathbf{x},\mathbf{y} \in \{0,1\}^n} \mathbf{x}^\mathsf{T}\mathbf{W}\mathbf{y} \right| \\ =& \left| \max_{\mathbf{u},\mathbf{v} \in \{0,1\}^{mn}} \mathbf{u}^\mathsf{T}\widetilde{\mathbf{W}}\mathbf{v} - \max_{\mathbf{u},\mathbf{v} \in \{0,1\}^{mn}} \mathbf{u}^\mathsf{T}\mathbf{W}'\mathbf{v} \right| \\ \leq& n^2 \cdot m^{3/2} \leq \frac{m^{1/2}}{2} \cdot m^{3/2} = \frac{m^2}{2} \ , \end{aligned}$$

where the first equality is by Lemma 8 and the first inequality is by Lemma 10.
Since the expression $m^2 \cdot \max_{\mathbf{x},\mathbf{y} \in \{0,1\}^n} \mathbf{x}^\mathsf{T}\mathbf{W}\mathbf{y}$ is an integer multiple of m^2, the value $\max_{\mathbf{u},\mathbf{v} \in \{0,1\}^{mn}} \mathbf{u}^\mathsf{T}\widetilde{\mathbf{W}}\mathbf{v}$ uniquely determines the value $\max_{\mathbf{x},\mathbf{y} \in \{0,1\}^n} \mathbf{x}^\mathsf{T}\mathbf{W}\mathbf{y}$. This then gives the desired reduction. ◀

5 Conclusion

We have studied Column Subset Section (CSS) for low rank binary matrix approximation. CSS is often used as an alternative approach of SVD for low rank approximation of real matrices, where the advantage of CSS is the interpretability of its results. For binary matrices, CSS is so far the only approach yielding theoretical guarantees, as solving the low rank problem exactly is NP-hard. We provide an upper bound on the approximation ratio of CSS for the GF(2) model and show the bound is tight. This is a complete characterization from an information-theoretic point of view. For the Boolean semiring model, we propose a Generalized CSS (GCSS) method, since CSS is not strong enough to yield a bound in this scenario. We also show an upper bound for GCSS.

CSS has been actively studied for nearly three decades and the first work can at least date back to [23], where it was called rank revealing QR in the numerical linear algebra community. The progress on CSS exhibits an interesting trajectory. Early results either gave bounds exponential in k or the running time of the algorithm is $O(n^k)$ [4, 7–9, 18, 27]. After efforts of many researches, there are now polynomial time algorithms that have polynomial bounds for the approximation ratio.

Our understanding of CSS for binary matrices is at the very beginning stage. It is an important problem for future work to develop efficient CSS algorithms that achieves or approximately achieves the bounds of this paper.

References

1 Jason Altschuler, Aditya Bhaskara, Gang Fu, Vahab Mirrokni, Afshin Rostamizadeh, and Morteza Zadimoghaddam. Greedy column subset selection: New bounds and distributed algorithms. *Proceedings of the 33rd International Conference on MachineLearning*, 2016.

2 Noga Amit. The bicluster graph editing problem. M.sc. thesis, Tel Aviv University, 2004.

3 Radim Belohlavek and Vilem Vychodil. Discovery of optimal factors in binary data via a novel method of matrix decomposition. *Journal of Computer and System Sciences*, 76(1):3–20, 2010.

4 Christian H Bischof and Gregorio Quintana-Ortí. Computing rank-revealing qr factorizations of dense matrices. *ACM Transactions on Mathematical Software (TOMS)*, 24(2):226–253, 1998.

5 Christos Boutsidis, Petros Drineas, and Malik Magdon-Ismail. Near-optimal column-based matrix reconstruction. *SIAM Journal on Computing*, 43(2):687–717, 2014.

6 Christos Boutsidis, Michael W Mahoney, and Petros Drineas. An improved approximation algorithm for the column subset selection problem. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 968–977. Society for Industrial and Applied Mathematics, 2009.

7 Tony F Chan. Rank revealing qr factorizations. *Linear algebra and its applications*, 88:67–82, 1987.

8 Tony F Chan and Per Christian Hansen. Low-rank revealing qr factorizations. *Numerical Linear Algebra with Applications*, 1(1):33–44, 1994.

9 Shivkumar Chandrasekaran and Ilse CF Ipsen. On rank-revealing factorisations. *SIAM Journal on Matrix Analysis and Applications*, 15(2):592–622, 1994.

10 Ali Civril and Malik Magdon-Ismail. Column subset selection via sparse approximation of svd. *Theoretical Computer Science*, 421:1–14, 2012.

11 Michael B Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceed-*

ings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, pages 163–172. ACM, 2015.

12 Chen Dan, Kristoffer Arnsfelt Hansen, He Jiang, Liwei Wang, and Yuchen Zhou. On low rank approximation of binary matrices. *CoRR*, abs/1511.01699, 2015.

13 Amit Deshpande and Luis Rademacher. Efficient volume sampling for row/column subset selection. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 329–338. IEEE, 2010.

14 Amit Deshpande, Luis Rademacher, Santosh Vempala, and Grant Wang. Matrix approximation and projective clustering via volume sampling. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1117–1126. Society for Industrial and Applied Mathematics, 2006.

15 Amit Deshpande and Santosh Vempala. Adaptive sampling and fast low-rank matrix approximation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 292–303. Springer, 2006.

16 Petros Drineas, Michael W Mahoney, and S Muthukrishnan. Relative-error cur matrix decompositions. *SIAM Journal on Matrix Analysis and Applications*, 30(2):844–881, 2008.

17 Herbert Fleischner, Egbert Mujuni, Daniël Paulusma, and Stefan Szeider. Covering graphs with few complete bipartite subgraphs. *Theor. Comput. Sci*, 410(21-23):2045–2053, 2009.

18 Leslie V Foster. Rank and null space calculations using matrix decomposition without column interchanges. *Linear Algebra and its Applications*, 74:47–71, 1986.

19 Mario Frank, Andreas P Streich, David Basin, and Joachim M Buhmann. Multi-assignment clustering for boolean data. *The Journal of Machine Learning Research*, 13(1):459–489, 2012.

20 Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *Journal of the ACM (JACM)*, 51(6):1025–1041, 2004.

21 Alexander A Frolov, Dusan Husek, Igor P Muraviev, and P Yu Polyakov. Boolean factor analysis by attractor neural network. *Neural Networks, IEEE Transactions on*, 18(3):698–707, 2007.

22 Nicolas Gillis and Stephen A. Vavasis. On the complexity of robust PCA and ℓ_1-norm low-rank matrix approximation. *CoRR*, abs/1509.09236, 2015.

23 Gene Golub. Numerical methods for solving linear least squares problems. *Numerische Mathematik*, 7(3):206–216, 1965.

24 Ming Gu and Stanley C Eisenstat. Efficient algorithms for computing a strong rank-revealing QR factorization. *SIAM Journal on Scientific Computing*, 17(4):848–869, 1996.

25 Harold W Gutch, Peter Gruber, Arie Yeredor, and Fabian J Theis. Ica over finite fields: separability and algorithms. *Signal Processing*, 92(8):1796–1808, 2012.

26 Dorit S Hochbaum and Anu Pathria. Forest harvesting and minimum cuts: a new approach to handling spatial constraints. *Forest Science*, 43(4):544–554, 1997.

27 Yoo Pyo Hong and C-T Pan. Rank-revealing QR factorizations and the singular value decomposition. *Mathematics of Computation*, 58(197):213–232, 1992.

28 Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

29 Peng Jiang, Jiming Peng, Michael Heath, and Rui Yang. A clustering approach to constrained binary matrix factorization. In *Data Mining and Knowledge Discovery for Big Data*, pages 281–303. Springer, 2014.

30 Kari K. Karhunen. über lineare methoden in der wahrscheinlichkeitsrechnung. *Ann. Acad. Sci. Fennicae. Ser. A. I. Math.-Phys.*, 37:1–79, 1947.

31 Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. Mining top-k patterns from binary datasets in presence of noise. In *SDM*, volume 10, pages 165–176, 2010.

32 Michael W Mahoney et al. Randomized algorithms for matrices and data. *Foundations and Trends® in Machine Learning*, 3(2):123–224, 2011.

33 Pauli Miettinen, Taneli Mielikainen, Aristides Gionis, Gautam Das, and Heikki Mannila. The discrete basis problem. *Knowledge and Data Engineering, IEEE Transactions on*, 20(10):1348–1362, 2008.

34 Pauli Miettinen, Taneli Mielikäinen, Aristides Gionis, Gautam Das, and Heikki Mannila. The discrete basis problem. *IEEE Trans. Knowl. Data Eng*, 20(10):1348–1362, 2008.

35 Amichai Painsky, Saharon Rosset, and Meir Feder. Generalized independent component analysis over finite alphabets. *Information Theory, IEEE Transactions on*, 2015.

36 C-T Pan. On the existence and computation of rank-revealing lu factorizations. *Linear Algebra and its Applications*, 316(1-3):199–222, 2000.

37 Ron M. Roth and Krishnamurthy Viswanathan. On the hardness of decoding the gale-berlekamp code. *IEEE Transactions on Information Theory*, 54(3):1050–1060, 2008.

38 Jouni K Seppänen, Ella Bingham, and Heikki Mannila. A simple algorithm for topic identification in 0–1 data. In *Knowledge Discovery in Databases: PKDD 2003*, pages 423–434. Springer, 2003.

39 Bao-Hong Shen, Shuiwang Ji, and Jieping Ye. Mining discrete patterns via binary matrix factorization. In John F. Elder IV, Françoise Fogelman-Soulié, Peter A. Flach, and Mohammed Javeed Zaki, editors, *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 757–766. ACM, 2009.

40 Tomáš Šingliar and Miloš Hauskrecht. Noisy-or component analysis and its application to link analysis. *The Journal of Machine Learning Research*, 7:2189–2213, 2006.

41 Larry Stockmeyer. The minimal set basis problem is NP-complete. IBM Research Report RC-5431, IBM Thomas J. Watson Research Center, 1975.

42 Jinsong Tan. Inapproximability of maximum weighted edge biclique and its applications. In Manindra Agrawal, Ding-Zhu Du, Zhenhua Duan, and Angsheng Li, editors, *TAMC 2008*, volume 4978 of *LNCS*, pages 282–293. Springer, 2008.

43 Joel A Tropp. Column subset selection, matrix factorization, and eigenvalue optimization. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 978–986. Society for Industrial and Applied Mathematics, 2009.

44 Jaideep Vaidya, Vijayalakshmi Atluri, and Qi Guo. The role mining problem: finding a minimal descriptive set of roles. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 175–184. ACM, 2007.

45 Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In Jozef Gruska, editor, *6th Symposium on Mathematical Foundations of Computer Science, MFCS 1977*, volume 53 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 1977.

46 Yining Wang and Aarti Singh. Column subset selection with missing data via active sampling. In *AISTATS*, pages 1033–1041, 2015.

47 Tianbao Yang, Lijun Zhang, Rong Jin, and Shenghuo Zhu. An explicit sampling dependent spectral error bound for column subset selection. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 135–143, 2015.

48 Arie Yeredor. Independent component analysis over galois fields of prime order. *Information Theory, IEEE Transactions on*, 57(8):5342–5359, 2011.

Optimal Strategies in Pushdown Reachability Games

Arnaud Carayol[1]
Université Paris-Est, LIGM (UMR 8049), CNRS, ENPC, ESIEE Paris, UPEM, France
arnaud.carayol@u-pem.fr

Matthew Hague[2]
Royal Holloway, University of London, UK
matthew.hague@rhul.ac.uk
https://orcid.org/0000-0003-4913-3800

Abstract

An algorithm for computing optimal strategies in pushdown reachability games was given by Cachat. We show that the information tracked by this algorithm is too coarse and the strategies constructed are not necessarily optimal. We then show that the algorithm can be refined to recover optimality. Through a further non-trivial argument the refined algorithm can be run in 2EXPTIME by bounding the play-lengths tracked to those that are at most doubly exponential. This is optimal in the sense that there exists a game for which the optimal strategy requires a doubly exponential number of moves to reach a target configuration.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Pushdown Systems, Reachability Games, Optimal Strategies, Formal Methods, Context Free

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.42

Acknowledgements We would like to thank the anonymous reviewers for their comments.

1 Introduction

Pushdown systems are popular models for program verification. They are equipped with an unbounded stack that can model the call stack of a procedural program. That is, the control flow of first-order recursive programs (such as C and Java programs) can be accurately modelled [10].

In a pushdown game, configurations of a pushdown system belong to one of two players (Elvis and the Anarchist). The player who owns a configuration chooses which configuration the game moves to next. In a reachability game, Elvis wins if he is able to force the play into a target configuration, while the Anarchist must prevent this from happening.

One may consider a reachability game to be a competition between a program (Elvis) and its environment (the Anarchist). The program is required to reach a good terminating configuration under all conditions presented by the (uncontrollable) environment. In this situation it is interesting to be able to determine the configurations from which Elvis is able to always win the game, and, moreover, the strategy he should employ. That is, how should

[1] Supported by the French National Research Agency (ANR), through the excellence program Bézout (ANR-10-LABX-58).
[2] Supported by EPSRC [EP/K009907/1].

the program behave in order to respond to the given inputs to ensure a correct execution. The problem of constructing a winning strategy for Elvis corresponds to synthesising a complete, correct program from a given program skeleton.

It is known that the players have *positional* winning strategies in a pushdown reachability game. That is a winning strategy needs only to have access to the current state of the game (as opposed to the entire history of play) [16]. A variety of methods are known for constructing winning strategies in pushdown reachability games [15, 4, 14, 12, 11, 8, 5]. One such method introduced by Cachat computes the *optimal* strategy [3]. That is, the strategy that will reach a target configuration in the fewest number of steps. Such a strategy has obvious applications in the synthesis of efficient programs.

Cachat's algorithm is based on the saturation technique. Saturation is a technique that, beginning from a finite automaton representing a set of configurations, repeatedly adds new transitions to the automaton. The goal is to expand the representation to include all configurations either reachable from, or that can reach, the initial set.

It was shown by Büchi that the set of configurations reachable from the initial configuration of a pushdown system form a regular language and hence can be represented by a finite state automaton [2]. While Büchi's procedure is exponential, Caucal showed that this problem can be solved in polynomial time [6]. The improved algorithm is a saturation process where transitions are incrementally added to a finite automaton. This technique was simplified and adapted to the model-checking setting by Bouajjani *et al.* in [1] and independently by Finkel *et al.* in [7]. In particular, it was shown that the set of predecessors of a regular set of target configurations is also regular. In the same work, the saturation method was shown to work for pushdown reachability games, though the complexity increases to EXPTIME, for which the problem is complete. Cachat builds on this algorithm by annotating each transition added to the finite automaton with a corresponding move in the pushdown game, as well as a weight indicating its "distance" from the target set.

Unfortunately, Cachat's algorithm contains a non-trivial error. In short, by keeping only a single weight per transition, Cachat loses important information about the cost of the different paths of execution through which the Anarchist may force play. This leads to the algorithm computing non-optimal weights for some choices of Elvis, meaning the recommended moves may no longer be optimal. In this work we present the following contributions:

- A counter-example showing how Cachat's algorithm may compute non-optimal strategies.
- A corrected saturation-based algorithm using weights that are fine-grained enough to compute optimal strategies precisely.

  Termination of this algorithm relies on well-quasi orders and we do not have an elementary bound on its runtime.
- A non-trivial proof that the above algorithm can be restricted to only include weights that are doubly exponential in size (whilst still computing optimal strategies). With such a restriction optimal strategies can be computed in 2EXPTIME.
- A matching lower bound giving a game in which Elvis's optimal strategy requires a doubly exponential number of moves to reach a target configuration.

We give preliminaries in Section 2 and the basic saturation algorithm in Section 3. Cachat's algorithm for optimal strategies appears in Section 4 along with our counter-example. We correct the algorithm in Section 5 with the complexity results in Section 6.

2 Preliminaries

2.1 Alternating Finite Automata

To analyse two player games we will make use of alternating finite automata to represent sets of configurations of a pushdown system. For convenience, we will often refer to alternating finite automata simply as automata.

▶ **Definition 2.1** (Alternating Automata). An *alternating automaton* is a tuple $\mathcal{A} = (\mathbb{S}, \Sigma, \mathcal{F}, \delta)$ where $\mathbb{S}$ is a finite set of states, Σ is a finite alphabet, $\mathcal{F} \subseteq \mathbb{S}$ is the set of accepting states, and $\delta \subseteq \mathbb{S} \times \Sigma \times 2^{\mathbb{S}}$ is a transition relation.

We write $s \xrightarrow{A} S$ for a transition (s, A, S) in δ. To simplify the presentation, we will assume that S is always non-empty. A run of an alternating automaton over $w = A_1 \dots A_\ell \in \Gamma^*$ is an unordered unranked tree of depth ℓ with nodes labelled by states in $\mathbb{S}$ and edges labelled by transitions in δ such that for each node η at depth $0 \leq i \leq \ell - 1$ labelled s there is a transition $\tau = s \xrightarrow{A_{i+1}} \{s_1, \dots, s_m\}$ and η has children $\eta_1, \dots, \eta_m$ labelled $s_1, \dots, s_m$ respectively and each edge (η, η_j) for all $1 \leq j \leq m$ is labelled by τ.

If the root of the run is labelled by s and the set of states appearing at the leaves of the run is S, we say that the run starts with s and ends in S. The run is accepting if $S \subseteq \mathcal{F}$. For a state s, the set $\mathcal{L}_s(\mathcal{A})$ contains all words over which there is an accepting run of $\mathcal{A}$ from s.

A branch in a run is a sequence of nodes $\eta_0 \cdots \eta_\ell$ with η_0 the root of the run, η_ℓ a leaf and for each $1 \leq i \leq \ell$, we have η_i is a child of η_{i-1}.

For a word $w \in \Sigma^*$, a state s and a set of states S, we write $\left[s \overset{w}{\underset{\mathcal{A}}{\Rightarrow}} S \right]$ for the set of all runs of $\mathcal{A}$ starting from s and ending precisely with S and $s \overset{w}{\underset{\mathcal{A}}{\Rightarrow}} S$ to denote the existence of a run of $\mathcal{A}$ over w starting from s and ending precisely in S.

2.2 Pushdown Reachability Games

A pushdown reachability game is played between two players on the configuration graph of a pushdown system. The owner of a configuration is indicated by its state and the set of target configurations is accepted by an alternating finite automaton.

▶ **Definition 2.2** (Pushdown Reachability Games). A two-player pushdown reachability game is a tuple $G = (\mathcal{Q}, \Gamma, \Delta, \mathcal{A})$ such that $\mathcal{Q}$ is a finite set of control states partitioned $\mathcal{Q} = \mathcal{Q}_E \uplus \mathcal{Q}_A$ into Elvis and the Anarchist states respectively, Γ is the finite stack alphabet, $\Delta \subseteq (\mathcal{Q} \times \Gamma) \times \left(\mathcal{Q} \times \Gamma^{\leq 2}\right)$ is the set of transitions, and $\mathcal{A}$ is an alternating finite automaton $(\mathbb{S}, \Gamma, \mathcal{F}, \delta)$ with $\mathcal{Q} \subseteq \mathbb{S}$.

We write $(q, A) \to (p, w)$ for the transition $((q, A), (p, w))$ in Δ. A configuration is a tuple (q, w) where q is a state in $\mathcal{Q}$ and w is a stack content in Γ^*. Let $\mathcal{C}_G$ be the set of configurations of G. In the configuration (q, Aw), it is possible to apply a transition $(q, A) \to (p, u)$ to go to the configuration (p, uw). A configuration (q, w) is final if the stack content w is accepted by $\mathcal{A}$ from the state q (*i.e.* $w \in \mathcal{L}_q(\mathcal{A})$).

A play of a pushdown game is a (possibly infinite) sequence $(q_0, w_0), (q_1, w_1), \dots$ where (q_0, w_0) is some starting configuration and (q_{i+1}, w_{i+1}) (if defined) is obtained from (q_i, w_i) via some transition $(q_i, A) \to (q_{i+1}, w) \in \Delta$. In the case where $q_i \in \mathcal{Q}_E$, it is Elvis who chooses the transition to apply, otherwise the Anarchist chooses the transition.

Elvis wins the game if there is some i such that (q_i, w_i) is final or if q_i belongs to the Anarchist and (q_i, w_i) does not have any successors. Otherwise, the Anarchist wins the play.

A strategy for Elvis is a partial function $\sigma : \mathcal{C}_G^* \to \Delta$ which may assign to each play $(q_0, w_0), \ldots, (q_\ell, w_\ell)$ with $q_\ell \in \mathcal{Q}_E$ a transition in Δ applicable to the configuration (q_ℓ, w_ℓ). A given play $(q_0, w_0), (q_1, w_1), \ldots$ is played according to σ if for all i such that $q_i \in \mathcal{Q}_E$ and (q_i, w_i) has a successor in the play, we have $\sigma((q_0, w_0), \ldots, (q_i, w_i)) = r$ and $(q_i, w_i) \to (q_{i+1}, w_{i+1})$ via transition r. A strategy is winning for Elvis from a configuration (q_0, w_0) if all maximal plays starting from (q_0, w_0) and according to σ are winning for Elvis. A strategy is called positional if its value only depends on the last configuration of the play. Hence a positional strategy is fully described by a mapping from the set of configurations belonging to Elvis to Δ. Winning strategies for the Anarchist are defined analogously.

The winning region W of a pushdown reachability game is the set of all configurations from which Elvis has a winning strategy.

It is well-known (see [9]) that $W = \mathrm{Pre}_G^*(\mathcal{A})$ defined as $\mathrm{Pre}_G^*(\mathcal{A}) = \bigcup_{i<\omega} \mathrm{Pre}_G^i(\mathcal{A})$ where

$$\begin{array}{rcl} \mathrm{Pre}_G^0(\mathcal{A}) & = & \{(q,w) \mid w \in \mathcal{L}_q(\mathcal{A})\} \\ \mathrm{Pre}_G^{i+1}(\mathcal{A}) & = & \mathrm{Pre}_G^i(\mathcal{A}) \cup \\ & & \{(q,w) \mid q \in \mathcal{Q}_E \wedge \exists\, (q,w) \to (q',w') \,.\, (q',w') \in \mathrm{Pre}_G^i(\mathcal{A})\} \cup \\ & & \{(q,w) \mid q \in \mathcal{Q}_A \wedge \forall\, (q,w) \to (q',w') \,.\, (q',w') \in \mathrm{Pre}_G^i(\mathcal{A})\} \,. \end{array}$$

The rank $\mathrm{rank}(p,w)$ of a configuration (p,w) in W is the smallest i such that $(p,w) \in \mathrm{Pre}_G^i(\mathcal{A})$. Intuitively the rank captures the distance for Elvis to a final configuration.

▶ **Definition 2.3** (Optimal Strategies)**.** A strategy for Elvis is optimal if, for plays ending in (p,w) in the winning region of Elvis, with $p \in \mathcal{Q}_E$, it prescribes a move to (p',w') that minimises $\mathrm{rank}(p',w')$ amongst all possible moves.

Optimal strategies are positional and winning from all configurations in W.

3 The Saturation Algorithm

In [1], Bouajjani et al. present an algorithm that given a pushdown reachability game $(\mathcal{Q}, \Gamma, \Delta, \mathcal{A})$ with an automaton $\mathcal{A} = (\mathbb{S}, \Gamma, \mathcal{F}, \delta)$, constructs a new automaton $\mathcal{B}$ accepting $\mathrm{Pre}_G^*(\mathcal{A})$. The requirements[3] on $\mathcal{A}$ are that no transition in δ goes back to a state in $\mathcal{Q}$. This is required to ensure that the invariants maintained by the algorithm hold initially.

The algorithm proceeds by adding transitions to $\mathcal{A}$ until no new transition can be added. The resulting automaton $\mathcal{B}$ accepts $\mathrm{Pre}_G^*(\mathcal{A})$. That is $w \in \mathcal{L}_q(\mathcal{B})$ iff $(q,w) \in \mathrm{Pre}_G^*(\mathcal{A})$.

The intuition of the algorithm can be seen as follows. Suppose q is a state of Elvis, there is a rule $(q,A) \to (p,w) \in \Delta$, and the configuration (p, ww') is accepted by the automaton by a run beginning with $p \overset{w}{\Rightarrow} S$. The configuration (q, Aw') should also be accepted since an application of the rule reaches (p, ww') from which a target configuration can be reached. Thus, a transition $q \xrightarrow{A} S$ is added, meaning (q, Aw') can now be accepted using the run we know exists over w' from S. For a move of the Anarchist we use alternation to gather together runs from all possible next configurations.

The algorithm constructs a finite sequence $(\mathcal{A}_i)_{i \in [0,N]}$ of automata. The automaton $\mathcal{A}_0$ is $\mathcal{A}$. Each $\mathcal{A}_i$ is of the form $(\mathbb{S}, \mathcal{F}, \delta_i)$, meaning that they only differ by their set of transitions. The construction guarantees that for all $i \in [0, N-1]$ we have $\delta_i \subseteq \delta_{i+1}$. It terminates when $\delta_{i+1} = \delta_i$. This occurs since the set of possible transitions is finite.

The set δ_{i+1} is obtained from δ_i as the smallest set of transitions such that

[3] This requirement is easily met by adding a copy of each state in $\mathcal{Q}$ if necessary.

1. $\delta_i \subseteq \delta_{i+1}$, and
2. for each $q \in \mathcal{Q}_E$, if $(q, A) \to (p, w) \in \Delta$ and $p \overset{w}{\underset{\mathcal{A}_i}{\Rightarrow}} S$, then $q \xrightarrow{A} S \in \delta_{i+1}$, and
3. for each $q \in \mathcal{Q}_A$ and $A \in \Gamma$ let

$$(q, A) \to (p_1, u_1), \ldots, (q, A) \to (p_n, u_n)$$

be all rules from (q, A) in Δ. For all sets $S_1, \ldots, S_n$ of states such that:

$$p_1 \overset{u_1}{\underset{\mathcal{A}_i}{\Rightarrow}} S_1, \ldots, p_n \overset{u_n}{\underset{\mathcal{A}_i}{\Rightarrow}} S_n \quad \text{we have} \quad q \xrightarrow{A} \bigcup_j S_j \in \delta_{i+1} \ .$$

One can prove that $(p, w) \in \mathrm{Pre}^*_G(\mathcal{A})$ iff $w \in \mathcal{L}_p(\mathcal{B})$ to obtain regularity of the winning region. Since an alternating automaton has at most exponentially many transitions in the number of states (and we do not add any new states), we have that $\mathcal{B}$ is constructible in EXPTIME.

4 Cachat's Algorithm

In Section 4.1, we describe Cachat's min-rank algorithm [3, 4]. This algorithm constructs a weighted alternating automaton which is used to associate to every accepted configuration a weight. We will see in Section 4.2 that this weight is an upper-bound on the rank of the configuration. In Section 4.3, we will show that contrary to Cachat's claim, it is not equal to the rank of the configuration and hence the associated strategy is not an optimal strategy.

4.1 Saturation Algorithm with Weights

Cachat's algorithm proceeds by annotating new transitions of the saturated automaton with two pieces of information: a weight in $\mathbb{N}$ and a rule of the pushdown system. Intuitively if a transition $q \xrightarrow{A} S$ is introduced by the saturation algorithm this means that, for every configuration (q, Aw), Elvis has a strategy to win without ever popping the A or to ensure that the A is popped and that the resulting state belongs to S. The weight of the transition is meant to capture the length of the longest play under an optimal strategy for Elvis. We will see that it is only an upper-bound on this length. The rule in the annotation is the one responsible for the introduction of the transition in the saturation algorithm.

Before presenting the algorithm, we need to define the weight of a run and of a set of runs when the transitions of the automaton are given weights by a function $\mathcal{W}$. The weight of a run ρ, denoted by $\mathcal{W}^*(\rho)$ is the maximum weight of a branch in the run where the weight of a branch is simply the sum of the weights of the transitions appearing in this branch. By convention, a run of depth 0 has weight 0.

For a word $w \in \Sigma^*$, a state s and a set of states S such that $s \overset{w}{\underset{\mathcal{A}}{\Rightarrow}} S$, we take:

$$\mathcal{W}^*(s \overset{w}{\underset{\mathcal{A}}{\Rightarrow}} S) = \min \left\{ \mathcal{W}^*(\rho) \mid \rho \in \left[s \overset{w}{\underset{\mathcal{A}}{\Rightarrow}} S \right] \right\}$$

The saturation function is updated to assign weights to new transitions based on the maximum weights of the runs it is based on. Formally a function α is defined which associates to each transition of the saturated automaton a tuple consisting of a weight and a rule of the pushdown system. For convenience, we use $\#$ to indicate the absence of an associated rule (for transitions of the initial automaton and moves of the Anarchist). Moreover, we denote by $\mathcal{W}$ the projection of α on the weight component.

Given some initial reachability target set represented by an automaton $\mathcal{A}_0 = \mathcal{A} = (\mathbb{S}, \Gamma, \delta_0, \mathcal{F})$, we initially define $\alpha(\tau) = (0, \#)$ for all $\tau \in \delta_0$.

The annotation function α is updated as new transitions are added. At each iteration, we define $\mathcal{A}_{i+1} = (\mathbb{S}, \Gamma, \delta_{i+1}, \mathcal{F})$ where δ_{i+1} is the smallest set of transitions such that

1. $\delta_i \subseteq \delta_{i+1}$, and
2. for each $q \in \mathcal{Q}_E$, if $r = (q, A) \to (p, w) \in \Delta$ and $p \overset{w}{\underset{\mathcal{A}_i}{\Longrightarrow}} S$ then $\tau = q \xrightarrow{A} S \in \delta_{i+1}$ and furthermore, we assign

$$\alpha(\tau) = \left(1 + \mathcal{W}^*\left(p \overset{w}{\underset{\mathcal{A}_i}{\Longrightarrow}} S\right), r\right) .$$

3. for each $q \in \mathcal{Q}_A$ and $A \in \Gamma$ let $(q, A) \to (p_1, u_1), \ldots, (q, A) \to (p_m, u_m)$ be all rules from (q, A) in Δ. For each set of runs $p_1 \overset{u_1}{\underset{\mathcal{A}_i}{\Longrightarrow}} S_1, \ldots, p_n \overset{u_m}{\underset{\mathcal{A}_i}{\Longrightarrow}} S_m$ we have $\tau = q \xrightarrow{A} \bigcup_j S_j \in \delta_{i+1}$ and furthermore, we assign

$$\alpha(\tau) = \left(1 + \max_{1 \leq j \leq m}\left(\mathcal{W}^*\left(p_j \overset{u_j}{\underset{\mathcal{A}_i}{\Longrightarrow}} S_j\right)\right), \#\right)$$

Cachat writes that the algorithm terminates when "no new transitions can be added". The formulation of the algorithm seems to indicate that the weights are final and that the algorithm terminates when all transitions have been added. It is possible to construct an example in which the weight of a transition would decrease from its initial value. It would be easy to adapt the algorithm to allow the weight of the transitions to decrease after their initial introduction but this would not fix the deeper problem pointed out in Section 4.3. In the case where there is only one player (*i.e.* all states belong to Elvis), it is possible to ensure the weight of transitions are final by adding transitions one by one: at each round the transition with the smallest possible weight is created [13, p. 63]. In the following, we will consider that the algorithms stops when the transition structure is stable (*i.e.* $\delta_{i+1} = \delta_i$) and that a transition τ is added to δ_{i+1} only if does not belong to δ_i.

4.2 Min-Rank Strategy

We now assume that $\mathcal{B}$ is the saturated automaton produced by the previous algorithm and that α is the corresponding annotation function. Recall $\mathcal{W}$ is the projection of α on the weight component.

First we remark that as the saturated automaton $\mathcal{B}$ is identical to the one obtained in the original saturation algorithm, $\mathcal{B}$ accepts the winning region of Elvis. Hence the weight of a configuration (q, w) in the winning region can be defined as the minimal weight for an accepting run for this configuration:

$$\mathcal{W}^*(q, w) = \min\left(\mathcal{W}^*\left(q \overset{w}{\underset{\mathcal{B}}{\Longrightarrow}} S\right) \mid S \subseteq \mathcal{F} \text{ and } q \overset{w}{\underset{\mathcal{B}}{\Longrightarrow}} S\right) .$$

Cachat defines what he calls the min-rank positional strategy for Elvis. In this strategy Elvis plays, at a configuration $(q, w) \in W$ (which he owns), the move corresponding to the rule annotating the first transition in any accepting run of $\mathcal{B}$ on w starting with p of minimal weight $\mathcal{W}^*(q, w)$. If the rule annotating the top-most transition is undefined then the configuration is final and no moves needs to be played.

As stated by Cachat, the move outputted by the strategy can be computed in time linear in the length of the input configuration and exponential in the size of the pushdown game (with an exponential precomputation in the size of the game). The algorithm consists in reading w from right to left while maintaining for each state q the weight of the minimal run

accepting the stack content from q. This information can be updated in $O(|\mathcal{B}|)$ upon reading a new stack symbol.

From the definition of the algorithm, it can be shown that for all configurations $(q, w) \in W$:

- if (q, w) is owned by the Anarchist, then for all configurations (q', w') with $(q, w) \to (q', w')$, we have $\mathcal{W}^*(q, w) > \mathcal{W}^*(q', w')$.
- if (q, w) is owned by Elvis, then for a configuration (q', w') prescribed by the min-rank positional strategy (if it exists) then $\mathcal{W}^*(q, w) > \mathcal{W}^*(q', w')$.

These properties allow the following properties of the min-rank strategy to be proved.

▶ **Theorem 4.1** ([3]). *The min-rank strategy is positional and winning from all configurations in Elvis's winning region.*

In [3], Cachat in addition claims that the min-rank strategy is optimal, which is not the case. The mistake lies in [3, Proposition 6] where Cachat's claims in that the weight of a configuration $(q, w) \in W$ is equal to the rank of this configuration. However, this turns out not to be true as we will see in the next section. Cachat's proof [4, p. 34] in fact shows that $\mathcal{W}^*(p, w)$ is an upper bound on $\text{rank}(p, w)$, but not the converse inequality.

▶ **Proposition 4.2** ([3]). *For any configuration $(q, w) \in \text{Pre}^*_G(\mathcal{A})$ we have $\text{rank}(q, w) \leq \mathcal{W}^*(q, w)$.*

From this, we can obtain the following corollary which we will need in the sequel.

▶ **Corollary 4.3** (Upper Bound). *Take a pushdown game $G = (\mathcal{Q}, \Gamma, \Delta, \mathcal{A})$ with an alternating automaton $\mathcal{A} = (\mathbb{S}, \Gamma, \mathcal{F}, \delta)$ and let $\mathcal{B}$ be the result of Cachat's saturation algorithm. For $C = 2^{|\mathcal{Q}|\cdot|\Gamma|\cdot 2^{|\mathbb{S}|}}$, we have:*

- *for all transitions $q \xrightarrow{A} S$ in $\mathcal{B}$, its weight is bounded by C, and*
- *for any configuration $(p, w) \in \text{Pre}^*_G(\mathcal{A})$ we have $\text{rank}(p, w) \leq C \cdot |w|$.*

Proof. At each iteration of the saturation, the weight of a new transition is at most $1 + 2k$ where k is the maximum weight appearing on a transition in the previous iteration. A direct induction shows that for all $i \geq 0$, the maximum weight of a transition in δ_i is at most $2^i - 1$. As there are at most as many iterations as there are possible transitions of the saturated automaton, after at most $|\mathcal{Q}| \cdot |\Gamma| \cdot 2^{|\mathbb{S}|}$ iterations, no new transitions will be added. It follows that the weight of a transition in the saturated automaton is at most $2^{|\mathcal{Q}|\cdot|\Gamma|\cdot 2^{|\mathbb{S}|}}$ which is the announced constant.

Now consider a configuration $(q, w) \in \text{Pre}^*_G(\mathcal{A})$. From Proposition 4.2, we know that $\text{rank}(q, w)$ is bounded by the weight of any accepting run of $\mathcal{B}$ on w starting from q. The cost of such a run is a most $C \cdot |w|$ which concludes. ◀

4.3 Non-optimality of the Min-Rank Strategy

We give a counter-example in which the weight of configurations are strictly greater than their rank. Then we adapt this counter-example into a game where the min-rank strategy is not optimal. The goal of our counter example is to introduce a transition $q \xrightarrow{A} \{r, s\}$ corresponding to a situation in the game where

- the cost of a play to r is low, but the play from r to a target configuration is long, and
- the cost of a play to s is high, but the play from s to a target configuration is short.

Our game has one control state p belonging to the Anarchist and all other states q, q', r, r', s, and f belong to Elvis. The goal is to reach a configuration $(f, \bot)$. We give the moves below,

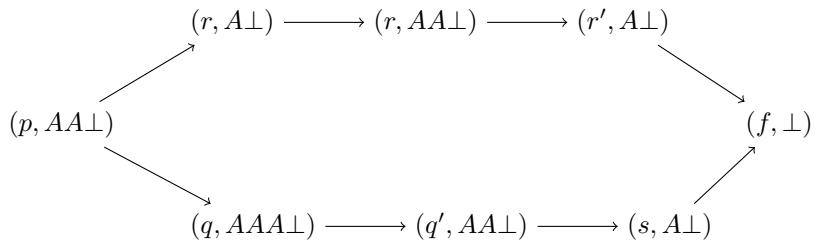


Figure 1 A game showing a counter example to the algorithm of Cachat.

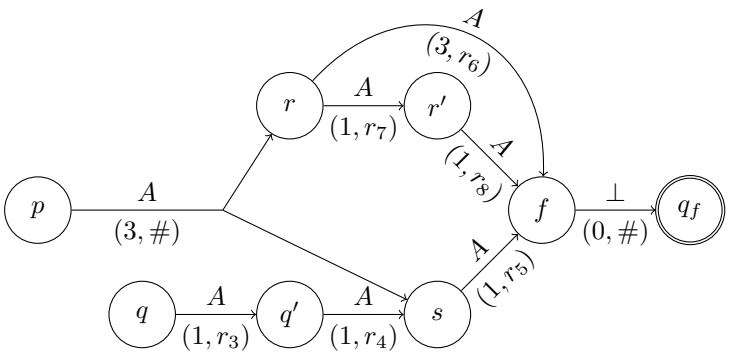


Figure 2 The saturated automaton.

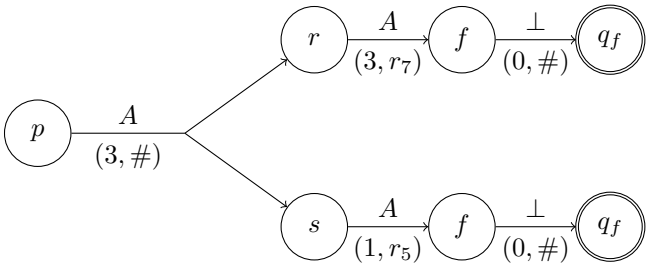


Figure 3 The accepting run of $(p, aa\bot)$.

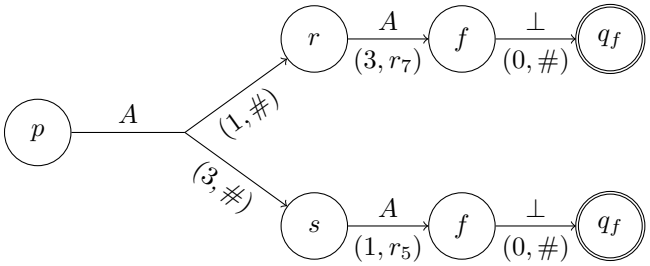


Figure 4 A run of $(p, aa\bot)$ with fine-grained weights.

and show in Figure 1 the game graph from $(p, AA\bot)$. No matter how the Anarchist plays, it will take four steps to reach $(f, \bot)$.

The moves available to the Anarchist are

$$r_1 = (p, A) \to (r, \varepsilon) \qquad \text{and} \qquad r_2 = (p, A) \to (q, AA)$$

and the remaining moves are available to Elvis and consist of

$$\begin{array}{lll} r_3 = (q, A) \to (q', \varepsilon) & r_4 = (q', A) \to (s, \varepsilon) & r_5 = (s, A) \to (f, \varepsilon) \\ r_6 = (r, A) \to (r, AA) & r_7 = (r, A) \to (r', \varepsilon) & r_8 = (r', A) \to (f, \varepsilon) \ . \end{array}$$

We start with $\mathcal{A}_0$ containing only the transition $f \xrightarrow[(0,\#)]{\bot} q_f$ where q_f is the only accepting state, and $(0, \#)$ is the annotation.

We then begin saturation. The pop rules r_3, r_4, r_5, r_7, and r_8 from states belonging to Elvis immediately lead to the introduction of new transitions. These can be seen in Figure 2 where the annotations show which rule lead to each new transition. The transitions from p and r are described below.

First, we can deal with the push at control state r using r_6. This leads to the introduction of a transition $r \xrightarrow[(3,r_6)]{A} f$ because of the run $r \xrightarrow[(1,r_7)]{A} r' \xrightarrow[(1,r_8)]{A} f$. Next, both rules from p need to be considered simultaneously. That is we introduce a transition $p \xrightarrow[(3,\#)]{A} \{r, s\}$ because of the rule r_1 which is a pop rule from p to r and r_2 and the run $q \xrightarrow[(1,r_3)]{A} q' \xrightarrow[(1,r_4)]{A} s$. No more transitions can be added. The result is shown in Figure 2. The alternating transition is shown with a split arrow.

Given this automaton, we consider the accepting run of $(p, AA\bot)$ shown in Figure 3. Note that the weight of this run is 6, but the longest run that the Anarchist can enforce is 4.

One can extend this to a full counter example to the optimality of Cachat's algorithm as follows. From an initial configuration $(p_0, AAA\bot)$ we give Elvis the choice of moving to $(p, AA\bot)$ via a pop, or to another configuration $(p_1, AA\bot)$ from which 5 steps are required to reach $(f, \bot)$. Since the rank of $(p, AA\bot)$ is estimated to be 6 rather than 4, the strategy will choose to move to $(p_1, AA\bot)$ rather than $(p, AA\bot)$, leading to a play that is not optimal.

To be more explicit, in the full counter example, the game has one control state p belonging to the Anarchist and all other states q, q', r, r', s, and f as well as p_0, p_1, p_2, p_3, p_4, and p_5 belong to Elvis. The goal is to reach a configuration $(f, \bot)$. The moves available to the Anarchist are as before

$$r_1 = (p, A) \to (r, \varepsilon) \qquad \text{and} \qquad r_2 = (p, A) \to (q, AA)$$

and the remaining moves are available to Elvis and consist of the previous rules

$$\begin{array}{lll} r_3 = (q, A) \to (q', \varepsilon) & r_4 = (q', A) \to (s, \varepsilon) & r_5 = (s, A) \to (f, \varepsilon) \\ r_6 = (r, A) \to (r, AA) & r_7 = (r, A) \to (r', \varepsilon) & r_8 = (r', A) \to (f, \varepsilon) \ . \end{array}$$

as well as

$$\begin{array}{lll} r_9 = (p_0, A) \to (p, \varepsilon) & r_{10} = (p_0, A) \to (p_1, \varepsilon) & r_{11} = (p_1, A) \to (p_2, A) \\ r_{12} = (p_2, A) \to (p_3, A) & r_{13} = (p_3, A) \to (p_4, A) & r_{14} = (p_4, A) \to (p_5, \varepsilon) \\ r_{15} = (p_5, A) \to (f, \varepsilon) \ . & & \end{array}$$

At $(p_0, AAA\bot)$ Elvis has two possible moves. The first is to $(p, AA\bot)$ which, as shown above, has a calculated rank of 6. The second is to $(p_1, AA\bot)$ which one can easily verify has a rank of 5. Thus, Cachat's strategy is to move to $(p_1, AA\bot)$ which is not optimal.

This overapproximation occurs because the information stored on the transition from p is too coarse. The weight of 3 comes from the weight of the run to s. The weight of the run to r is 1. Thus, if we were to store a weight for each control state we would obtain two weights on the transition from p. This would allow us to identify that the true cost of the run is 4. This can be seen in Figure 4 and is the basis of our corrected algorithm.

5 Computing Optimal Strategies

We refine Cachat's algorithm to compute optimal strategies in pushdown reachability games. The key idea is to replace simple annotations $\alpha\left(s \xrightarrow{A} S\right) = (d, r)$ with a more fine-grained version, which assigns a separate weight to each state in S. For this, we include annotations in the definition of a transition. That is, a transition is a tuple (s, A, S, D, r) where s is a state in $\mathbb{S}$, A is a character in Γ, $S \subseteq \mathbb{S}$, $D : S \mapsto \mathbb{N}$ is a weight function, and r is a rule of G. We write such transitions as $s \xrightarrow[D,r]{A} S$. As before, we calculate the weights of runs.

5.1 Profile of a run

In an automaton run ρ with annotated transitions, we define the weight of a branch inductively

$$\mathcal{W}^*(\eta_0) = 0 \quad \text{and} \quad \mathcal{W}^*(\eta_0 \cdots \eta_\ell) = D(s_1) + \mathcal{W}^*(\eta_1 \cdots \eta_\ell)$$

where D is the weight function the transition labeling (η_0, η_1) and s_1 is the state labeling η_1. We define a run profile.

▶ **Definition 5.1** (Profiles). Given a run ρ over a word w, the *profile* of ρ is given by $\mathbb{P}(\rho) = (S, D)$ where S is the set of states in $\mathbb{S}$ labelling leaves of ρ and $D : S \to \mathbb{N}$ is the function such that, for all $s \in S$, we have that $D(s)$ is the maximum weight of a branch from the root node to a leaf labelled s. Moreover, we define

$$(1 + D)(s) = 1 + D(s) \quad \text{and} \quad \max(D_1, \ldots, D_m)(s) = \max_{1 \le j \le m} (D_j(s)) \ .$$

Finally, given (S, D) and (S, D'), we write $D \le D'$ when for all $s \in S$ we have $D(s) \le D(s')$. By Dickson's Lemma, $\le$ is a well-quasi-ordering on the weights.

5.2 Saturation

We use the saturation algorithm with run profiles rather than Cachat's annotations. At each iteration, we set $\mathcal{A}_{i+1} = (\mathbb{S}, \Gamma, \delta_{i+1}, \mathcal{F})$ where δ_{i+1} is the smallest set of transitions with

1. $\delta_i \subseteq \delta_{i+1}$, and
2. for each $q \in \mathcal{Q}_E$, if $r = (q, A) \to (p, w) \in \Delta$ and ρ is a run of $\mathcal{A}_i$ over w from p with profile $\mathbb{P}(\rho) = (S, D)$ then

$$\tau = q \xrightarrow[1+D,r]{A} S \in \delta_{i+1} \ .$$

3. for each $q \in \mathcal{Q}_A$ and $A \in \Gamma$ let $(q, A) \to (p_1, u_1), \ldots, (q, A) \to (p_m, u_m)$ be all rules from (q, A) in Δ. For each set $\rho_1, \ldots, \rho_m$ of runs of $\mathcal{A}_i$ such that for each $1 \le j \le m$ the run ρ_j is a run over u_j with root note labelled s_j and profile (S_j, D_j), we have

$$\tau = q \xrightarrow[D',\#]{A} \bigcup_j S_j \in \delta_{i+1}$$

where $D' = 1 + \max(D_1, \ldots, D_m)$.

The algorithm terminates at the first i such that for all $s \xrightarrow[D,r]{A} S \in \delta_{i+1}$ there exists $s \xrightarrow[D',r']{A} S \in \delta_i$ with $D' \leq D$. Equivalently, the algorithm terminates when the set of transitions with a minimal weights stabilizes.

▶ **Lemma 5.2.** *The saturation algorithm terminates and computes* $\mathrm{Pre}^*_G(\mathcal{A})$.

Proof. We have $\mathrm{Pre}^*_G(\mathcal{A})$ from saturation without weights. We terminate as the set of possible transitions (without weight) is finite and $\leq$ is a well-quasi-ordering on the weights. ◀

5.3 Defined Strategy

We define our optimal strategy σ_O as follows. Let $\mathcal{A}'$ be the saturated automaton and let $\mathrm{AccRuns}(q,w)$ be the set of accepting runs of $\mathcal{A}'$ over w from q. We define

$$\mathcal{W}^*(q,w) = \min_{\rho \in \mathrm{AccRuns}(q,w)} \left(\max_{\substack{\mathbb{P}(\rho)=(S,D),\\ s\in S}} (D(s)) \right).$$

From each configuration (q,w) in the winning region of Elvis, if it is the move of Elvis, he plays the move which leads to the configuration (q',w') that minimises the value of $\mathcal{W}^*(q',w')$. In particular, let $q \xrightarrow[D,r]{A} S'$ be the first transition on a run with weight $\mathcal{W}^*(q,w)$. If it is Elvis's move, he should play the rule r. If $r = \#$, then either Elvis has already reached a target configuration or it is the Anarchist's move. Note, this strategy is non-deterministic since there may be multiple choices of minimal run. In order to define a strategy that is a function, we can fix an ordering on the moves of the game, and always choose the smallest. Note, moreover, that σ_O is positional.

▶ **Lemma 5.3.** *The strategy* σ_O *is an optimal winning strategy.*

6 Optimal Computation of Optimal Strategies

We show how to reduce the complexity to 2EXPTIME as well as give a lower bound example showing that a doubly exponential number of moves is optimal. For this we will first show that we can bound the value of the weights appearing in *minimal* transitions by a constant K which is doubly exponential in the size of the pushdown game. Then we will restrict the saturation to only consider transitions with weights at most K. Finally we will give a lower bound showing that the constant K needs to be doubly exponential.

6.1 Bounding Play Lengths

We show that there exists a constant K such that any point-wise minimal transitions has all its weights below K. For this we need a stronger result below.

▶ **Lemma 6.1.** *Let* $K = 2^{|\mathcal{Q}|\cdot|\mathbb{S}|\cdot|\Gamma|\cdot 2^{|\mathbb{S}|}+|\mathbb{S}|}$. *For all transitions* $q \xrightarrow[D,r]{A} S$ *such that for some* $s \in S$ *we have* $D(s) > K$ *then there exists a transition* $q \xrightarrow[D',r']{A} S$ *such that* $D' \leq D$ *and* $D'(s) \leq K$ *for all* $s \in S$.

The proof of Lemma 6.1 is non-trivial. To give the idea of the proof, we consider the case of a pushdown game whose target is the empty stack. Intuitively we proceed as follows.

Let C be the bound obtained in Corollary 4.3. Fix a transition $q \xrightarrow[D,r]{A} S$ of the saturated automaton. We consider the *reduced game*, which broadly corresponds to the game starting with (q, A) and where Elvis aims to empty the stack whilst reaching one of the control states in S. Let σ be a strategy for Elvis associated with the transition in the sense that it ensures that any play following σ and ending in $s \in S$ has length at most $D(s)$.

The simplest case is when $D(s) \geq C$ for all $s \in S$. Then we can find a transition by Cachat that improves on all points. Otherwise, there is at least one $D(s) < C$. We make a new strategy which plays according to σ for $D(s)$ moves. If we have not reached s in this time, we know that playing by σ will never reach s. In particular, there exists a strategy to reach $S \setminus \{s\}$. Moreover, we have increased the stack height by at most C. Thus, we know from Cachat that we can empty the stack and reach any state in $S \setminus \{s\}$ in $C \cdot C$ moves (that is at most C moves per stack character that needs to be removed). We repeat the above argument but this time remove some state s' with $D(s') < C^2$. We play until we are sure not to reach s', increasing the stack height by at most C^2. This means we can reach $S \setminus \{s, s'\}$ in $C \cdot (C + C^2)$ moves and so on. The existence of the strategy in turns implies the existence of a transition in the saturated automaton. In this way, we obtain the bound $K = 2^{|\mathbb{S}|} \cdot C^{|\mathbb{S}|}$.

▶ Remark. We know from Corollary 4.3 (Upper Bound) that there is a doubly exponential bound C on the weight of individual transitions. It is therefore tempting to consider this bound as a proof of the sufficiency of the saturation algorithm with shortcuts described in Section 6.2. However, the bound obtained from Cachat does not guarantee *a priori* that for every transition outside of the bound, there is a transition within the bound that is a pointwise improvement. For example, it is conceivable that Elvis may have two strategies for reaching either the state q_1 or q_2 from q whilst removing A from the stack. The first, corresponding to Cachat's bound, may give a play length of 2 whether the Anarchist forces play to q_1 or q_2. The second, which may violate Cachat's bound, may give a play length of 1 if the Anarchist forces play to q_1, but an extremely large play length (violating Cachat's bound) if the Anarchist forces play to q_2. Let's say this large play length is 100.

Now, consider a configuration (q, Aw). Suppose (q_1, w) requires 100 steps to reach a target configuration, and (q_2, w) is a target configuration. The strategy corresponding to the within-bounds transition has rank 102, whilst the strategy corresponding to the out-of-bounds transition has rank 101. Thus, we have not dismissed the need for out-of-bounds transitions.

6.2 Shortcutting Saturation

We adjust the saturation algorithm by insisting that a transition $s \xrightarrow[D,r]{A} S$ only appears in δ_{i+1} if for all $s \in S$ we have $D(s) \leq K$. Lemma 6.1 guarantees that the set of point-wise minimal transitions is not affected by this restriction.

With this restriction, the worst case running time of saturation becomes doubly exponential. The number of possible weight functions is $K^{|\mathbb{S}|}$ and hence the number of possible transitions is doubly exponential. Since at least one transition must be added during each step of the saturation, the algorithm must terminate in 2EXPTIME.

Like Cachat using a backward algorithm, the move for this strategy on (q, w) can be computed in time linear in the length of w and doubly exponential in the size of the pushdown game (assuming that the saturated automaton has been computed). We read w from right to left maintaining for each state s the value $\mathcal{W}^*(s, u)$ (if it exists) where u is the word read so far. This information can be updated in $\mathcal{O}(|\mathcal{B}|)$ upon reading a new letter and allows the first transition of an accepting run of minimal weight to be found.

6.3 Lower Bound

We give an example game where the shortest run to the target set is doubly exponential, hence showing that the bound K needs to be doubly exponential. The intuition is simple. First, suppose we wanted to force an exponential-length run. In this case we could store a binary number of n digits on the stack, with the least significant bit at the top. E.g. the number 3 would be encoded in a configuration $(q_0, 11000)$ when $n = 5$. To increment the number, we pop 1 characters from the stack until we find the first 0. We record in the control state how many pops are needed. In this case we need two pops and reach $(q_2, 000)$. Then, we replace the topmost 0 with a 1 and push 0s onto the stack until the height is n again. In our example, we reach $(q_0, 00100)$. The goal is to reach a stack with only 1 character, from a stack with only 0. This requires 2^n steps and can be done even in a single-player game.

To generate a doubly exponential run, we follow the same outline, but require the binary encoding to be exponentially long. We cannot use only the control states to enforce this length since it would require an exponential number of them. However, when rebuilding the stack during the increment, we can build a game which forces Elvis to construct a stack of at least exponential height. To do this, after changing the first 0 to 1 Elvis must push any number of 0 characters. Once he is done the Anarchist may accept that the stack is large enough, or challenge the height. To challenge the height we use the fact that the least common divisor of the first n prime numbers is exponential in n. Hence, the Anarchist can pick any of the first n primes, say p, and start a subgame with control states $q_0, \ldots, q_{p-1}$. From each of these control states q_i the only move is a pop to $q_{(i+1 \bmod p)}$. This sub-game is won only if q_0 is reached when the bottom of the stack is reached. Consequently, Elvis must have built the stack up to a multiple of the least common divisor of the first n primes. Note, Elvis may build a stack that is taller than the least common divisor, but this only makes reaching the target state harder.

7 Conclusion

We have studied optimal strategy construction for pushdown reachability games. Initial results due to Cachat [3] unfortunately were too coarse in their analysis and the claimed optimality is in fact an over-approximation. We showed that a refinement of Cachat's algorithm can be made to compute the optimal strategy accurately; however, the additional information required makes it difficult to obtain good complexity results. We gave a non-trivial argument that the algorithm can be refined further to obtain a 2EXPTIME algorithm. Moreover, the doubly exponential weights computed by the algorithm are optimal as demonstrated by a game where the winning strategy requires a doubly exponential number of moves to reach a target configuration.

References

1. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, pages 135–150, 1997.
2. R. J Büchi. Regular canonical systems. *Archive for Mathematical Logic*, 6(3):91–111, 1964. doi:10.1007/BF01969548.
3. T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *ICALP*, pages 704–715, 2002. doi:10.1007/3-540-45465-9_60.
4. T. Cachat. *Games on Pushdown Graphs and Extensions*. PhD thesis, RWTH Aachen, 2003. URL: http://darwin.bth.rwth-aachen.de/opus3/volltexte/2004/957/pdf/Cachat_Thierry.pdf.

5 Arnaud Carayol and Matthew Hague. Regular strategies in pushdown reachability games. In *Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings*, pages 58–71, 2014. doi:10.1007/978-3-319-11439-2_5.

6 D. Caucal. Récritures suffixes de mots. Research Report RR-0871, INRIA, 1988.

7 A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *INFINITY*, volume 9, pages 27–37, 1997.

8 Wladimir Fridman. Formats of winning strategies for six types of pushdown games. In *Proceedings First Symposium on Games, Automata, Logic, and Formal Verification, GANDALF 2010, Minori (Amalfi Coast), Italy, 17-18th June 2010.*, pages 132–145, 2010. doi:10.4204/EPTCS.25.14.

9 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.

10 N. D. Jones and S. S. Muchnick. Even simple programs are hard to analyze. *J. ACM*, 24:338–350, April 1977. doi:10.1145/322003.322016.

11 O. Kupferman, N. Piterman, and M. Y. Vardi. An automata-theoretic approach to infinite-state systems. In *Essays in Memory of Amir Pnueli*, pages 202–259, 2010. doi:10.1007/978-3-642-13754-9_11.

12 N. Piterman and M. Y. Vardi. Global model-checking of infinite-state systems. In *CAV*, pages 387–400, 2004.

13 S. Schwoon. *Model-checking Pushdown Systems*. PhD thesis, Technical University of Munich, 2002.

14 O. Serre. *Contribution à l'étude des jeux sur des graphes de processus à pile*. PhD thesis, Université Paris 7 – Denis Diderot, UFR d'informatique, 2004. URL: http://tel.archives-ouvertes.fr/tel-00011326.

15 I. Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001. doi:10.1006/inco.2000.2894.

16 W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.

Why are CSPs Based on Partition Schemes Computationally Hard?

Peter Jonsson
Department of Computer and Information Science, Linköping University, Linköping, Sweden
peter.jonsson@liu.se

Victor Lagerkvist
Department of Computer and Information Science, Linköping University, Linköping, Sweden
victor.lagerkvist@liu.se

Abstract

Many computational problems arising in, for instance, artificial intelligence can be realized as infinite-domain constraint satisfaction problems (CSPs) based on *partition schemes*: a set of pairwise disjoint binary relations (containing the equality relation) whose union spans the underlying domain and which is closed under converse. We first consider partition schemes that contain a strict partial order and where the constraint language contains all unions of the basic relations; such CSPs are frequently occurring in e.g. temporal and spatial reasoning. We identify three properties of such orders which, when combined, are sufficient to establish NP-hardness of the CSP. This result explains, in a uniform way, many existing hardness results from the literature. More importantly, this result enables us to prove that CSPs of this kind are not solvable in subexponential time unless the exponential-time hypothesis (ETH) fails. We continue by studying constraint languages based on partition schemes but where relations are built using disjunctions instead of unions; such CSPs appear naturally when analysing first-order definable constraint languages. We prove that such CSPs are NP-hard even in very restricted settings and that they are not solvable in subexponential time under the randomised ETH. In certain cases, we can additionally show that they cannot be solved in $O(c^n)$ time for any $c \geq 0$.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases Constraint satisfaction problems, infinite domains, partition schemes, lower bounds

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.43

Acknowledgements We thank the anonymous reviewers for several helpful comments. The authors are partially supported by the Swedish Research Council (VR) under grant 2017-04112. In addition, the second author has received funding from the DFG-funded project "Homogene Strukturen, Bedingungserfüllungsprobleme, und topologische Klone" (Project number 622397), and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681988, CSP-Infinity).

1 Introduction

The *constraint satisfaction problem* over a constraint language Γ (CSP(Γ)) is the decision problem of verifying whether a set of constraints based on the relations in Γ admits a satisfying assignment. For finite domains the complexity of CSP(Γ) is well understood due to the recent dichotomy theorem separating tractable from NP-complete problems [6, 30], but for infinite domains the situation differs markedly. This class of problems includes both undecidable problems and NP-intermediate problems, and it is therefore common to impose

additional assumptions on the allowed constraints. The predominant method has been to fix a constraint language Γ, usually satisfying certain model-theoretic properties, and analyse the complexity of CSPs over first-order reducts of Γ. Traditionally, this has also been the case for CSPs arising from artificial intelligence, e.g. temporal and spatial reasoning problems, albeit usually with weaker closure conditions.

Motivated by problems of this form, we study the complexity of infinite-domain CSPs over *partition schemes*. A partition scheme [20] is a set of pairwise disjoint binary relations $\mathcal{B}$ over a domain D such that $\bigcup_{R\in\mathcal{B}} R = D^2$ and which for every relation contains its converse. Partition schemes are the de facto standard for CSPs in the artificial intelligence community [8], due to their capability of modelling many different kinds of reasoning problems. Given a partition scheme, the predominant way of forming new relations is to allow unions of the relations in $\mathcal{B}$, and we let $\mathcal{B}^{\vee=}$ denote this set. We will also study languages where each relation can be defined as a disjunction of constraints from $\mathcal{B}$ of arity at most $k \geq 1$, and let $\mathcal{B}^{\vee k}$ denote this set. Note that $\mathcal{B}^{\vee=} \subseteq \mathcal{B}^{\vee k}$ for sufficiently large k but that $\mathcal{B}^{\vee k} \subseteq \mathcal{B}^{\vee=}$ does not necessarily hold for any $k > 1$. Languages of the form $\mathcal{B}^{\vee k}$ occur naturally in theoretical CSP research since such classification projects typically aim to understand the complexity of all first-order reducts of a base set $\mathcal{B}$ of relations.

Famous AI examples of formalisms based on partition schemes include *Allen's interval algebra*, the *region connection calculus*, and the *rectangle algebra*. For more examples, see e.g. the survey by Dylla et al. [9]. $\mathrm{CSP}(\mathcal{B}^{\vee=})$ problems have been proven to be NP-hard for many choices of $\mathcal{B}$. The proofs have utilised various reductions from various problems, but there has not been a clear explanation why the majority of them are NP-hard. We will try to obtain such an explanation in the sequel. Our first step (in Section 3) is to note that the majority of practically relevant partition schemes contain strict partial orders satisfying certain properties, which we in this paper refer to as *infinite height*, *in-forks*, and *out-forks*. In Section 4 we prove that these properties are sufficient to guarantee that $\mathrm{CSP}(\mathcal{B}^{\vee=})$ is NP-hard. It might be interesting to observe that we do not need any strong model-theoretic properties, e.g. ω-categoricity, which is otherwise common for infinite-domain CSPs. This result is also interesting to compare to the procedure by Renz and Li [25] which takes a partition scheme as input and tries to prove NP-hardness. One important distinction is that our result provides a concrete source of NP-hardness while the algorithm in Renz and Li gives no such insight. Moreover, this procedure is not complete, and is due to computational constraints not applicable to e.g. the rectangle algebra, while it is a straightforward task to prove that this algebra falls within the scope of our result. Hence, our study offers a more theoretical explanation of why so many naturally occurring CSPs over partition schemes are computationally hard.

Having identified a natural class of NP-hard CSPs based on partition schemes, we turn, in Section 4.2 and Section 5, to the problem of showing lower bounds for problems of this form. Traditionally, it is fair to say that such investigations have largely been neglected by both the artificial intelligence community and the CSP community. There are a few reasons for this. First, significant efforts have been made to solve hard reasoning problems with efficient heuristics [24], which are typically difficult to analyse rigorously even if they work well for certain real-world instances. Second, existing lower bounds are typically based on size-preserving reductions from SAT-like problems where one needs the ability to express disjunctive clauses, which is difficult to express with partition schemes. In fact, to the best of our knowledge, the only concrete lower bounds for a CSP over a partition scheme is the bound by Jonsson and Lagerkvist [16] which relates the complexity of Allen's interval algebra to the complexity of the Chromatic Number problem. We show that a

size-preserving reduction from a SAT-like problem, perhaps contrary to intuition, is possible for certain CSPs over partition schemes, using ideas from Opatrny [23]. More precisely we prove that $\mathrm{CSP}(\mathcal{B}^{\vee=})$ cannot be solved in subexponential time unless the *exponential-time hypothesis* is false. One way of interpreting this result is that $\mathrm{CSP}(\mathcal{B}^{\vee=})$ is far from being polynomial-time solvable: there is a constant $c > 1$ such that the problem cannot be solved in $O(c^n)$ time. An immediate consquence of lower bounds of this form is that we can immediately rule out certain kinds of algorithms for $\mathrm{CSP}(\mathcal{B}^{\vee=})$, e.g. algorithms based on graph-decomposition and k-consistency, which typically run in subexponential or polynomial time. It is of course tempting to strengthen our lower bound even further since the current best known algorithm for $\mathrm{CSP}(\mathcal{B}^{\vee=})$ for an arbitrary partition scheme $\mathcal{B}$ runs in $2^{O(n^2)}$ time, if $\mathrm{CSP}(\mathcal{B})$ is polynomial-time solvable [16, 27]. While we do not succeed in doing this, we can provide stronger lower bounds for $\mathrm{CSP}(\mathcal{B}^{\vee k})$: we prove that $\mathrm{CSP}(\{\prec\}^{\vee 4})$, where $\prec$ is a strict partial order of infinite height, is not solvable in $O(c^n)$ time for any $c \geq 0$ assuming the complexity theoretical assumption known as the *randomised exponential-time hypothesis* (r-ETH). We also show that $\mathrm{CSP}(\mathcal{B}^{\vee 2})$ cannot be solved in subexponential time if we assume the r-ETH and that a non-empty relation $R \subseteq \{(x, y, z) \in D^3 \mid x \neq y, x \neq z, y \neq z\}$ can be defined in $\mathcal{B}^{\vee 2}$. Note that we do not require $\mathcal{B}$ to contain any partial orders in this case. We conclude the paper with some discussion in Section 6, where we point out some future research directions concerning both lower and upper bounds.

2 Preliminaries

In this section we introduce the necessary prerequisites concerning constraint satisfaction problem, disjunctive relations, and partition schemes. We begin by defining the CSP problem when it is parameterized by a set of relations.

▶ **Definition 1.** Let Γ be a set of finitary relations over some set D of values. The *constraint satisfaction problem* over Γ ($\mathrm{CSP}(\Gamma)$) is defined as follows:

Instance: A set V of variables and a set C of *constraints* of the form $R(v_1, \ldots, v_k)$, where k is the arity of R, $v_1, \ldots, v_k \in V$ and $R \in \Gamma$.
Question: Is there a function $f : V \to D$ such that $(f(v_1), \ldots, f(v_k)) \in R$ for every $R(v_1, \ldots, v_k) \in C$?

The set Γ is called a *constraint language*. Given an instance I of $\mathrm{CSP}(\Gamma)$ we write $||I||$ for the number of bits required to represent I. We will occasionally encounter *bounded-degree* CSP instances. Let (V, C) denote an instance of $\mathrm{CSP}(\Gamma)$. If a variable x occurs in B constraints in C, then we say that the degree of x is B. We let $\mathrm{CSP}(\Gamma)$-B denote the $\mathrm{CSP}(\Gamma)$ problem where each variable in the input is restricted to have degree at most B. Note that if (V, C) is a $\mathrm{CSP}(\Gamma)$-B instance, then $|C| \leq B \cdot |V|$, implying that the number of constraints is linearly bounded with respect to the number of variables.

We continue by describing how to use disjunctions for combining relations.

▶ **Definition 2.** Let D be a set of values and let $\mathcal{B} = \{B_1, \ldots, B_m\}$ denote a finite set of relations over D, i.e. $B_i \subseteq D^j$ for some $j \geq 1$.

1. A *disjunctive formula* over $\mathcal{B}$ is of the form $B_1(\mathbf{x}_1) \vee \cdots \vee B_t(\mathbf{x}_t)$ where $\mathbf{x}_1, \ldots, \mathbf{x}_t$ are sequences of variables from $\{x_1, \ldots, x_p\}$ such that the length of $\mathbf{x}_j$ equals the arity of B_j, and $B_1, \ldots, B_t \in \mathcal{B}$. The arity of a disjunctive formula $B_1(\mathbf{x}_1) \vee \cdots \vee B_t(\mathbf{x}_t)$ is t.
2. $\mathcal{B}^{\vee k} = \{R \mid R \text{ is definable by a disjunctive formula over } \mathcal{B} \text{ of arity } l \leq k\}$.

For simplicity we represent relations in $\mathcal{B}^{\vee k}$ by their defining disjunctive formulas. Two syntactially distinct disjunctive formulas may now denote the same relation, implying that

Table 1 The thirteen basic relations in Allen's interval algebra. The endpoint relations $x^s < x^e$ and $y^s < y^e$ that are valid for all relations have been omitted.

Basic relation		Example	Endpoints
x precedes y	p	xxx yyy	$I^+ < J^-$
y preceded by x	p^{-1}		
x meets y	m	xxxx yyyy	$I^+ = J^-$
y met-by x	m^{-1}		
x overlaps y	o	xxxx yyyy	$I^- < J^- < I^+$, $I^+ < J^+$
y overl.-by x	o^{-1}		
x during y	d	xxx yyyyyyy	$I^- > J^-$, $I^+ < J^+$
y includes x	d^{-1}		
x starts y	s	xxx yyyyyyy	$I^- = J^-$, $I^+ < J^+$
y started by x	s^{-1}		
x finishes y	f	xxx yyyyyyy	$I^+ = J^+$, $I^- > J^-$
y finished by x	f^-1		
x equals y	$\equiv$	xxxx yyyy	$I^- = J^-$, $I^+ = J^+$

this representation is not unique. To avoid tedious technicalities we ignore this issue and whenever convenient view constraint languages as multisets.

We are now ready to introduce *partition schemes* [20]. Let $\mathcal{B} = \{B_1, \ldots, B_m\}$ be a set of binary relations over a domain D. We say that $\mathcal{B}$ is *jointly exhaustive* if $\bigcup \mathcal{B} = D^2$ and that $\mathcal{B}$ is *pairwise disjoint* if $B_i \cap B_j = \emptyset$ whenever $i \neq j$. We say that $\mathcal{B}$ is a *partition scheme* if (1) $\mathcal{B}$ is jointy exhaustive and pairwise disjoint, (2) $\mathsf{eq}_D = \{(x, x) \mid x \in D\} \in \mathcal{B}$, and (3) for every $B_i \in \mathcal{B}$, the converse relation $B_i^\smile$ (i.e. $B_i^\smile = \{(y, x) \mid (x, y) \in B_i\}$) is in $\mathcal{B}$. We define $\mathcal{B}^{\vee=}$ to be the set of all unions of relations from $\mathcal{B}$. Equivalently, each relation in $\mathcal{B}^{\vee=}$ can be viewed as a disjunction $B_1(x, y) \vee B_2(x, y) \vee \cdots \vee B_k(x, y)$ for some $\{B_1, \ldots, B_k\} \subseteq \mathcal{B}$. We sometimes abuse notation and write $(B_1, \ldots, B_k)$ to denote the relation $B_1 \cup \cdots \cup B_k$. The set $\mathcal{B}^{\vee=}$ and the problem $\mathrm{CSP}(\Gamma)$ where $\Gamma \subseteq \mathcal{B}^{\vee=}$ are typical objects that are studied in artificial intelligence literature. For example, it has been common to use a *relation algebra* $\mathcal{A}$ as a starting point and then define a *network satisfaction problem* over $\mathcal{A}$, which in our notation is nothing else than the CSP over a set of binary relations. Note that if $\mathrm{CSP}(\mathcal{B})$ is polynomial-time solvable, then both $\mathrm{CSP}(\mathcal{B}^{\vee k})$ and $\mathrm{CSP}(\mathcal{B}^{\vee=})$ are members of NP.

▶ **Example 3.** *Allen's interval algebra* [2] is a well-known formalism for temporal reasoning where one considers relations between intervals of the form $I = [I^+, I^-]$, where $I^+, I^- \in \mathbb{R}$ is the start and end point, respectively. In Allen's algebra one can for instance describe that one interval begins before another interval, and one express such relations in terms of a partition scheme consisting of 13 basic relations (see Table 1), and then form more complicated relations by taking the union of the basic relations. If we let $\mathcal{A}$ denote the set of 13 basic relations in Allen's algebra, then $\mathrm{CSP}(\mathcal{A}^{\vee=})$ is an alternative formulation of the network consistency problem over Allen's algebra.

An extension of the interval algebra is the so-called *rectangle algebra* [13, 22]. Here, one considers relations between rectangles in the plane by extending the basic relations in the interval algebra to the projections of a rectangle onto the x- and y-axis, respectively. In other words, given $r, s \in \mathcal{A}$ and two rectangles represented by the intervals I_x, I_y, J_x, J_y we may define the relation $r \oplus s$ in the rectangle algebra holding if $I_x(r)J_x$ and $I_y(s)J_y$.

3 Partial Orders

CSPs based on partition schemes are very often used for *qualitative reasoning*. We acknowledge that it is not obvious how to define "qualitative reasoning" rigorously, but the concept seems to have an informal meaning that is generally accepted. Renz and Nebel [27, p. 161] write

> Qualitative reasoning is an approach for dealing with commonsense knowledge without using numerical computation. Instead, one tries to represent knowledge using a limited vocabulary such as qualitative relationships between entities or qualitative categories of numerical values, ...

Abstraction is the defining feature of qualitative reasoning: qualitative reasoning is about disregarding unnecessary and uninteresting details. With this in mind, it is clear that an important kind of qualitative relationships between objects are "part-of" relations. One may argue that such relations are strict partial orders that satisfy certain additional properties. We will now define three properties of strict partial orders, *infinite height*, *in-fork*, and *out-fork*, that appear to be relevant in the pursuit of classifying the complexity of $\mathrm{CSP}(\mathcal{B}^{\vee=})$. A typical example of such a relation is the NTPP relation in RCC-8 – this can be viewed as an archetypical example of a "part-of" relation. Many other relations that are not "part-of" relations satisfy these properties, too: one example is the precedes relation p in Allen's algebra. In fact, relations of this kind appear very frequently in CSPs for qualitative reasoning.

Let $\prec \subseteq D^2$ denote a binary relation let $\succ$ denote its converse $\prec^{\smile}$. We say that $\prec$ is a *strict partial order* if there is no $d \in D$ such that $d \prec d$ (irreflexivity) and for arbitrary $d, d', d'' \in D$: $d \prec d'$ and $d' \prec d''$ imply $d \prec d''$ (transitivity). Note that these two properties also ensure that $\prec$ is antisymmetric, i.e. if $d \prec d'$, then $d' \prec d$ does not hold.

We will now define three additional properties of strict partial orders. First define $\sqcap = D^2 \setminus \bigcup\{\prec, \succ, \mathsf{eq}_D\}$, and note that $x \sqcap y$ holds if and only if x and y are incomparable with respect to $\prec$.

▶ **Definition 4.** Let $\prec \subseteq D^2$ be a strict partial order over a domain D. We define the following properties.

C1. (infinite height) for every $k \geq 1$, there exists a sequence of elements $d_1, d_2, \ldots, d_k$ in D such that $d_1 \prec d_2 \prec \cdots \prec d_k$,

C2. (in-fork) if $a, b, c \in D$ and $a \prec b \prec c$, then there exists $d_1 \in D$ such that $d_1 \sqcap a$, $d_1 \sqcap b$, and $d_1 \prec c$, and

C3. (out-fork) if $a, b, c \in D$ and $a \prec b \prec c$, then there exists $d_2 \in D$ such that $d_2 \succ a$, $d_2 \sqcap b$, and $d_2 \sqcap c$.

Partial orders satisfying these three properties are abundant in the artificial intelligence literature, but has to the best of our knowledge not been explicitly formalized before. The conditions in-fork and out-fork are illustrated in Figure 1. Given a binary relation $\prec$ it is typically easy to check if it is a strict partial order of infinite height, but checking if it also satisfies in-fork and out-fork may need additional work. Consider Allen's algebra and the relation p, i.e. the relation stating that one interval appears strictly before another interval. In this case, $\sqcap$ is the relation that holds if and only if two distinct intervals have at least one point in common. Pick three intervals $I_j = [I_j^-, I_j^+]$, $1 \leq j \leq 3$, such that $I_1(\mathsf{p})I_2(\mathsf{p})I_3$. For in-fork, we choose $I_4 = [I_1^-, I_2^+]$ so that $I_4 \sqcap I_1$, $I_4 \sqcap I_2$, and $I_4 \prec I_3$. For out-fork, one may choose $I_5 = [I_2^-, I_3^+]$.

Let us consider another example where the domain contains the closed disks in $\mathbb{R}^2$ and the relation $\prec$ is the strict subset relation. Pick three disks $d_1, d_2, d_3 \in D$ such that $d_1 \prec d_2 \prec d_3$. How to choose suitable disks for verifying in-fork and out-fork is illustrated in Figure 2. This

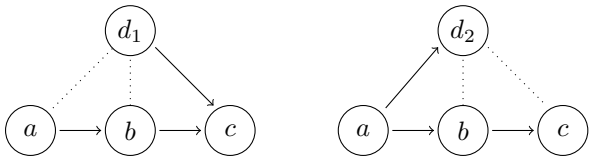


Figure 1 Illustration of in-fork (left) and out-fork (right). Arrows denote the $\prec$ relation and dotted lines the $\sqcap$ relation.

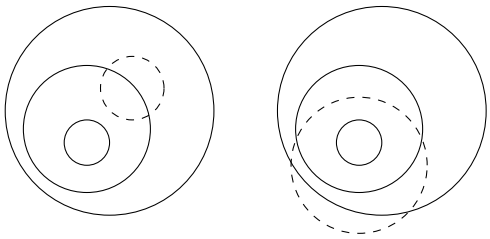


Figure 2 The dashed circles show possible choices of disks for in-fork (left) and out-fork (right).

example can easily be adapted to relations such as (PP) in RCC-5 and (NTPP) in RCC-8, and the relation $\mathsf{d} \oplus \mathsf{d}$ in the rectangle algebra. Many additional examples can be found in the survey by Dylla et al. [9], e.g. Goyal & Egenhofer's Cardinal Direction Calculus and Ragni & Scivos' Dependency Calculus. Last, let us remark that there are examples of strict partial orders that do not have in- or out-forks. Well-known examples are the less-than relation $<$ in the (1-dimensional) point algebra and in the branching time algebra. Interestingly, $\mathrm{CSP}(\mathcal{B}^{\vee=})$ is polynomial-time solvable in these two cases and we will come back to this observation at the end of Section 4.1.

4 Lower Bounds for $\mathrm{CSP}(\mathcal{B}^{\vee=})$

We will now study the computational complexity of $\mathrm{CSP}(\mathcal{B}^{\vee=})$ when $\mathcal{B}$ contains a strict partial order of infinite height with in- and out-forks. In Section 4.1, we prove that $\mathrm{CSP}(\mathcal{B}^{\vee=})$ is NP-hard and we use this result in Section 4.2 for proving that $\mathrm{CSP}(\mathcal{B}^{\vee=})$ cannot be solved in subexponential time (given that the ETH holds).

4.1 NP-hardness

NP-hardness of $\mathrm{CSP}(\mathcal{B}^{\vee=})$ for specific partition schemes $\mathcal{B}$ containing a strict partial order of infinite height with in- and out-forks has been proven many times in the literature. Examples where this connection is quite pronounced can be found in, for instance, Grigni et al. [12], Renz and Nebel [26], Moratz et al. [21], and Krokhin et al. [18] The basis for our reduction is the NP-complete problem Betweenness.

Instance: A finite set A and a collection T of ordered triples (a, b, c) of distinct elements from A.
Question: Is there a total ordering $<$ on A such that for each $(a, b, c) \in T$, we have either $a < b < c$ or $c < b < a$?

Our hardness result requires two steps that are presented in Lemma 5 and Theorem 6.

▶ **Lemma 5.** *Let $\mathcal{B}$ be a set of binary relations over a domain D containing a strict partial order $\prec$ of infinite height. Let $G(a, b, c, x_1, \ldots, x_m)$ be an instance (V, C) of CSP$(\mathcal{B}^{\vee=})$, where $V = \{a, b, c, x_1, \ldots, x_k\}$, and having the following properties.*

G1. *For arbitrary elements* $d_a, d_b, d_c \in D$ *such that* $d_a \prec d_b$ *and* $d_b \prec d_c$*, there exist elements* $d_1, \ldots, d_m \in D$ *such that the function* $s : V \to \{d_a, d_b, d_c, d_1, \ldots, d_m\}$ *defined by* $s(a) = d_a$*,* $s(b) = d_b$*,* $s(c) = d_c$*, and* $s(x_i) = d_i$*,* $1 \leq i \leq m$*, is a solution to the instance* $(V, C \cup \{a \prec b, b \prec c\})$*.*

G2. *For arbitrary elements* $d_a, d_b, d_c \in D$ *such that* $d_c \prec d_b$ *and* $d_b \prec d_a$*, there exist elements* $d_1, \ldots, d_m \in D$ *such that the function* $s : V \to \{d_a, d_b, d_c, d_1, \ldots, d_m\}$ *defined by* $s(a) = d_c$*,* $s(b) = d_b$*,* $s(c) = d_a$*, and* $s(x_i) = d_i$*,* $1 \leq i \leq m$*, is a solution to the instance* $(V, C \cup \{c \prec b, b \prec a\})$*.*

G3. $(V, C \cup \{b \prec a, b \prec c, a(\prec, \succ)c\})$ *is not satisfiable.*

G4. $(V, C \cup \{a \prec b, c \prec b, a(\prec, \succ)c\})$ *is not satisfiable.*

Let Γ *be the set of relations that appear in* G*. Then, CSP*$(\Gamma \cup \{\prec, \succ\})$ *is NP-hard.*

Proof. Let $\Gamma' = \Gamma \cup \{(\prec, \succ)\}$. We present a polynomial-time reduction from Betweenness to CSP(Γ'). Arbitrarily choose an instance (A, T) of Betweenness and construct an instance I of CSP(Γ') as follows:

1. for each pair of distinct elements $a, b \in A$, add the constraint $a(\prec, \succ)b$ to I, and
2. for each triple $(a, b, c) \in T$, introduce m fresh variables $x_1, \ldots, x_m$ and add $G(a, b, c, x_1, \ldots, x_m)$ to I.

We refer to the variables in I that correspond to the set A as basic variables and the other variables as auxiliary variables. We first assume that s is a solution to I. Let $S = \{s(a) \mid a \in A\}$. The constraints introduced in step (1) implies that the $|S| = |A|$ and the relation $\prec$ induces a total order on the set S. Assume to the contrary that there, for example, exists a triple $(a, b, c) \in T$ such that $s(b) \prec s(a) \prec s(c)$. Then, the instance $(V, C \cup \{b(\prec)a, b(\prec)c, a(\prec, \succ)c\})$ introduced in step (2) is satisfiable and this contradicts our assumptions. Analogously, we can rule out all orderings except $s(a) \prec s(b) \prec s(c)$ and $s(c) \prec s(b) \prec s(a)$. We conclude that there is a solution to the instance (A, T): for all $a, b \in A$, set $a < b$ if and only if $s(a) \prec s(b)$.

Assume now that there exists a solution $<$ to (A, T). We show how to construct a solution to the instance I. We rename the members of A such that $A = \{a_1, \ldots, a_n\}$ and $a_1 < a_2 < \cdots < a_n$. Arbitrarily choose elements $d_1, \ldots, d_n \in D$ such that $d_1 \prec d_2 \prec \cdots \prec d_n$. Such elements exist since $\prec$ is a strict partial order of infinite height. Let $s(a_i) = d_i$, $1 \leq i \leq n$, and note that s satisfies all constraints introduced in step 1.

Arbitrarily choose a triple $(a, b, c) \in T$ and consider the gadget $G(a, b, c, x_1, \ldots, x_k)$ that is introduced in step 2. If $a < b < c$, then $s(a) \prec s(b) \prec s(c)$ and $x_1, \ldots, x_k$ can be assigned values that satisfy the gadget by condition (1). If $c < b < a$, then $s(c) \prec s(b) \prec s(a)$ and $x_1, \ldots, x_k$ can be assigned values that satisfy the gadget by condition (2). Thus, for every triple $(a, b, c) \in T$, we can find values for the auxiliary variables that satisfy all G-gadgets. Note that two distinct G-gadgets do not have any auxiliary variables in common. We conclude that I is satisfiable. ◀

▶ **Theorem 6.** *Let* $\mathcal{B}$ *be a partition scheme with domain* D *containing a strict partial order* $\prec$ *of infinite height with in- and out-forks. Then CSP*$(\mathcal{B}^{\vee=})$ *is NP-hard.*

Proof. First observe that the relation $\sqcap = D^2 \setminus \bigcup\{\prec, \succ, \mathsf{eq}_D\}$ is a member of $\mathcal{B}^{\vee=}$ since $\mathcal{B}$ is a partition scheme. We will now define the following gadget: $G(a, b, c, x_1, x_2) = (\{a, b, c, x_1, x_2\}, \{x_1 \sqcap a, x_1 \sqcap b, x_1(\prec, \succ)c, x_2(\prec, \succ)a, x_2 \sqcap b, x_2 \sqcap c\})$. We demonstrate that G satisfies the preconditions of Lemma 5. We first consider the following condition:

C4. if $a \prec b \prec c$, then there does not exist $d_3 \in D$ such that $d_3 \sqcap a$, $d_3(\prec, \succ)b$, and $d_3 \sqcap c$.

We verify that C4 always holds under the assumptions stated in the theorem. Assume to the contrary that $a \prec b \prec c$ and $d_3 \in D$ satisfies $d_3 \sqcap a$, $d_3(\prec, \succ)b$, and $d_3 \sqcap c$. The relation $\prec$ is a strict partial order so it is transitive. If $d_3 \prec b$, then $d_3 \prec c$ and $d_3 \sqcap c$ cannot hold since the relations $\prec$ and $\sqcap$ are disjoint. Similarly, if $d_3 \succ b$, then $a \prec d_3$ and $d_3 \sqcap a$ cannot hold.

Next, we consider conditions G1 and G2 and show that they are satisfied: we see that proper assignments to variables x_1 and x_2 exist due to in-fork and out-fork. Assume to the contrary that G3 does not hold, i.e. $\{x_1 \sqcap a, x_1 \sqcap b, x_1(\prec, \succ)c, x_2(\prec, \succ)a, x_2 \sqcap b, x_2 \sqcap c\} \cup \{b(\prec)a, b(\prec)c, a(\prec, \succ)c\}$ is satisfiable. Under these constraints, two orderings of a, b, c are possible: $b \prec a \prec c$ and $b \prec c \prec a$. We consider the case $b \prec a \prec c$; the other case is analogous. Note now that $x_2 \sqcap b$, $x_2(\prec, \succ)a$, and $x_2 \sqcap c$. These constraints do not have a solution due to C4, and we conclude that G3 holds. That G4 holds can be shown analogously. The result then follows from Lemma 5. ◀

Hence, the properties in Definition 4 are sufficient for establishing NP-hardness of $\mathrm{CSP}(\mathcal{B}^{\vee=})$, and it is thus natural to ask to which extent they are also necessary. Although a complete answer seems difficult to obtain, we may at least observe that if $\prec \in \mathcal{B}$ is a strict partial order of finite height, then $\mathrm{CSP}(\mathcal{B}^{\vee=})$ is NP-hard, regardless of whether $\prec$ have in- and out-forks or not. This can be seen via a polynomial-time reduction from k-Colourability to $\mathrm{CSP}(\mathcal{B}^{\vee=})$ for some constant $k \geq 1$. Let (V, E) be an arbitrary undirected graph. Introduce variables $c_1, \ldots, c_k$ for each colour, and constrain them as $c_1(\prec)c_2(\prec)\ldots(\prec)c_k$. For each vertex $v \in V$, introduce a variable w and the constraints $w(\prec, \succ, \mathsf{eq}_D)c_i$, $1 \leq i \leq k$, and observe that $\succ, \mathsf{eq}_D \in \mathcal{B}$ since $\mathcal{B}$ is a partition scheme. Note that these constraints imply that w equals exactly one colour variable in any satisfying assignment. Finally, introduce the constraint $w(\prec, \succ)w'$ for each edge (v, v') in E. It is easy to verify that the resulting $\mathrm{CSP}(\mathcal{B}^{\vee=})$ instance has a solution if and only if (V, E) is k-colourable. It is also easy to verify that the reduction can be computed in polynomial time since k is a constant that only depends on the choice of $\mathcal{B}$. Since k-Colourability is NP-hard whenever $k \geq 3$, NP-hardness of $\mathrm{CSP}(\mathcal{B}^{\vee=})$ follows.

Similarly, it is natural to ask what happens if $\prec$ is a strict partial order of infinite height which does not have in- and/or out-forks. We have seen that this sometimes leads to tractability, as in the case of e.g. the point algebra and the branching time algebra, but this is not always the case. For a simple counter example, let $D = \{(0, i), (1, i), (2, i) \mid i \in \mathbb{N}\}$ and define $\prec \subseteq D^2$ such that $(a, b) \prec (c, d)$ if and only if $a = c$ and $b < d$. It is easy to verify that $\prec$ is a strict partial order of infinite height and that it does not have in- or out-forks. Let $\mathcal{B} = \{\prec, \succ, \sqcap, \mathsf{eq}_D\}$ where $\sqcap = D^2 \setminus \bigcup\{\prec, \succ, \mathsf{eq}_D\}$, and observe that $\mathcal{B}$ is a partition scheme. We show that $\mathrm{CSP}(\mathcal{B}^{\vee=})$ is an NP-hard problem via a polynomial-time reduction from 3-Colourability. Let (V, E) be an arbitrary undirected graph. For each vertex $v \in V$, introduce a variable w, and for each edge $(w, w') \in E$, introduce the constraint $w \sqcap w'$. Note that $((a, b), (c, d)) \in \sqcap$ if and only if $a \neq c$ and that a and c are restricted to the three-element set $\{0, 1, 2\}$. Given this, it is easy to verify that the resulting $\mathrm{CSP}(\mathcal{B}^{\vee=})$ instance has a solution if and only if (V, E) is 3-colourable.

4.2 ETH-based Lower Bound

Based on the results presented in the previous section, we will now show that $\mathrm{CSP}(\mathcal{B}^{\vee=})$ cannot be solved in subexponential time if $\mathcal{B}$ contains a strict partial order of infinite height with in- and out-forks, unless the exponential-time hypothesis (ETH) does not hold. If $\mathrm{CSP}(\Gamma)$ is solvable in $O(c^n)$ time by a deterministic algorithm for every $c > 1$ (where n denotes the number of variables) then $\mathrm{CSP}(\Gamma)$ is said to be *subexponential*. The exponential-time hypothesis is the conjecture that 3-SAT is not solvable in subexponential time [15].

The NP-hardness proof of BETWEENNESS by Opatrny [23] is based on a reduction from the RANK-3 HYPERGRAPH 2-COLOURABILITY problem. A *hypergraph* is a pair $H = (V, \mathcal{E})$ such that V is a non-empty finite set and $\mathcal{E}$ is a non-empty finite set of subsets of V. The elements of V are called the *nodes* of H and the elements of $\mathcal{E}$ are the *edges* of H. The *rank* of H is $\max\{|e| \mid e \in \mathcal{E}\}$, and the RANK-$k$ HYPERGRAPH 2-COLOURABILITY problem is defined as follows.

Instance: A rank-k hypergraph $H = (V, \mathcal{E})$.
Question: Do there exist sets $V_0, V_1 \subseteq V$ such that $V_0 \cap V_1 = \emptyset$ and $V_0 \cap e \neq \emptyset$, $V_1 \cap e \neq \emptyset$ for every $e \in \mathcal{E}$?

Define relations $R_1 = \{(0,1),(1,0)\}$ and $R_2 = \{0,1\}^3 \setminus \{(0,0,0),(1,1,1)\}$ and note that $\mathrm{CSP}(\{R_1, R_2\})$ is an obvious reformulation of the RANK-3 HYPERGRAPH 2-COLOURABILITY problem. Lemma 2 in Opatrny [23] immediately implies the following result.

▶ **Lemma 7.** *Let $I = (V, C)$ denote an arbitrary instance of CSP$(\{R_1, R_2\})$. It is possible to construct an instance (A, T) of the* BETWEENNESS *problem in polynomial time with the following properties.*

1. *I has a solution if and only if (A, T) has a solution,*
2. *$|A| \leq |V| + 1 + |C|$, and*
3. *$|T| \leq 2|C|$.*

▶ **Theorem 8.** *Assume the ETH holds. If $\mathcal{B}$ is a partition scheme such that $\prec \in \mathcal{B}$ and $\prec$ is a strict partial order of infinite height with in- and out-forks, then CSP$(\mathcal{B}^{\vee=})$ is not solvable in subexponential time.*

Proof. Results by Jonsson et al. [17] imply that $\mathrm{CSP}(\{R_1, R_2\})$-$B$ cannot be solved in subexponential time for some $B \geq 1$. Let $I = (V, C)$ denote an arbitrary instance of $\mathrm{CSP}(\{R_1, R_2\})$-$B$. Recall that $|C| \leq B \cdot |V|$ since each variable can occur in at most B constraints. Lemma 7 shows that we can (in polynomial time) construct an instance (A, T) of BETWEENNESS such that

1. I has a solution if and only if (A, T) has a solution,
2. $|A| \leq K \cdot |V|$, and
3. $|T| \leq L \cdot |C| \leq L \cdot B \cdot |V|$.

for some universal constants K, L. Lemma 5 combined with the standard gadget shows that we can (in polynomial time) construct an instance $I' = (V', C')$ of $\mathrm{CSP}(\mathcal{B}^{\vee=})$ such that

1. I' has a solution if and only if (A, T) has a solution and
2. $|V'| \leq |A| + 2|T|$.

Note that $|V'| \leq |A| + 2|T| \leq K|V| + 2L|C| \leq K|V| + 2LB|V| = (K + 2LB)|V|$. If $\mathrm{CSP}(\mathcal{B}^{\vee=})$ is solvable in subexponential time, then $\mathrm{CSP}(\{R_1, R_2\})$-$B$ is solvable in subexponential time, too, and this leads to a contradiction. ◀

In summary, we may rule out subexponential time algorithms for $\mathrm{CSP}(\mathcal{B}^{\vee=})$ for partition schemes $\mathcal{B}$ containing a strict partial order of infinite height with in- and out-forks. However, the best general algorithm for $\mathrm{CSP}(\mathcal{B}^{\vee=})$ runs in $O(2^{O(n^2)})$ time (if $\mathrm{CSP}(\mathcal{B})$ is tractable) [16, 27]. Hence, there is a large discrepancy between the upper and lower bound for this problem, suggesting that (at least) one of these bounds can be strengthened.

5 Lower Bounds for CSP($\mathcal{B}^{\vee k}$)

In this section, we will make use of the randomised version of ETH, and need a few additional definitions. First, let $\Gamma_{d,k}$, $d, k \geq 1$, denote the set of relations with arity at most k over the domain $\{1, \ldots, d\}$. A CSP algorithm A is said to be a $2^{c \cdot n}$-randomised algorithm if its running time is bounded by $2^{c \cdot n} \cdot poly(||I||)$ (where n is the number of variables) and its error probability is at most $1/3$. Let $c_{d,k} = \inf\{c \mid \exists\ 2^{c \cdot n}\text{-randomised algorithm for CSP}(\Gamma_{d,k})\}$. The variant of the ETH that we will use in the forthcoming lower bound states that $c_{2,3} > 0$, i.e., 3-SAT cannot be solved in subexponential time even if we are allowed to use randomised algorithms. We let r-ETH denote this hypothesis. Traxler [29] has shown the following result.

▶ **Theorem 9.** *If r-ETH holds, then there exists a universal constant $\alpha > 0$ such that for all $d \geq 3$, $\alpha \cdot \log(d) \leq c_{d,2}$.*

We begin by proving a result for $\mathcal{B}^{\vee 2}$ that is analogous to Theorems 6 and 8. Let $\mathcal{B}$ be a partition scheme over a domain D. Assume that $\mathcal{B}$ admits a gadget that forces three variables to be assigned distinct values, i.e., it is possible to define a non-empty ternary relation R such that $R \subseteq \{(x, y, z) \in D^3 \mid x \neq y, x \neq z, y \neq z\}$. This gadget can be defined for all examples considered in this paper, and in particular it can be defined by any strict partial order relating at least three elements (via $R(x, y, z) \equiv x \prec y \prec z$). Let $S(x, y, z) \equiv \mathsf{eq}_D(x, y) \vee \mathsf{eq}_D(x, z)$. Note that the relation S is a member of $\mathcal{B}^{\vee 2}$ since $\mathcal{B}$ is a partition scheme.

▶ **Theorem 10.** *Assume that $\mathcal{B}$ is a partition scheme admitting a gadget as described above. Then, CSP($\mathcal{B}^{\vee 2}$) is NP-hard, and if the r-ETH holds, there is no $2^{\frac{c_{3,2}}{5} \cdot n}$-randomised algorithm for CSP($\mathcal{B}^{\vee 2}$).*

Proof. We present a polynomial-time reduction from CSP($\Gamma_{3,2}$) to CSP($\mathcal{B}^{\vee 2}$). If the given CSP($\Gamma_{3,2}$) instance contains n variables, then the CSP($\mathcal{B}^{\vee 2}$) instance will contain at most $5n + K$ variables where K is a constant. By Theorem 9, CSP($\Gamma_{3,2}$) cannot be solved in $2^{c_{3,2} n}$ time, so CSP($\mathcal{B}^{\vee 2}$) cannot be solved in $2^{\frac{c_{3,2}}{5} n}$ time.

Let (V, C) be an arbitrary instance of CSP($\Gamma_{3,2}$) where $V = \{x_1, \ldots, x_n\}$. To construct our CSP($\mathcal{B}^{\vee 2}$) instance, we perform the following steps.

1. Introduce three variables d_1, d_2, d_3 and the gadget that makes them distinct. These variables will be used to denote the three domain elements.
2. For each variable x_i, $1 \leq i \leq n$, we introduce the variable x'_i.
3. For each variable x_i, $1 \leq i \leq n$, introduce the variable y_i together with the constraints $S(y_i, d_2, d_3)$ and $S(x_i, d_1, y_i)$. These constraints imply that x_i is equal to d_1, d_2, or d_3.
4. For each variable x_i, $1 \leq i \leq n$, introduce variables $x_i^{\neq 1}, x_i^{\neq 2}, x_i^{\neq 3}$ together with the constraints $S(x_i^{\neq 1}, d_2, d_3)$, $S(x_i^{\neq 2}, d_1, d_3)$, and $S(x_i^{\neq 3}, d_1, d_2)$. These variables are used for "simulating" inequalities in step 5.
5. For each constraint $R(x_i, x_j) \in C$ and each tuple $(a, b) \in \{1, 2, 3\}^2$ that is not in R, introduce the constraint $\mathsf{eq}_D(x_i, x_i^{\neq a}) \vee \mathsf{eq}_D(x_j, x_j^{\neq b})$.

The resulting CSP($\mathcal{B}^{\vee 2}$) instance (V', C') can obviously be constructed in polynomial time. It contains $5n$ variables plus the constant number of variables needed for the gadget. We claim that (V', C') has a solution if and only if (V, C) has a solution. Assume that $f : V \to \{1, 2, 3\}$ is a solution to (V, C). Let $c_1, c_2, c_3 \in D$ be three distinct values that are permitted by the gadget. Let U denote the set of other values used by the gadget. Define $f' : V' \to U \cup \{c_1, c_2, c_3\}$ as follows.

- f' assigns suitable values from U to the gadget,
- $f'(d_i) = c_i$, $1 \leq i \leq 3$,
- $f'(y_i) = c_2$ if $f(x_i) = 2$ and $f'(y_i) = c_3$ otherwise,
- $f'(x'_i) = c_{f(x_i)}$
- $f'(x_i^{\neq 1}) = c_{f(x_i)}$ if $f(x_i) \neq 1$ and $f'(x_i^{\neq 1}) = c_2$, otherwise,
- $f'(x_i^{\neq 2}) = c_{f(x_i)}$ if $f(x_i) \neq 2$ and $f'(x_i^{\neq 2}) = c_1$, otherwise,
- $f'(x_i^{\neq 3}) = c_{f(x_i)}$ if $f(x_i) \neq 3$ and $f'(x_i^{\neq 3}) = c_2$, otherwise.

The function f' can easily be seen to satisfy the constraints introduced in steps 1, 3 and 4. We consider the constraints introduced in step 5. Pick a constraint $R(x_i, x_j) \in C$ and a tuple $(a, b) \in \{1, 2, 3\}^2$ that is not in R. We assume without loss of generality that $a = 1$ and $b = 2$. The corresponding constraint in C' is now $\mathsf{eq}_D(x'_i, x_i^{\neq 1}) \vee \mathsf{eq}_D(x'_j, x_j^{\neq 2})$. We know that $f(x_i) \neq 1$ or $f(x_j) \neq 2$. Assume, for example, that $f(x_i) = 2$ and $f(x_j) = 2$. We see that $f'(x_i^{\neq 1}) = c_2$ and $f'(x_j^{\neq 2}) = c_2$ so f' satisfies this constraint. The other cases can be verified analogously.

Assume that $f' : V' \to U \cup \{c_1, c_2, c_3\}$ is a solution to (V', C') where c_i, $1 \leq i \leq 3$, is the value assigned to variable d_i. Define $f : V \to \{1, 2, 3\}$ such that $f(x_i) = p$ when $f(x'_i) = c_p$. Arbitrarily choose a constraint $R(x_i, x_j) \in C$ and assume to the contrary that $(f(x_i), f(x_j)) = (a, b) \notin R$. This implies that f' satisfies the constraint $\mathsf{eq}_D(x'_i, x_i^{\neq a}) \vee \mathsf{eq}(x'_j, x_j^{\neq b})$ that was introduced in step 5. In order to do so, either $f'(x'_i) = f'(x_i^{\neq a})$ and $f'(x'_i) \neq c_a$ or $f'(x'_j) = f'(x_j^{\neq b})$ and $f'(x'_j) \neq c_b$. In both cases, $(f(x_i), f(x_j)) \neq (a, b)$ and this leads to a contradiction. ◀

If we consider $\mathcal{B}^{\vee k}$ with larger k and require that certain relations are members of $\mathcal{B}$, then stronger lower bounds can be obtained.

▶ **Theorem 11.** *Let $\prec \subseteq D^2$ be a strict partial order of infinite height over a domain D. If the r-ETH holds, then there is no $2^{c \cdot n}$-randomised algorithm for CSP($\{\prec\}^{\vee 4}$) for any $c \geq 0$.*

Proof. Assume there exists a $2^{c \cdot n}$-randomised algorithm for CSP($\{\prec\}^{\vee 4}$). Arbitrarily choose $d \geq 3$ such that $c_{d,2} > c$. We show how to polynomial-time reduce CSP($\Gamma_{d,2}$) to CSP($\{\prec\}^{\vee 4}$) in a way such that only a constant number of new variables are introduced. This implies that CSP($\Gamma_{d,2}$) can be solved by a $2^{c \cdot n}$-randomised algorithm where $c < c_{d,2}$ which contradicts the r-ETH due to Traxler's result.

Let $I = (V, C)$ be an arbitrary instance of CSP($\Gamma_{d,2}$). We assume (without loss of generality) that the variable domain is $\{1, \ldots, d\}$. Introduce $d + 1$ fresh variables $V_1 = \{a_1, \ldots, a_{d+1}\}$ and define $C_1 = \{a_1(\prec)a_2, a_2(\prec)a_3, \ldots, a_d(\prec)a_{d+1}\}$. Since $\prec$ is a strict partial order of infinite height, we know that $I_1 = (V_1, C_1)$ is satisfiable. In every solution s, it holds that $s(a_i) \prec s(a_j)$ when $1 \leq i < j \leq d + 1$ by the transitivity of $\prec$. We then constrain each $x \in V$ as follows: $a_1(\prec)x$, $x(\prec)a_i \vee a_i(\prec)x$ for $2 \leq i \leq d$, and $x(\prec)a_{d+1}$. Let C_2 denote the corresponding set of constraints and let $I_2 = (V \cup V_1, C_1 \cup C_2)$. It is easy to verify that in every solution s to I_2, each variable $x \in V$ satisfies $s(a_i) \prec s(x) \prec s(a_{i+1})$ for exactly one $1 \leq i \leq d$. For each constraint $S(x, y)$ in C, we finally introduce the following set of constraints $\{x(\prec)a_e \vee a_{e+1}(\prec)x \vee y(\prec)a_{e'} \vee a_{e'+1}(\prec)y \mid (e, e') \notin S\}$.

Let C_3 denote the resulting set of constraints and let $I_3 = (V \cup V_1, C_1 \cup C_2 \cup C_3)$. We claim that I_3 is satisfiable if and only if I is satisfiable. Assume that I_3 has the solution s_3. We know that every variable v in V satisfies $s(a_i) \prec s(v) \prec s(a_{i+1})$ for exactly one $1 \leq i \leq d$. The constraints in C_3 assure that s_3 assigns values to the variables in V that are consistent with the constraints in (V, C). Thus, the function $s : V \to D$ defined by $s(v) = i$ where $v \in V$ and $s_3(a_i) \prec s_3(v) \prec s_3(a_{i+1})$ is a solution to I.

Assume that I has the solution s. We construct a solution s_3 to I_3 as follows. Arbitrarily choose $e_1, \ldots, e_{d+1}, e'_1, \ldots, e'_d$ in D such that $e_i \prec e'_i \prec e_{i+1}$, $1 \leq i \leq d$; such elements exists since $\prec$ has infinite height. Let $s_3(a_i) = e_i$, $1 \leq i \leq d+1$. This choice satisfies all constraints in C_1. Let $s_3(v) = e'_i$, $v \in V$, when $s(v) = i$. It follows from the choice of $e_1, \ldots, e_{d+1}, e'_1, \ldots, e'_d$ that all constraints in C_2 are satisfied. Finally, s_3 satisfies the constraints in C_3: this is an immediate consequence of s being a solution to the instance I combined with the restrictions imposed by the constraints in $C_1 \cup C_2$.

Last, we verify that I_3 can be computed in polynomial time. The constraints in C_1 and C_2 can be computed in constant time since d is fixed, and each constraint in C gives rise to at most d^2 new constraints in C_3, so this set can trivially be computed in polynomial time. ◀

The bound in Theorem 11 is substantially stronger than the bounds that we have been able to prove for CSP($\mathcal{B}^{\vee=}$). We may also observe that CSP($\{\prec\}^{\vee k}$), $k \geq 1$, is solvable in $O(|V|! \cdot \mathrm{poly}(||I||)) = 2^{O(|V| \log |V|)} \cdot \mathrm{poly}(||I||)$ time, implying that the lower bound in Theorem 11 does not admit large improvements (unless r-ETH fails).

▶ **Theorem 12.** *Let $\prec \subseteq D^2$ be a strict partial order of infinite height over a domain D, and let $k \geq 1$. Then CSP($\{\prec\}^{\vee k}$) is solvable in $O(|V|! \cdot \mathrm{poly}(||I||))$ time.*

Proof. Let (V, C) be an instance of CSP($\{\prec\}^{\vee k}$). For each total order $\sqsubset$ over V, we answer yes if there for every disjunctive clause in C exists a disjunct $x \prec y$ such that $x \sqsubset y$. The time complexity of this algorithm is clear, and we now turn to correctness. Assume first that f is a satisfying assignment to (V, C). Let C' denote the set of all disjuncts satisfied by f. This set induces a strict partial order which can be extended into a total order by topological sorting. For the other direction, assume that $\sqsubset$ satisfies at least one disjunct in every clause. Let $i_1, \ldots, i_{|V|} \subseteq \{1, \ldots, |V|\}$ be indices such that $x_{i_1} \sqsubset \ldots \sqsubset x_{i_{|V|}}$ and $|\{i_1, \ldots, i_{|V|}\}| = |V|$. Since $\prec$ is of infinite height there then exists $d_1, \ldots, d_{|V|} \in D$ such that $d_1 \prec \ldots \prec d_{|V|}$, and we can form a satisfying assignment f by letting $f(x_{i_j}) = d_j$ for every $i_j \in \{i_1, \ldots, i_{|V|}\}$. ◀

6 Discussion

Our main focus has been to study the complexity of CSPs over partition schemes $\mathcal{B}$, with a particular emphasis on CSP($\mathcal{B}^{\vee=}$) when $\mathcal{B}$ contains a strict partial order. We have identified three properties resulting in NP-hardness, which explains the NP-hardness for many different CSP problems. Towards a better understanding of the time complexity of these problems we have also proven lower bounds under complexity-theoretic assumptions. We have studied lower bounds for CSP($\mathcal{B}^{\vee k}$), too, and obtained general bounds for this kind of problems. At this stage it is worth to yet again point out that none of our results require model-theoretic assumptions such as ω-categoricity, i.e., that the first-order theory of $\mathcal{B}$ admits only one model up to isomorphism. A large amount of research on infinite-domain CSPs has concentrated on ω-categorical constraint languages. However, there are interesting problems that are not amenable using this approach.

▶ **Example 13.** Bodirsky and Jonsson [5, Sec. 4.2] present a partition scheme $\mathcal{B}$ with domain $\mathbb{R}^3$ that demonstrate how to integrate arithmetics into partition schemes. They show that $\mathcal{B}$ is not ω-categorical and there does not exist any ω-categorical constraint language Γ such that CSP(Γ) and CSP($\mathcal{B}$) is the same computational problem. We will not define $\mathcal{B}$ explicitly, but remark that the relation $\mathrm{Less} = \{((a,b,p),(c,d,q)) \subseteq (\mathbb{R}^3)^2 \mid a < c \wedge p \neq q\}$ is a member of $\mathcal{B}^{\vee=}$. Obviously, the constraints $\mathrm{Less}(d_1, d_2)$ and $\mathrm{Less}(d_2, d_3)$ force d_1, d_2, d_3

to be assigned distinct values. By definition, there exists relations $B_1, \ldots, B_k \in \mathcal{B}$ such that Less $= B_1 \cup \cdots \cup B_k$ so there exist (not necessarily distinct) $1 \leq i, j \leq k$ such that the constraints $B_1(d_1, d_2)$ and $B_2(d_2, d_3)$ force d_1, d_2, d_3 to be assigned distinct values, too. We know that $\mathsf{eq}_D \in \mathcal{B}$ since $\mathcal{B}$ is a partition scheme. We conclude (by Theorem 10) that CSP($\mathcal{B}^{\vee 2}$) cannot be solved in $O(2^{\frac{c_{3,2}}{5}n})$ time.

One important consequence of lower bound results is that they can be used to rule out certain types of algorithms. First of all, k-consistency algorithms are not applicable since they run in polynomial time for arbitrary fixed k. The powerful generalisation of k-consistency, the *Datalog* framework [11, 4], is not applicable either since every Datalog program runs in polynomial time, too. Another example is provided by graph-decomposition algorithms for CSPs (for instance, algorithms that exploit treewidth). Such algorithms have been highly influential in the CSP context [1, 3, 7], but they typically result in polynomial-time or subexponential algorithms and are therefore unlikely to be usable for CSP($\mathcal{B}^{\vee=}$) problems. Even more can be said if we take a detour via degree-bounded problems.

▶ **Lemma 14.** *Let $\mathcal{B}$ be a constraint language such that CSP($\mathcal{B}$) is solvable in polynomial time. For arbitrary constants B and k, CSP($\mathcal{B}^{\vee k}$)-B can be solved in $2^{B \cdot \log k \cdot |V|} \cdot poly(||I||)$ time and CSP($\mathcal{B}^{\vee=}$)-B can be solved in $2^{B \cdot \log(|\mathcal{B}|-1) \cdot |V|} \cdot poly(||I||)$ time.*

Proof. Let $I = (V, C)$ be an arbitrary instance of CSP($\mathcal{B}^{\vee k}$)-B. Pick one disjunct out of each constraint in C, put the disjuncts into the set S, and check whether S is satisfiable or not. This check can be performed in polynomial time. There are at most $k^{B \cdot |V|}$ different sets S since each constraint contains at most k disjuncts and there are at most $B \cdot |V|$ constraints in C. Furthermore, (V, C) is satisfiable if and only if at least one of them is satisfiable. We conclude that (V, C) can be solved in $k^{B \cdot |V|} \cdot poly(||I||)$ time. The proof for CSP($\mathcal{B}^{\vee=}$)-B is essentially identical, with the difference that we never need to consider a relation containing all relations in $\mathcal{B}$, explaining $|\mathcal{B}| - 1$ in the exponent. ◀

Thus, both CSP($\mathcal{B}^{\vee k}$)-B and CSP($\mathcal{B}^{\vee=}$)-B can be solved in $2^{O(n)}$ time. We know (from Theorem 11) that there is no 2^{cn}-randomised algorithm for the CSP($\{\prec\}^{\vee 4}$) problem. This shows that techniques used for transforming CSP instances into sparse instances, e.g. *linear kernelisations* [19], are unlikely to be applicable to CSP($\mathcal{B}^{\vee k}$). We cannot rule out linear kernelisations for CSP($\mathcal{B}^{\vee=}$), though, since we do not have a sufficiently strong lower bound in this case.

Naturally, there are approaches that are not directly ruled out by our lower bounds. Jonsson and Lagerkvist [16] have presented general results for obtaining algorithms based on enumeration of domain values. These algorithms are sometimes much faster than the branching algorithms that are typically used for infinite-domain CSPs: the branching algorithm for CSP($\mathcal{A}^{\vee=}$) runs in $2^{O(n^2)}$ time while the enumeration-based algorithm runs in $2^{O(n \log n)}$ time. The range of applicability for enumeration-based algorithms is unfortunately not well understood, and more work is needed to clarify this. Another viable approach is to use methods that have been successful in solving finite-domain CSPs. Einarson [10] demonstrates how the finite-domain version of the PPSZ algorithm [14] can be applied to infinite-domain CSPs. His results are inconclusive: the algorithm is faster than previously known algorithms for certain CSP($\mathcal{B}^{\vee k}$) problems but it is, for instance, not competetive for Allen's interval algebra CSP($\mathcal{A}^{\vee=}$).

These examples suggest that it may be worthwhile to strengthen the subexponential lower bound for CSP($\mathcal{B}^{\vee=}$) even further – if possible. One possible way of doing this is to exploit the *strong exponential-time hypothesis*, i.e. the conjecture that SAT is not solvable in $O^*(c^n)$

time for any $c < 2$, The challenge here is that the SETH intrinsically requires reductions where one can "simulate" clauses of arbitrary high arity with a very small overhead. This seems difficult for $\mathrm{CSP}(\mathcal{B}^{\vee=})$ and in Theorem 8 we could only produce a reduction from a SAT problem with a linear number of constraints. This assumption cannot be made for SAT since sparsification, the process of reducing an instance to a subexponential number of instances with a linear number of constraints, is not possible for SAT [28]. Another possibility is to use bounds based on the Chromatic Number problem: Jonsson and Lagerkvist [16, Th. 21] have related the time complexity of Allen's interval algebra with the time complexity of the Chromatic Number problem and obtained concrete lower bounds of the form $O^*(c^n)$ for a constant $c > 1$ depending on the complexity of Chromatic Number. Thus, we ask the following: should stronger lower bounds for $\mathrm{CSP}(\mathcal{B}^{\vee=})$ be pursued in the setting of CNF-SAT and the SETH, or are problems of this kind fundamentally closer to e.g. colouring problems?

References

1 J. Alber, H. Fernau, and R. Niedermeier. Parameterized complexity: exponential speed-up for planar graph problems. *J. Algorithms*, 52(1):26–56, 2004.

2 J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.

3 S. Arora, B. Barak, and D. Steurer. Subexponential algorithms for unique games and related problems. *J. ACM*, 62(5):42:1–42:25, 2015.

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

5 M. Bodirsky and P. Jonsson. A model-theoretic view on qualitative constraint reasoning. *J. Artif. Intell. Res. (JAIR)*, 58:339–385, 2017.

6 A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *Proc. 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS-2017)*, pages 319–330, 2017.

7 R. de Haan, I. A. Kanj, and S. Szeider. On the subexponential-time complexity of CSP. *J. Artif. Intell. Res.*, 52:203–234, 2015.

8 I. Düntsch. Relation algebras and their application in temporal and spatial reasoning. *Artif. Intell. Rev*, 23(4):315–357, Jun 2005.

9 F. Dylla, J. H. Lee, T. Mossakowski, T. Schneider, A. van Delden, J. van de Ven, and D. Wolter. A survey of qualitative spatial and temporal calculi: Algebraic and computational properties. *ACM Comput. Surv.*, 50(1):7:1–7:39, 2017.

10 C. Einarson. An extension of the PPSZ algorithm to infinite-domain constraint satisfaction problems. Master's thesis report, Department of Computer and Information Science, Linköpings Universitet, 2017.

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

12 M. Grigni, D. Papadias, and C. H. Papadimitriou. Topological inference. In *Proc. 14th International Joint Conference on Artificial Intelligence (IJCAI-1995)*, pages 901–907, 1995.

13 H. Güsgen. Spatial reasoning based on Allen's temporal logic. Technical report ICSI TR89-049, International Computer Science Institute, 1993.

14 T. Hertli, I. Hurbain, S. Millius, R. A. Moser, D. Scheder, and M. Szedlák. The PPSZ algorithm for constraint satisfaction problems on more than two colors. In *Proc. 22nd International Conference on Principles and Practice of Constraint Programming (CP-2016)*, pages 421–437, 2016.

15 R. Impagliazzo and R. Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

16 P. Jonsson and V. Lagerkvist. An initial study of time complexity in infinite-domain constraint satisfaction. *Artif. Intell.*, 245:115–133, 2017.

17 P. Jonsson, V. Lagerkvist, G. Nordh, and B. Zanuttini. Strong partial clones and the time complexity of SAT problems. *J. Comput. Syst. Sci.*, 84:52–78, 2017.

18 A. Krokhin, P. Jeavons, and P. Jonsson. Reasoning about temporal relations: The tractable subclasses of Allen's interval algebra. *J. ACM*, 50(5):591–640, 2003.

19 V. Lagerkvist and M. Wahlström. Kernelization of constraint satisfaction problems: A study through universal algebra. In *Proc. 23rd International Conference on Principles and Practice of Constraint Programming (CP-2017)*, pages 157–171, 2017.

20 G. Ligozat and J. Renz. What is a qualitative calculus? A general framework. In *Proc. 8th Pacific Rim International Conference on Artificial Intelligence (PRICAI-2004)*, pages 53–64, 2004.

21 R. Moratz, J. Renz, and D. Wolter. Qualitative spatial reasoning about line segments. In *Proc. 14th European Conference on Artificial Intelligence (ECAI-2000)*, pages 234–238, 2000.

22 A. Mukerjee and G. Joe. A qualitative model for space. In *Proc. 8th National Conference on Artificial Intelligence (AAAI-1990)*, pages 721–727, 1990.

23 J. Opatrny. Total ordering problem. *SIAM J. Comput.*, 8(1):111–114, 1979.

24 J. Renz. Qualitative spatial and temporal reasoning: Efficient algorithms for everyone. In *Proc. 20th International Joint Conference on Artifical Intelligence (IJCAI-2007)*, pages 526–531, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

25 J. Renz and J. J. Li. Automated complexity proofs for qualitative spatial and temporal calculi. In *Proc. Principles of Knowledge Representation and Reasoning (KR-2008)*, pages 715–723, 2008.

26 J. Renz and B. Nebel. On the complexity of qualitative spatial reasoning: A maximal tractable fragment of the region connection calculus. *Artif. Intell.*, 108(1–2):69–123, 1999.

27 J. Renz and B. Nebel. Qualitative spatial reasoning using constraint calculi. In Marco Aiello, Ian Pratt-Hartmann, and Johan van Benthem, editors, *Handbook of Spatial Logics*, pages 161–215. Springer, 2007.

28 R. Santhanam and S. Srinivasan. On the limits of sparsification. In *Proc. 39th International Colloquium on Automata, Languages, and Programming (ICALP-2012)*, pages 774–785, 2012.

29 P. Traxler. The time complexity of constraint satisfaction. In *Proc. 3rd International Workshop on Parameterized and Exact Computation (IWPEC-2008)*, pages 190–201, 2008.

30 D. Zhuk. A proof of CSP dichotomy conjecture. In *Proc. 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS-2017)*, pages 331–342, 2017.

Directed Graph Minors and Serial-Parallel Width

Argyrios Deligkas
Leverhulme Research Centre, University of Liverpool, UK
argyrios.deligkas@liverpool.ac.uk

Reshef Meir
Faculty of Industrial Engineering and Management, Technion, Israel
reshefm@ie.technion.ac.il

Abstract

Graph minors are a primary tool in understanding the structure of undirected graphs, with many conceptual and algorithmic implications. We propose new variants of *directed graph minors* and *directed graph embeddings*, by modifying familiar definitions. For the class of 2-terminal directed acyclic graphs (TDAGs) our two definitions coincide, and the class is closed under both operations. The usefulness of our directed minor operations is demonstrated by characterizing all TDAGs with serial-parallel width at most k; a class of networks known to guarantee bounded negative externality in nonatomic routing games. Our characterization implies that a TDAG has serial-parallel width of 1 if and only if it is a directed series-parallel graph. We also study the computational complexity of finding a directed minor and computing the serial-parallel width.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis, Mathematics of computing → Graph theory

Keywords and phrases directed minors, pathwidth

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.44

Related Version A full version of the paper is available at `https://tinyurl.com/y9hcukyz`.

1 Introduction

Graph theory has been one of the fundamental tools in computer science since its inception and in many computational problems the inputs are in a form of a graph, e.g., analysis of electric circuits and communication networks, and training of neural nets. More important still, numerous problems from various domains are often solved by reducing them to some algorithmic problem on a graph. Some prominent examples include search and path-finding [31]; planning graphs [3]; constraint satisfaction [26]; AND-OR graph [4]; and inference in Bayesian networks [7].

The *structure* of these graphs is often crucial to the modeling of the problem. For instance, the last two examples above use *directed acyclic graphs* (DAGs), which are also used to represent belief structures, influence relations and decision diagrams [15]. Restrictions on the degree, maximum length, or other properties of the underlying graph, can be exploited: problems that are not guaranteed to have a solution in general may behave better on some classes of graphs, and many algorithms are guaranteed to have a lower runtime subject to structural assumptions.

Graph minors

When considering *undirected graphs*, some of the primary tools of structural analysis use *graph embeddings* and *graph minors*. These are substructures whose exclusion from a graph indicates certain "simplicity" properties. Some famous results are the characterization of planar graphs [23], and of graphs with bounded treewidth [32] by excluded minors. In fact, for undirected graphs there is by now a sound theory of graph minors with many applications; see, e.g., [25] for a survey, and [8] for algorithmic implications. The culmination of this theory is the Graph Minor Theorem [40, 33], which states that *any* class of undirected graphs that is closed under the minor operation, can be characterized by a finite set of excluded minors.

Perhaps the most important application of graph minor theory to computer science is its use for developing efficient algorithms on graphs with bounded treewidth and/or other properties [13, 6, 34]. Graph minors were also recently used to characterize classes of graphs induced by planning problems to identify potential effects of time-inconsistent planning [21, 38].

Although the graphs encountered in many theoretical and realistic problems are *directed*, there is no single theory of directed graph minors, and results are far more scarce than in the undirected case. Several papers suggested various definitions of directed minors, embeddings, and subdivisions, and provided various characterization results [17, 20, 14, 18, 19, 22]. However, each such definition uses different graph operations, some of which we explain in detail later on. Certain notions of directed minors are only applicable for subclasses of directed graphs. For example, the definitions in [22] apply only to minors with a certain structure called "crown".

In this paper, we will be interested mainly in directed graphs that are acyclic (DAG), or 2-terminal, or both (TDAG). 2-terminal graphs occur in routing [1], circuit analysis [36] and in many planning problems [21]. Thus understanding the structure of graphs in these classes is an important challenge.

Paper structure and contribution. In the first part of the paper (Section 3) we define new notions of graph embedding and graph minor for general directed graphs.[1] We show that for the class of 2-terminal directed acyclic graphs (TDAGs) these two operations exactly reverse one another. Thus, a TDAG G' is a directed minor of G if and only if it is embedded in G. Also, the class of TDAG is closed under directed minor and directed embedding operations. We thus argue that our definitions provide a sound basis for a theory of directed graph minors, at least for the class of TDAGs.

To demonstrate the usefulness of our directed minor theory, we apply it in Section 4 to characterize TDAGs with bounded *parallel width* and *serial-parallel width*. The parallel width of a graph corresponds to the maximal cut separating the source from the target. Serial-parallel width of a graph is a parameter recently introduced in the context of routing games [28], and it is useful for bounding negative externalities. We describe a finite set of graphs (generalized variants of the Braess/Wheatstone network) whose exclusion as directed minors of a TDAG G is necessary and sufficient to determine that G has serial-parallel width lower than k, for any k.

In Section 5 we settle several computational questions arising from our definitions. Some proofs are omitted due to space constraints and are available in the full version of this paper which is attached at the end of the file.

[1] To avoid confusion, we should note that the term *graph embedding* is used in the machine learning literature to describe embedding of graphs in various topological or metric spaces (e.g., [41]), which is a very different problem.

2 Preliminaries

For convenience, we will use the letter H for undirected graphs, and the letter G for directed graphs. We denote a path in graph $\langle V, E\rangle$ by $(v_1, v_2, \ldots, v_m)$, where for every $i \leq m-1$, $(v_i, v_{i+1}) \in E$. We use dash to abbreviate the path, e.g. $a - b - c$ is an abbreviation to a path $(a, \ldots, b, \ldots, c)$; if more than one such path exists, we refer to one of them arbitrarily, unless stated otherwise.

If nodes x, y are on some path p, then p_{xy} denotes the *open* subpath of p between nodes x and y, and $[p_{xy}] = x - p_{xy} - y$ the *closed* subpath that includes the extreme vertices.

▶ **Definition 1** (2-terminal graph [29, 14])**.** A *2-terminal [directed] graph* $G = \langle V, E, s, t\rangle$ is a [directed] multigraph with no self-loops and two distinguished vertices $s, t \in V$, such that every vertex and edge belong to at least one [directed] simple $s - t$ path.

A *forward-subtree* of a directed 2-terminal graph G is a subset of edges that form a directed tree with a single source. Similarly, a *backward-subtree* of G is a subset of edges that form a directed tree with a single target.

A directed 2-terminal graph with no cycles is referred to as *TDAG* (2-Terminal Directed Acyclic Graph). The vertices of a TDAG can always be sorted in increasing order, called *topological order*, so that all edges, and thus all directed paths, are from v_i to v_j for some $j > i$. In particular, s and t are the first and last vertices, respectively.

▶ **Lemma 2.** *A DAG is a TDAG if and only if it has a unique source and a unique sink.*

3 Directed Graph Minors and Embeddings

In undirected graphs, a graph H' is called a *minor* of H if H' can be obtained from H by a sequence of edge deletions and contractions. As an example of a simple characterization via exclusion of minors, observe that any graph H (not a multigraph) is acyclic if and only if it excludes a triangle as a minor.

3.1 Directed minors.

There are several extensions of the notion of a minor to directed graphs. One that is closest to our needs is the *butterfly minor* [17], see Def. 3 without the underlined part. However, neither the class of 2-terminal graphs nor the class of TDAGs is closed under the butterfly minor operation, since, for example, it may leave an isolated node. We thus modify it by restricting which edges may be deleted (underlined).

▶ **Definition 3** (Directed minor)**.** A graph G' is a *directed minor* (or simply *a d-minor*) of a directed graph G, if G' can be obtained from G by a sequence of the following local operations:

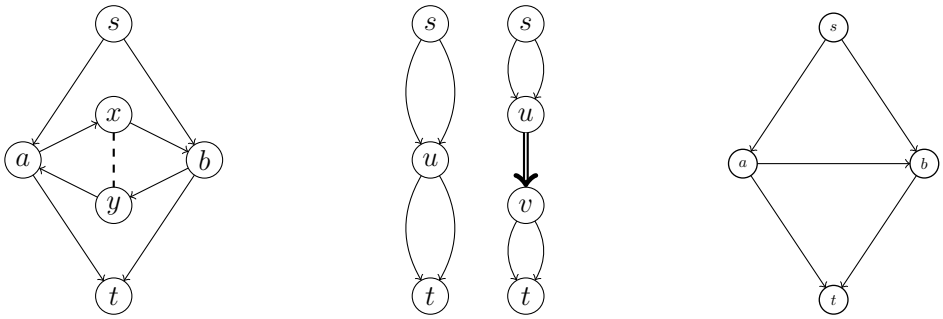
Deletion. Deleting an edge (a, b) where a has outdegree at least 2, and b has indegree at least 2.

Backward contraction. Contracting an edge (a, b) where b has indegree 1.

Forward contraction. Contracting an edge (a, b) where a has outdegree 1.

For example, the edge (a, b) in Fig. 1c may not be contracted, but can be backward-contracted after the edge (s, b) is deleted.

▶ **Lemma 4.** *The class of directed acyclic graphs is closed under d-minor operations.*



(a) **(b)** **(c)** The Braess graph.

Figure 1 The graph in Fig. 1a is a directed 2-terminal graph (solid edges only). Adding the dashed edge (x, y), regardless of its direction, results in a non-2-terminal graph. Fig. 1b: The graph G' on the left is d-embedded in G on the right, as we can forward-split u into (u, v) (u retains all incoming edges, and v retains at least one outgoing edge). However, there is no edge we can add or subdivide to get G from G' so G' is not h-embedded in G. The Braess graph G_B is on Fig. 1c. Examples.

3.2 Graph Embeddings

There are various definitions of graph embeddings and subdivisions [12, 29, 14], which can be summarised together as follows.

▶ **Definition 5** (Homeomorphic embedding)**.** A [directed] graph G' is *h-embedded* in (or a topological minor of) G, if G (or a graph isomorphic to G) can be derived from G' by a sequence of the following operations:

Addition. The addition of a new edge joining two existing vertices.

Subdivision. Replacement of an edge (a, b) by two edges (a, x) and (x, b).

Terminal extension. (only for 2-terminal graphs) Addition of a new edge e joining s or t with a new vertex, which becomes the new source or target.

For an undirected graph H', every h-embedding operation maintains various properties like being a 2-terminal graph. However, for a 2-terminal directed graph G', an h-embedding operation may not maintain this property (see Fig. 1a). Also, this set of operations is not rich enough for our needs. Thus, we propose a new definition for directed embeddings.

▶ **Definition 6** (Directed embedding)**.** A directed graph G' is *d-embedded* in a directed graph G if G' is isomorphic to G or to a graph derived from G by a sequence of the following operations:

Addition. Addition of a new edge (a, b), such that there is no path $b - a$.

Forward split. Replacement of node $a \neq t$ with outdegree greater than zero, by two nodes a_1 and a_2 and an edge (a_1, a_2), where a_1 retains all incoming edges, and a_2 retains at least one outgoing edge.

Backward split. Replacement of node $a \neq s$ with indegree greater than zero, by two nodes a_1 and a_2 and an edge (a_1, a_2), where a_2 retains all outgoing edges, and a_1 retains at least one incoming edge.

It is not hard to see that a subdivision of an edge (directed or undirected) can be replicated by splitting one of its end nodes, and a terminal extension can be replicated by splitting the terminal (backward split of s or forward split of t). We thus allow the operations of **edge subdivision** and **terminal extension** as valid d-embedding operations as well.

▶ **Lemma 7.** *The classes of 2-terminal directed graphs and directed acyclic graphs are closed under d-embedding.*

In particular, if G' is a TDAG and G' is d-embedded in G, then G is a TDAG.

For a 2-terminal directed graph G, the graph G' is a *valid subgraph* of G if it is a subgraph of G and is also 2-terminal. While the next lemma may seem trivial, note that it does not hold for general 2-terminal directed graphs, since a single edge is not d-embedded in any cyclic graph.

▶ **Lemma 8.** *Let G be a TDAG. If G' is a valid subgraph of G, then G' is d-embedded in G.*

We will need the following lemma later on, but it is useful to know regardless. An immediate corollary is that embedding steps only add paths and increase the connectivity of a graph.

▶ **Lemma 9.** *If G, G' differ by a single split step of vertex a into (a, b), then there is a one to one mapping between paths in G' and paths in G.*

3.3 Relations among graph operations

The way we defined them, d-minors are more restrictive than butterfly minors, whereas d-embeddings are more permissive than h-embeddings when restricting attention to acyclic graphs; see Fig. 1b. However, d-embeddings are not infinitely richer than h-embeddings. A vertex is called a *hub* if it has both an indegree and an outdegree larger than one.

▶ **Proposition 10.** *Let $G' = \langle V', E' \rangle$ and let $J \subseteq V'$ be the hubs of G'. There is a set $\mathcal{G}$ of at most $2^{|J| \times |V'|^2}$ graphs, such that for any $G = \langle V, E \rangle$, graph G' is d-embedded in G if and only if some graph in $\mathcal{G}$ is h-embedded in G. Each such graph has at most $|V|(1 + |J|)$ vertices.*

For the class of TDAGs, the concepts of directed-minor and directed-embedding turn out to be equivalent.

▶ **Theorem 11.** *Let G and G' be TDAGs. G' is d-embedded in G if and only if G' is a d-minor of G.*

Intuitively, addition and deletion operations cancel one another, as do split and contraction operations. This equivalence does not hold for general directed graphs, as added edges may not qualify for deletion (e.g. if we add an edge (a, b) where a has only incoming edges), and vice versa (if we remove an edge that is part of a cycle).

Proof. By induction, it is sufficient to show this for G', G that differ by a single d-embedding or d-minor operation. "⇒" There are 3 cases, depending on the embedding operation:

1. The addition of edge (a, b) to G' can be reversed by deleting the same edge from G. Note that $b \neq s$ as otherwise there is a path in G' from $b = s$ to a, and similarly $a \neq t$. Thus, a has outdegree at least 1 in G' and at least 2 in G. Similarly, b has indegree at least 2 in G, and thus deleting the edge (a, b) is a valid d-minor step.
2. Suppose that a vertex a in G' is split to $\{a, b\}$ with a forward split. Then, since a retains all incoming edges, b has a single incoming edge (a, b) in G. Thus, we can contract the edge (a, b) in G using backward contraction.
3. Similarly, a backward split can be reversed with a forward contraction.

"⇐" There are 3 cases, depending on the d-minor operation:

1. If the edge (a, b) is deleted from G, then since G is acyclic there is no path $b - a$. Thus adding (a, b) to G' is a valid d-embedding step.

2. Suppose that the edge (a, b) in G is backward-contracted to some vertex x in G'. This means that b has a single incoming edge. Thus all edges incoming to the pair $\{a, b\}$ are leading to a. Let $R(a)$ and $R(b)$ be the out-neighbors of a and b in G, respectively. Then by forward-splitting node x in G' and split the outgoing edges of x according to $R(a)$ and $R(b)$, we get the graph G^i.
3. Similarly, forward contraction can be reversed with backward split. ◀

4 Serial-Parallel Width

A *cut* in a 2-terminal graph $G = \langle V, E, s, t\rangle$ is a set of edges $C \subseteq E$ such that there is no $s - t$ path in $E \setminus C$. C is *minimal* if there is no cut $C' \subsetneq C$.

A set of edges $S \subseteq E$ is *parallel* if there is some $C \subseteq E$ s.t. $S \subseteq C$, and C is a minimal cut;S is *serial* if there is a simple directed $s - t$ path p that contains S.

▶ **Definition 12** (Parallel Width)**.** The *parallel width* of a directed 2-terminal graph, $PW(G)$, is the size of the largest parallel set $S \subseteq E$.

▶ **Definition 13** (Serial-Parallel Width [28])**.** The *serial-parallel width* of a directed 2-terminal graph, $SPW(G)$, is the size of the largest set $S \subseteq E$ that is both serial and parallel.

Intuitively, the parallel width is the size of a maximum $s-t$ cut. For example, the width of an electric circuit coincides with the parallel width of its underlying TDAG [5]. A serial-parallel width of k means that there are at least k source-target paths, and some additional path that edge-intersects all of them. It was shown in [28] that in nonatomic routing games with diverse players, the negative externality is bounded by the serial-parallel width of the underlying network.

▶ **Example 14.** Consider the Braess graph in Fig. 1c. The minimal $s - t$ cuts are: $\{sa, sb\}, \{at, bt\}$, $\{sa, bt\}$, and $\{sb, ab, at\}$. The set $\{sa, bt\}$ is both parallel and serial, which means $SPW(G_B) \geq 2$. The set $\{sa, at\}$ is serial but not parallel; and $\{sa, sb, ab\}$ is neither. In fact, the *only* parallel set of size greater than 2 is $\{sb, ab, at\}$, which is not serial, thus $SPW(G_B) < 3$. We conclude that the serial-parallel width of the Braess graph is 2.

In contrast, both graphs in Fig. 1b have $PW(G) = 2$ and $SPW(G) = 1$.

For any 2-terminal graph G, we have $1 \leq SPW(G) \leq |V| - 1$. The lower bound is since any single edge is both parallel and serial, and the upper bound since there is no simple path of length $|V|$ or more.

▶ **Definition 15.** For any $k \geq 2$, we define the *k-serial-parallel graph* $G_{SP(k)}$ as follows. $G = \langle V, E, s, t\rangle$, where $V = \{s, t, a_2, \ldots, a_k, b_1, \ldots, b_{k-1}\}$, and $E = \bigcup_{i=2}^{k-1}\{(s, a_i), (a_i, b_i), (b_i, t), (b_i, a_{i+1})\} \cup \{(s, b_1), (a_k, t)\}$ (see Fig.2). Furthermore, $G_{P(k)}$ is a TDAG that contains k internally disjoint $s - t$ paths.

▶ **Definition 16.** A graph G is a *variant of* $G_{SP(k)}$ if we replace the edges $\{(s, a_i)\}_{i=2}^{k}$ with an arbitrary forward-subtree that respects the lexicographic order $s, a_2, \ldots, a_k$, and replace the edges $\{(b_i, t)\}_{i=1}^{k-1}$ with an arbitrary backward-subtree that respects the lexicographic order $b_1, \ldots, b_{k-1}, t$.

The serial-parallel width of $G_{SP(k)}$ is exactly k, where
$\{(s, b_1), (a_2, b_2), \ldots, (a_{k-1}, b_{k-1}), (a_k, t)\}$ are the serial-parallel edges.

The graph $G_{SP(k)}$ was used under different names in [2, 30, 28], usually to derive examples of games with high equilibrium costs.

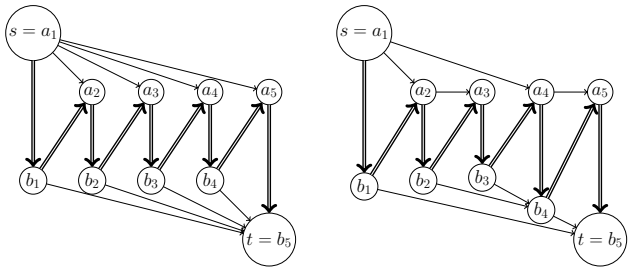


Figure 2 The left figure is the graph $G_{SP(5)}$, and the right figure is a variant of it. For convenience, the long path in each graph appears in double lines, and the forward- and backward-trees in thin lines.

▶ **Lemma 17.** *If S is a set of parallel [serial] edges in a 2-terminal graph G', then after any sequence of d-embedding steps on G', the set S is still parallel [resp., serial]. In particular, if G' is d-embedded in G then $PW(G) \geq PW(G')$ and $SPW(G) \geq SPW(G')$.*

Proof sketch. For serial sets the statement is obvious.

Consider a sequence of J d-embedding operations on $G^0 = G'$ that ends in $G^J = G$. Suppose that S is parallel. Let C^0 be a minimal cut in $G^0 = G'$ containing S. We show by induction that after every step $j \leq J$ there is a minimal cut C^j in G^j, such that $C^{j-1} \subseteq C^j$.

Assume by induction that C^{j-1} is a minimal cut in G^{j-1}. The graph G^j differs from G^{j-1} either by a single added edge, or by a single split vertex. By Lemma 9 split steps do not change the set of paths, and thus $C^j = C^{j-1}$ is still a minimal cut. Thus suppose G^j differs by an addition step of an edge $e = (a, b)$. Either C^{j-1} is still a cut in G^j, or e connects a node a reachable from s to a node b with a path to t. In the latter case, $C^j = C^{j-1} \cup \{e\}$ is a cut. To see that C^j is minimal suppose we remove an edge $e' \neq e$. If $C' = C^j \setminus \{e'\}$ is a cut in G^j, then $C' \setminus \{e\} = C^{j-1} \setminus \{e'\}$ is a cut in G^{j-1}, in contradiction to the induction hypothesis that C^{j-1} is minimal. In either case, S is still contained in a minimal cut C^j after every operation, and in particular contained in a minimal cut C^J of $G^J = G$. ◀

4.1 Characterization of graphs with bounded serial-parallel width

Before we get to our main theorem we start with a characterization of parallel sets.

▶ **Proposition 18** (Parallel sets characterization)**.** *Let $G = \langle V, E, s, t\rangle$ be a TDAG, and a set of k edges $S \subseteq E$, where for each $e_i \in S$, $e_i = (a_i, b_i)$. The following conditions are equivalent: (1) S is parallel; (2) there is a forward-subtree T_s in G with root s and leafs $\{a_i\}_{i\leq k}$, and a backward-subtree T_t in G with leaf t and roots $\{b_i\}_{i\leq k}$; (3) there is a sequence of d-minor operations that deletes or contracts all edges except S.*

Proof. "1 ⇒ 2": Suppose that S is parallel, then it is contained in a minimal cut C. Let G_C be graph G without the edges of C. Let T_s be all vertices reachable from s in G_C, and T_t all vertices from which t is reachable and let $G(X)$ be the subgraph of G induced by $X \subseteq V$. $T_s \cap T_t = \emptyset$ as otherwise there is a path from s to t in G_C. Also, $a_i \in T_s$ for all i, as otherwise the edge e_i can be removed from C and $C \setminus \{e_i\}$ is still a cut. Likewise for $b_i \in T_t$. Since G is a TDAG, and $G(T_s)$ contains a path from s to every a_i, then $G(T_s)$ is w.l.o.g. a forward-tree. Similarly for T_t.

"2 ⇒ 3": The union of $G(T_s), G(S)$, and $G(T_t)$ is a valid subgraph G' of G of which S is a minimal cut: for any e_i there is a path $s - a_i - b_i - t$. Since G' is a valid subgraph of G, then by Lemma 8 it is d-embedded and thus a d-minor of G. Then, since all nodes in T_s

have indegree at most 1, we can backward-contract all of T_s to a single node s. Similarly, we forward-contract all of T_t to the node t, and we are left with a graph that has two nodes whose only edges are S.

"$3 \Rightarrow 1$": By Theorem 11 we can consider the reverse sequence of d-embedding operations from $G^0 = G_{P(k)}$ to $G^J = G$. By Lemma 17, the set S is still parallel after every operation and in particular in G. ◀

We get a characterization of graphs with bounded parallel width as a simple corollary.

▶ **Theorem 19.** *For any TDAG G and $k \geq 2$, $PW(G) \geq k$ if and only if $G_{P(k)}$ is a d-minor of G.*

Proof. "$\Rightarrow$": Consider some parallel set S of size k. By Prop. 18 there is a sequence of d-minor operations that ends with a graph whose only edges are S. This graph is $G_{P(k)}$. "$\Leftarrow$": Follows directly from Lemma 17 and Thm. 11, since $PW(G_{P(k)}) = k$. ◀

▶ **Theorem 20** (Main Theorem)**.** *For any TDAG G and $k \geq 2$, $SPW(G) \geq k$ if and only if some variant of $G_{SP(k)}$ is a d-minor of G.*

Proof. "$\Rightarrow$": Consider the graph G. Suppose that $SPW(G) \geq k$, then there is a set $S = \{e_1, \ldots, e_k\}$ that is part of a minimal cut C between s and t. Denote $e_i = (a_i, b_i)$.

By Prop. 18, G has a forward-subtree T_s with root s and leafs $\{a_i\}_{i \leq k}$, and a backward-subtree T_t in G with leaf t and roots $\{b_i\}_{i \leq k}$. Also, by definition of the parallel width, there is a *simple* $s - t$ path p' containing S, w.l.o.g. in lexicographic order.

We now describe a series of d-minor steps on G that will result in a variant of $G_{SP(k)}$. Delete all edges and vertices that are not part of p', T_s or T_t. This leaves us with a graph G' that is a valid subgraph of G and thus, by Lemma 8 and Thm. 11, is also a d-minor of G.

p' is composed of a sequence of subpaths between vertices $s, y_1, x_2, y_2 \ldots, x_{k-1}, y_{k-1}, x_k, t$, where each x_i is the first intersection of $[p'_{b_{i-1}a_i}]$ with T_s. Thus x_i is an ancestor of (or coincides with)a_i in T_s. Similarly, $\{y_i\}_{i=1}^{k-1}$ are on the backward-subtree T_t, where y_i is the last intersection of $[p'_{b_i a_{i+1}}]$ and T_t. Denote by $A_i \subseteq \{a_2, \ldots, a_k\}$ all leafs of the subtree of T_s rooted at x_i, and by $B_i \subseteq \{b_1, \ldots, b_{k-1}\}$ all roots of the subtree of T_t whose leaf is y_i. In particular, $a_i \in A_i$, and $a_j \notin A_i$ for $j < i$, as otherwise there is a cycle $x_i - a_j - b_j - y_j - x_i$. Likewise, $b_i \in B_i$ and $b_j \notin B_i$ for $j > i$.

Note that the indegree of all nodes in T_s is 1, except for $\{x_i\}_{i=2}^{k}$ whose indegree is 2 (one edge from the parent in T_s, and one from the predecessor node on p'), and s whose indegree is 0. We thus backward-contract all edges in T_s that do not point to some x_i. We get a forward-subtree $\hat{T}_s$:

- The root of $\hat{T}_s$ is $s = x_1$, and its nodes are $\{x_i\}_{i=2}^{k}$;
- Each path $x_i - a_i$ in G' becomes a single node $x_i = a_i$ in $\hat{G}$;
- The subtree rooted by x_i in T_s becomes a subtree in $\hat{T}_s$ over nodes A_i maintaining their order, i.e., children have higher index than their parent. For example, in Fig. 2 on the right, $A_4 = \{a_4, a_5\}$ and a_4 is a parent of a_5.

We similarly contract T_t to $\hat{T}_t$ on nodes $\{y_i\}$.

The last step is to contract every subpath $[p'_{y_i x_{i+1}}]$ to a single edge (y_i, x_{i+1}). Denote the union of these edges by $\hat{F}$, so that $S \cup \hat{F}$ is the path we get after contracting p'.

We get that the contracted graph $\hat{G}' = S \cup \hat{F} \cup \hat{T}_s \cup \hat{T}_t$ is isomorphic to a variant of $G_{SP(k)}$. More specifically, s and t are mapped to themselves, each x_i for $i = 2, \ldots, k$ in $\hat{G}$ is mapped to a_i in $G_{SP(k)}$, and each y_i for $i = 1, \ldots, k-1$ in $\hat{G}'$ is mapped to b_i in $G_{SP(k)}$. For each $i = 2, \ldots, k$, let j be the maximal index such that x_j is an ancestor of x_i in T_s. If such

j exists, then the parent of a_i in $\hat{G}'$ is a_j, and otherwise its parent is $s = a_1$. The parent of a_i in $G_{SP(k)}$ is the closest ancestor x_j of the node x_i in T_s (and similarly for the child of b_i).

"⇐": Follows directly from Lemma 17 and Thm. 11, since SPW for any variant of $G_{SP(k)}$ is k. ◀

Since $G_{SP(k)}$ has $2k$ vertices, we get that $SPW(G) \leq \frac{|V|}{2}$. Another corollary of Theorem 20 is a generalization of the lower bounds on negative externality from [2, 28]. These papers show how instances with high externality (depending on k) can be constructed from any variant of $G_{SP(k)}$. By Theorem 20 this is true for *any graph* G with $SPW(G) \geq k$.

4.2 Series-parallel graphs

Series-parallel 2-terminal graphs have been long studied in contexts such as electric circuits [9], complexity of graph algorithms [37], and also routing games [29, 11].

▶ **Definition 21** (Series-parallel graph [10, 16]). *[directed] series-parallel graph* is a 2-terminal graph $\langle V, E, s, t\rangle$, and is either a single edge (s, t), or is composed recursively by one of the two following steps:

Serial composition. Combine two [directed] 2-terminal graphs $\langle V_1, E_1, s_1, t_1\rangle, \langle V_2, E_2, s_2, t_2\rangle$ serially by merging t_1 with s_2.

Parallel composition. Combine two [directed] 2-terminal graphs $\langle V_1, E_1, s_1, t_1\rangle$, $\langle V_2, E_2, s_2, t_2\rangle$ in parallel by merging s_1 with s_2, and t_1 with t_2.

Our last result in this section is showing that directed series-parallel graphs (DSP) characterize exactly the 2-terminal graphs with serial-parallel width of 1.

▶ **Proposition 22** ([14]). *Let G be a 2-terminal directed graph. Then G is a DSP if and only if the directed Braess graph G_B is not h-embedded in G.*

Proposition 22 and the relation between h-embeddings and d-embeddings yield the following.

▶ **Theorem 23.** *Let G be a TDAG, and let $k \geq 2$. The following conditions coincide. (1) G is a directed series-parallel graph. (2) The directed Braess graph G_B is not d-embedded in G. (3) $SPW(G) = 1$.*

Proof. Note that G_B has no hubs, as all vertices have at most 3 neighbors. Thus by Prop. 10, G_B is d-embedded in G if and only if it is h-embedded (as $|J| = 0$, $\mathcal{G}$ contains only G_B itself). Thus we get (1) $\iff$ (2).

(2) $\iff$ (3) follows as a special case from Thm. 20. ◀

5 Computational Problems

We first ask whether we can efficiently decide when a directed graph is 2-terminal.

▶ **Proposition 24.** *It is $\mathcal{NP}$-complete to decide if a directed graph is 2-terminal, but in $\mathcal{P}$ if the graph is acyclic.*

The next two natural computational questions accept as input 2-terminal graphs G and G'.

IsDMinor: Is G' a d-minor of G?

IsDEmbedded: Is G' d-embedded in G?

The complexity may depend on whether the graphs are TDAGs (in which case the questions coincide), and also on whether G' is a fixed graph of size k. We write down some of our results explicitly, and summarize all of them in Table 1.

Table 1 The computational complexity of problems we study. Results without references either follow from other results in the table or from known results.
* - IsDEmbedded is easy if the minor G' is acyclic.

	2-terminal graph		TDAG	
	any k	fixed k	any k	fixed k
IsSerial	$\mathcal{NP}$-c	$\mathcal{NP}$-c [P. 25]	$\mathcal{P}$	$\mathcal{P}$
IsParallel	?	?	?	$\mathcal{P}$ [P. 26]
IsSerialParallel	$\mathcal{NP}$-c	$\mathcal{NP}$-c [P. 25]	?	$\mathcal{P}$ [P. 26]
MaxSerial	$\mathcal{NP}$-c	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$
MaxParallel	$\mathcal{NP}$-c	?	$\mathcal{NP}$-c [P. 27]	$\mathcal{P}$ [C. 28]
MaxSerialParallel	?	?	?	$\mathcal{P}$ [C. 28]
IsDMinor	$\mathcal{NP}$-c	?	$\mathcal{NP}$-c	$\mathcal{P}$
IsDEmbedded	$\mathcal{NP}$-c	$\mathcal{P}$ *	$\mathcal{NP}$-c	$\mathcal{P}$

5.1 Testing properties of edge sets

We are interested in the following questions on a given 2-terminal graph $G = \langle V, E, s, t\rangle$ and a set $S = \{(a_i, b_i)\}_{i\leq k}$ of k edges:
IsSerial: Is there an $s - t$ path containing S?
IsParallel: Is S parallel?
IsSerialParallel: Is S both serial and parallel?
Note that since all of these properties are phrased in terms of existence, containment in $\mathcal{NP}$ is trivial.

Our main tool in many of the results, both positive and negative, will be the m-VertexDisjointPaths problem: given a directed graph $G = \langle V, E\rangle$ and m pairs of vertices $\{(x_i, y_i)\}_{i\leq m}$, find whether there are vertex-disjoint paths $x_i - y_i$ in G for all $i \leq m$. This problem is equivalent to that of checking if a graph G' is h-embedded in G [12], yet using it for our problems requires some modifications. The problem is $\mathcal{NP}$-complete even when G is a DAG [39], and $\mathcal{NP}$-complete for $m = 2$ in general directed graphs [12]. In contrast, it is in $\mathcal{P}$ when G is a DAG *and* m is fixed [12].

▶ **Proposition 25.** IsSerial *and* IsSerialParallel *are $\mathcal{NP}$-complete even for $k = 3$.*

For $k = 1$ every instance is a 'yes' instance, as any single edge is part of a simple path and part of a minimal cut.

The most tricky part is the complexity of identifying a parallel set. Using some of the structural results obtained in the previous sections, we can show the following.

▶ **Proposition 26.** IsParallel *is in $\mathcal{P}$ for TDAGs and fixed k.*

Proof. Denote $e_i = (a_i, b_i)$ for any $e_i \in S$. Denote $A = \{a_1, \ldots, a_k\}$ and $B = \{b_1, \ldots, b_k\}$. By Prop. 18, it suffices to decide if G contains a forward-subtree T_s to all of A, and a backward-subtree T_t from all of B to t. Note that T_s contains at most $k - 1$ "junctions", i.e., nodes with outdegree greater than one (including s). Suppose first that we guess what these vertices are and what is their hierarchy, and denote them by $X = \{s = x_1, \ldots, x_{k'}\}$ and relations T_X. We similarly guess a set Y of junctions in T_t and the relations among them T_Y. Our algorithm works as follows:

- For every x_j with degree d_j in T_X, split x_j into $d_j + 1$ nodes such that one of them x_j^0 retains all incoming edges (entry port), and each of the other $x_j^{v_j}$ (exit port) retains all outgoing edges. v_j is the first node from $X \cup A$ downward from x_j on T_s.

- Connect x_j^0 to all of $x_j^{v_j}$.
- Similarly split each $y_j \in Y$ to multiple entry ports and a single exit port.
- Find vertex-disjoint paths from each exit port to the entry port of one child in T_X or T_Y, respectively. E.g. from $x_j^{v_j}$ to a_i if $v_j = a_i$ for some $i \leq k$, or to $x_{j'}^0$ if $v_j = x_{j'}$ for some $j' \leq k'$.

Consider the algorithm above. The total number of edges in each tree T_X, T_Y is at most $2k$, so the total number of paths we seek in each iteration is less than $4k$. Such paths, if exist, can be found in time $|V|^{O(k^2)}$ due to the result of [12].

If such vertex-disjoint paths exist, then merging back all copies of each junction will provide us with a disjoint forward-subtree T_s and backward-tree T_t. In the other direction, if such trees exist and use junctions X and Y respectively, then the paths between every two junctions are vertex-disjoint except in the junctions themselves. Since we split each junction, these paths will be fully vertex disjoint. Thus the algorithm will always find trees T_s, T_t using junctions X, Y, if such exist.

The total number of iterations is the number of ways to select $2k$ vertices out of $|V|$, times the number of trees we can try on each set of size $2k$ (less than $(2k)^{(2k)}$ by Cayley's formula), so in total no more than $|V|^{O(k^2)}$ iterations.

The total runtime is $|V|^{O(k^2)}$ which is polynomial for fixed k.

In the full version of the paper we have shown a polynomial time algorithm to determine if S is serial(even polynomial in k for TDAGs). Hence, we can check whether S is serial-parallel by checking each property separately. ◀

5.2 Testing width properties of graphs

Given 2-terminal graph G and an integer k we study:
MaxSerial: Is there a serial set S of size $\geq k$?
MaxParallel: Is $PW(G) \geq k$?
MaxSerialParallel: Is $SPW(G) \geq k$?

▶ **Proposition 27.** MaxParallel *is $\mathcal{NP}$-complete even on TDAGs.*

Proof. MaxParallel problem is in $\mathcal{NP}$. Given any 2-terminal directed graph $G = \langle V, E\rangle$ and a set S of edges in E we can easily check whether S is an $s-t$ cut; if S is indeed an $s-t$ cut, then by deleting the edges in S there is no directed path from s to t and this can be easily verified via Dijkstra algorithm .

To show completeness we reduce from MaxDiCut on DAGs [24]. In an instance of MaxDiCut problem we are given a directed acyclic graph $G = \langle V, E\rangle$ and an integer k, and we are asked if there is a partition of V into two sets V_1 and V_2 so that the cardinality of the edge set $C = \{(u, v) \in E | u \in V_1, v \in V_2\}$ is at least k. We construct a 2-terminal DAG G' as follows. We add the vertex s and we connect it with every vertex $v \in V$ via an edge directed from s to v. Furthermore, we add the vertex t and we connect it with every vertex $v \in V$ via an edge directed from v to t. Clearly, G' is a 2-terminal graph. Furthermore, it is not hard to see that no directed cycles were created. Thus, G' is a 2-terminal DAG. We will prove that there exists a directed cut of size k in G if and only if there exists an $s-t$ directed cut of size $|V| + k$ in G'.

Firstly, assume that in G there exists a partition of V into V_1 and V_2 such that the size of C, i.e., the number of directed edges from V_1 to V_2, is k. Then, the set S that contains C, the edges from the vertices of V_1 to t and the edges from s to vertices of V_2, is a minimal $s-t$ cut. Observe, $|S| = |C| + |V_1| + |V_2| = k + |V|$. To see why S is an $s-t$ cut, observe

that there is no path of the form $s - v - t$ with $v \in V$, because one of the edges (s, v) and (v, t) is missing. The only other way to reach t from s is to go from s to some vertex of V_1, move to V_2, and then reach t. But every edge from V_1 to V_2 is in C, hence there is no such $s - t$ path. Furthermore, S is minimal since for any edge (u, v) in C there is clearly a path $s - u - v - t$ in G' that does not contain any other edge in S, and for any other edge in $S \setminus C$ there is an $s - t$ path of length three that does not use any other in S.

For the other direction now, consider a minimal $s - t$ cut S in G' of size $|V| + k$. Denote by A all the vertices accessible from s in $E \setminus S$, and by B all other vertices of G. The cut S contains every edge from A to B, every edge from s to B, and every edge from A to t, so in particular we get that the size of the cut defined by the partition of V to A and B in G is exactly $|S| - (|A| + |B|) = |V| + k - |V| = k$. Finally, observe that the partition defined by A and B is a directed cut for G, because otherwise there would be a directed $s - t$ path and thus S would not be an $s - t$ cut. ◀

As an immediate corollary we get that IsDMinor and IsDEmbedded are $\mathcal{NP}$-complete even on a TDAG. When $G' = \langle V', E' \rangle$ is fixed, both problems are in $\mathcal{P}$: we use the algorithm of [12] for h-embedding as a subroutine on at most $2^{|V'|^3}$ graphs due to Proposition 10.

Since by Theorems 19 and 20 finding the parallel (or serial-parallel) width is equivalent to check for excluded minors whose size is a function of k, we get the following.

▶ **Corollary 28.** MaxParallel *and* MaxSerialParallel *are in* $\mathcal{P}$ *for TDAGs and fixed* k.

6 Discussion

Many different variations of operations can be used to obtain "simple" graphs that capture the essential forbidden properties of large classes of graphs: minors, embeddings, subdivisions, etc. These operations should be rich enough to allow for a small set of forbidden graphs, but restricted enough to only capture the intended class.

We believe that d-embeddings and d-minors will turn out to be useful, beyond the applications demonstrated in the paper. For example, in [21] bad graphs for planning are identified by *undirected minors*, which mislabels many graphs due to ignoring edge directions. A tighter characterization could be obtained by d-minors.

It is interesting whether d-embeddings or d-minors can be used to characterize other classes of directed graphs, such as graphs with bounded triangular width [27] or D-width [35]. Finally, there is the question of whether a directed graph version of the Graph Minor Theorem holds for d-minors or d-embeddings [19].

References

1 I. Ashlagi, D. Monderer, and M. Tennenholtz. Two-terminal routing games with unknown active players. *Artificial Intelligence*, 173(15):1441–1455, 2009.

2 M. Babaioff, R. Kleinberg, and C. Papadimitriou. Congestion games with malicious players. In *EC*, pages 103–112. ACM, 2007.

3 A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial intelligence*, 90(1-2):281–300, 1997.

4 C. Chang and J. Slagle. An admissible and optimal algorithm for searching AND/OR graphs. *Artificial Intelligence*, 2(2):117–128, 1971.

5 B. Codenotti and M. Leoncini. *Parallel Complexity of Linear System Solution*. World Scientific, 1991.

6 D. Cohen, M. Cooper, P. Jeavons, and S. Zivny. Tractable classes of binary csps defined by excluded topological minors. In *IJCAI*, pages 1945–1951, 2015.

7 G. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial intelligence*, 42(2-3):393–405, 1990.

8 E. Demaine, M. Hajiaghayi, and K. Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *FOCS*, pages 637–646. IEEE, 2005.

9 R.J Duffin. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, 10(2):303–318, 1965.

10 D. Eppstein. Parallel recognition of series-parallel graphs. *Inf. and Comp.*, 98(1):41–55, 1992.

11 A. Epstein, M. Feldman, and Y. Mansour. Efficient graph topologies in network routing games. *Games and Economic Behavior*, 66(1):115–125, 2009.

12 Fortune, Hopcroft, and Wyllie. The directed subgraph homeomorphism problem. *TCS: Theoretical Computer Science*, 10, 1980.

13 V. Gogate and R. Dechter. A complete anytime algorithm for treewidth. In *UAI*, pages 201–208, 2004.

14 R. Holzman and N. Law-Yone. Network structure and strong equilibrium in route selection games. *Mathematical Social Sciences*, 46(2):193–205, 2003.

15 E. Horvitz, J. Breese, and M. Henrion. Decision theory in expert systems and artificial intelligence. *International journal of approximate reasoning*, 2(3):247–302, 1988.

16 A. Jakoby, M. Liśkiewicz, and R. Reischuk. Space efficient algorithms for directed series–parallel graphs. *Journal of Algorithms*, 60(2):85–114, 2006.

17 T. Johnson, N. Robertson, P. Seymour, and R. Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.

18 T. Johnson, N. Robertson, P. Seymour, and R. Thomas. Excluding a grid minor in planar digraphs. *arXiv:1510.00473*, 2015.

19 K. Kawarabayashi and S. Kreutzer. Towards the graph minor theorems for directed graphs. In *ICALP*, pages 3–10. Springer, 2015.

20 S. Kintali and Q. Zhang. Forbidden directed minors and kelly-width. *arXiv:1308.5170*, 2013.

21 J. Kleinberg and S. Oren. Time-inconsistent planning: a computational problem in behavioral economics. In *EC*, pages 547–564. ACM, 2014.

22 S. Kreutzer. Nowhere crownful classes of directed graphs. In *Encyclopedia of Algorithms*, pages 1416–1419. Springer, 2016.

23 Casimir Kuratowski. Sur le probleme des courbes gauches en topologie. *Fundamenta mathematicae*, 15(1):271–283, 1930.

24 Michael Lampis, Georgia Kaouri, and Valia Mitsou. On the algorithmic effectiveness of digraph decompositions and complexity measures. *Discrete Optimization*, 8(1):129–138, 2011. Parameterized Complexity of Discrete Optimization. `doi:10.1016/j.disopt.2010.03.010`.

25 László Lovász. Graph minor theory. *Bulletin of the American Mathematical Society*, 43(1):75–86, 2006.

26 A. Mackworth. Consistency in networks of relations. In *Readings in AI*, pages 69–78. Tioga Publ. Col., 1981.

27 K. Meer. An extended tree-width notion for directed graphs related to the computation of permanents. *Computer Science–Theory and Applications*, pages 247–260, 2011.

28 R. Meir and D. Parkes. Playing the wrong game: Bounding negative externalities in diverse populations of agents. In *AAMAS'18*, 2018. To appear.

29 I. Milchtaich. Network topology and the efficiency of equilibrium. *GEB*, 57:321–346, 2006.

30 E. Nikolova and N. Stier-Moses. The burden of risk aversion in mean-risk selfish routing. In *EC*, pages 489–506, 2015.

31 Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1(3-4):193–204, 1970.

32 N. Robertson and P. Seymour. Graph minors. II. algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986.

33 N. Robertson and P. Seymour. Graph minors. xx. wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004.

34 M. Rowland, A. Pacchiano, and A. Weller. Conditions beyond treewidth for tightness of higher-order lp relaxations. In *AI and Statistics*, pages 10–18, 2017.

35 M. Safari. D-width: A more natural measure for directed tree width. In *MFCS*, pages 745–756. Springer, 2005.

36 C. Shannon. The synthesis of two-terminal switching circuits. *Bell Labs Technical Journal*, 28(1):59–98, 1949.

37 K. Takamizawa, T. Nishizeki, and N. Saito. Linear-time computability of combinatorial problems on series-parallel graphs. *JACM*, 29(3):623–641, 1982.

38 P. Tang, Y. Teng, Z. Wang, S. Xiao, and Y. Xu. Computational issues in time-inconsistent planning. In *AAAI*, pages 3665–3671, 2017.

39 J. Vygen. NP-completeness of some edge-disjoint paths problems. *Discrete Applied Mathematics*, 61(1):83–90, 1995.

40 K. Wagner. Über eine eigenschaft der ebenen komplexe. *Mathematische Annalen*, 114(1):570–590, 1937.

41 Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, pages 1112–1119, 2014.

The Complexity of Finding Small Separators in Temporal Graphs

Philipp Zschoche
Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany
zschoche@tu-berlin.de

Till Fluschnik[1]
Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany
till.fluschnik@tu-berlin.de

Hendrik Molter[2]
Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany
h.molter@tu-berlin.de

Rolf Niedermeier
Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany
rolf.niedermeier@tu-berlin.de

Abstract

Temporal graphs are graphs with time-stamped edges. We study the problem of finding a small vertex set (the separator) with respect to two designated terminal vertices such that the removal of the set eliminates all temporal paths connecting one terminal to the other. Herein, we consider two models of temporal paths: paths that pass through arbitrarily many edges per time step (non-strict) and paths that pass through at most one edge per time step (strict). Regarding the number of time steps of a temporal graph, we show a complexity dichotomy (NP-hardness versus polynomial-time solvability) for both problem variants. Moreover we prove both problem variants to be NP-complete even on temporal graphs whose underlying graph is planar. We further show that, on temporal graphs with planar underlying graph, if additionally the number of time steps is constant, then the problem variant for strict paths is solvable in quasi-linear time. Finally, we introduce and motivate the notion of a temporal core (vertices whose incident edges change over time). We prove that the non-strict variant is fixed-parameter tractable when parameterized by the size of the temporal core, while the strict variant remains NP-complete, even for constant-size temporal cores.

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

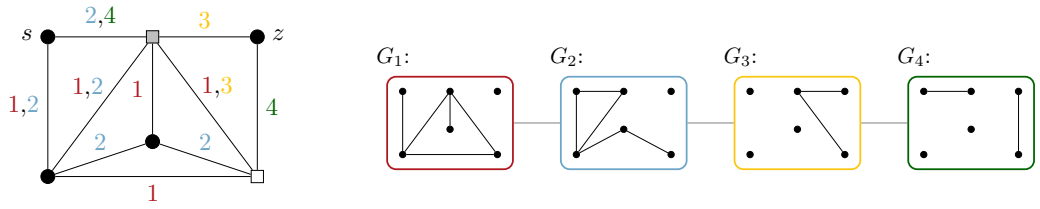
Keywords and phrases (non-)strict temporal paths, temporal core, single-source shortest paths, node multiway cut, length-bounded cuts, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.45

Related Version A full version of the paper is available at `https://arxiv.org/abs/1711.00963`.

Acknowledgements We thank anonymous reviewers for their constructive feedback which helped us to improve the presentation of this work.

[1] Supported by the DFG, project DAMM (NI 369/13) and project TORE (NI 369/18).
[2] Partially supported by the DFG, project MATE (NI 369/17).



(a) A temporal graph $\boldsymbol{G}$. **(b)** Layers of $\boldsymbol{G}$.

Figure 1 Subfigure (a) shows a temporal graph $\boldsymbol{G}$ and subfigure (b) shows its four layers $G_1, \ldots, G_4$. The gray squared vertex forms a strict temporal (s, z)-separator, but no temporal (s, z)-separator. The two squared vertices form a temporal (s, z)-separator.

1 Introduction

In complex network analysis, it is nowadays very common to have access to and process graph data where the interactions among the vertices are time-stamped. When using static graphs as a mathematical model, the dynamics of interactions are not reflected and important information of the data might not be captured. *Temporal graphs* address this issue. A temporal graph is, informally speaking, a graph where the edge set may change over a discrete time interval, while the vertex set remains unchanged. Having the dynamics of interactions represented in the model, it is essential to adapt definitions such as connectivity and paths to respect temporal features. This directly affects the notion of *separators* in the temporal setting. Vertex separators are a fundamental primitive in static network analysis and it is well-known that they can be computed in polynomial time (see, e.g., proof of [1, Theorem 6.8]). In contrast to the static case, Kempe et al. [25] showed that in temporal graphs it is NP-hard to compute minimum separators.

Temporal graphs are well-established in the literature and are also referred to as time-varying [27] and evolving [15] graphs, temporal networks [24, 25, 30], multidimensional networks [8], link streams [26, 36], and edge-scheduled networks [7]. In this work, we use the well-established model in which each edge has a time stamp [8, 24, 3, 22, 25, 30]. Assuming discrete time steps, this is equivalent to a sequence of static graphs over a fixed set of vertices [31]. Formally, we define a temporal graph as follows.

▶ **Definition 1.1** (Temporal Graph)**.** An (undirected) *temporal graph* $\boldsymbol{G} = (V, \boldsymbol{E}, \tau)$ is an ordered triple consisting of a set V of vertices, a set $\boldsymbol{E} \subseteq \binom{V}{2} \times \{1, \ldots, \tau\}$ of *time-edges*, and a maximal time label $\tau \in \mathbb{N}$.

See Figure 1 for an example with $\tau = 4$, that is, a temporal graph with four time steps, also referred to as *layers*. The static graph obtained from a temporal graph $\boldsymbol{G}$ by removing the time stamps from all time-edges we call the *underlying graph* of $\boldsymbol{G}$.

Many real-world applications have temporal graphs as underlying mathematical model. For instance, it is natural to model connections in public transportation networks with temporal graphs. Other examples include information spreading in social networks, communication in social networks, biological pathways, or spread of diseases [24].

A fundamental question in temporal graphs, addressing issues such as connectivity [5, 30], survivability [27], and robustness [34], is whether there is a "time-respecting" path from a

distinguished start vertex s to a distinguished target vertex z.[3] We provide a thorough study of the computational complexity of separating s from z in a given temporal graph.

Moreover, we study two natural restrictions of temporal graphs:

(i) planar temporal graphs and

(ii) temporal graphs with a bounded number of vertices incident to edges that are not permanently existing – these vertices form the so-called *temporal core*.

Both restrictions are naturally motivated by settings e.g. occurring in (hierarchical) traffic networks. We also consider two very similar but still significantly differing temporal path models (both used in the literature), leading to two corresponding models of temporal separation.

Two path models. We start with the introduction of the "non-strict" path model [25]. Given a temporal graph $\boldsymbol{G} = (V, \boldsymbol{E}, \tau)$ with two distinct vertices $s, z \in V$, a *temporal (s,z)-path* of length ℓ in $\boldsymbol{G}$ is a sequence $P = ((\{s = v_0, v_1\}, t_1), (\{v_1, v_2\}, t_2), \ldots, (\{v_{\ell-1}, v_\ell = z\}, t_\ell))$ of time-edges in $\boldsymbol{E}$, where $v_i \neq v_j$ for all $i, j \in \{0, \ldots, \ell\}$ with $i \neq j$ and $t_i \leq t_{i+1}$ for all $i \in \{1, \ldots, \ell - 1\}$. A vertex set S with $S \cap \{s, z\} = \emptyset$ is a *temporal (s,z)-separator* if there is no temporal (s,z)-path in $\boldsymbol{G} - S := (V \setminus S, \{(\{v, w\}, t) \in \boldsymbol{E} \mid v, w \in V \setminus S\}, \tau)$. We are ready to state the central problem of our paper.

Temporal (s,z)-Separation
Input: A temporal graph $\boldsymbol{G} = (V, \boldsymbol{E}, \tau)$, two distinct vertices $s, z \in V$, and $k \in \mathbb{N}$.
Question: Does $\boldsymbol{G}$ admit a temporal (s,z)-separator of size at most k?

Our second path model is the "strict" variant. A temporal (s,z)-path P is called *strict* if $t_i < t_{i+1}$ for all $i \in \{1, \ldots, \ell - 1\}$. In the literature, strict temporal paths are also known as journeys [3, 2, 31, 30].[4] A vertex set S is a *strict temporal (s,z)-separator* if there is no strict temporal (s,z)-path in $\boldsymbol{G} - S$. Thus, our second main problem, Strict Temporal (s,z)-Separation, is defined in complete analogy to Temporal (s,z)-Separation, just replacing (non-strict) temporal separators by strict ones.

While the strict version of temporal separation immediately appears as natural, the non-strict variant can be viewed as a more conservative version of the problem. For instance, in a disease-spreading scenario the spreading speed might be unclear. To ensure containment of the spreading by separating patient zero (s) from a certain target (z), a temporal (s,z)-separator might be the safer choice.

Main results. Table 1 provides an overview on our results.[5]

A central contribution is to prove that both Temporal (s,z)-Separation and Strict Temporal (s,z)-Separation are NP-complete for all $\tau \geq 2$ and $\tau \geq 5$, respectively, strengthening a result by Kempe et al. [25] (they show NP-hardness of both variants for all $\tau \geq 12$). For Temporal (s,z)-Separation, our hardness result is already tight.[6] For the strict variant, we identify a dichotomy in the computational complexity by proving

[3] In the literature the sink is usually denoted by t. To be consistent with Michail [31] we use z instead as we reserve t to refer to points in time.

[4] We also refer to Himmel [21] for a thorough discussion and comparison of temporal path concepts.

[5] Due to the space constraints, several details and proofs (marked with $\star$) are deferred to a long version of this paper, see e.g. `https://arxiv.org/abs/1711.00963`.

[6] Temporal (s,z)-Separation with $\tau = 1$ is equivalent to (s,z)-Separation on static graphs.

Table 1 Overview on our results. Herein, NP-c. abbreviates NP-complete, n and $\boldsymbol{m}$ denote the number of vertices and time-edges, respectively, $G_{\downarrow}$ refers to the underlying graph of an input temporal graph. [a] (Thm. 3.1; W[1]-hard wrt. k) [b] (Thm. 3.2) [c] (Cor. 4.3) [d] (Prop. 4.4) [e] (Thm. 5.2)

	General (Section 3)		Planar $G_{\downarrow}$ (Section 4)		Temporal core (Section 5)
(s,z)-Separation	$2 \leq \tau \leq 4$	$5 \leq \tau$	τ unbounded	τ constant	constant size
Temporal	NP-complete[a]		NP-c.[c]	*open*	$n^{O(1)} + O(\boldsymbol{m} \log \boldsymbol{m})$ [e]
Strict Temporal	$\mathcal{O}(k \cdot \boldsymbol{m})$ [b]	NP-c.[a]	NP-c.[c]	$\mathcal{O}(\boldsymbol{m} \log \boldsymbol{m})$ [d]	NP-complete[a]

polynomial-time solvability of Strict Temporal (s,z)-Separation for $\tau \leq 4$. Moreover, we prove that both problems remain NP-complete on temporal graphs that have an underlying graph that is planar.

We introduce the notion of temporal cores in temporal graphs. Informally, the temporal core of a temporal graph is the set of vertices whose edge-incidences change over time. We prove that Temporal (s,z)-Separation is fixed-parameter tractable (FPT) when parameterized by the size of the temporal core, while Strict Temporal (s,z)-Separation remains NP-complete even if the temporal core is empty.

A particular aspect of our results is that they demonstrate that the choice of the model (strict versus non-strict) for a problem can have a crucial impact on the computational complexity of said problem. This contrasts with wide parts of the literature where both models were used without discussing the subtle but crucial differences in computational complexity.

Technical contributions. To show the polynomial-time solvability of Strict Temporal (s,z)-Separation for $\tau \leq 4$, we prove that a classic separator result of Lovász et al. [28] translates to the strict temporal setting. This is surprising since many other results about separators in the static case do not apply in the temporal case. In this context, we also develop a linear-time algorithm for Single-Source Shortest Strict Temporal Paths, improving the running time of the best known algorithm due to Wu et al. [37] by a logarithmic factor.

We settle the complexity of Length-Bounded (s,z)-Separation on planar graphs by showing its NP-hardness, which was left unanswered by Fluschnik et al. [17] and promises to be a valuable intermediate problem for proving hardness results. In the hardness reduction for Length-Bounded (s,z)-Separation we introduce a grid-like, planarity-preserving vertex gadget that is generally useful to replace "twin" vertices which in many cases are not planarity-preserving and which are often used to model weights.

While showing that Temporal (s,z)-Separation is fixed-parameter tractable when parameterized by the size of the temporal core, we employ a case distinction on the size of the temporal core, and show that in the non-trivial case we can reduce the problem to Node Multiway Cut. We identify an "above lower bound parameter" for Node Multiway Cut that is suitable to lower-bound the size of the temporal core, thereby making it possible to exploit a fixed-parameter tractability result due to Cygan et al. [12].

Related work. Our most important reference is the work of Kempe et al. [25] who proved that Temporal (s,z)-Separation is NP-hard. In contrast, Berman [7] proved that computing temporal (s,z)-cuts (edge deletion instead of vertex deletion) is polynomial-time

solvable. In the context of survivability of temporal graphs, Liang and Modiano [27] studied cuts where an edge deletion only lasts for δ consecutive time stamps. Moreover, they studied a temporal maximum flow defined as the maximum number of sets of journeys where each two journeys in a set do not use a temporal edge within some δ time steps. A different notion of temporal flows on temporal graphs was introduced by Akrida et al. [2]. They showed how to compute in polynomial time the maximum amount of flow passing from a source vertex s to a sink vertex z until a given point in time.

The vertex-variant of Menger's Theorem [29] states that the maximum number of vertex-disjoint paths from s to z equals the size of a minimum-cardinality (s, z)-separator. In static graphs, Menger's Theorem allows for finding a minimum-cardinality (s, z)-separator via maximum flow computations. However, Berman [7] proved that the vertex-variant of an analogue to Menger's Theorem for temporal graphs, asking for the maximum number of (strict) temporal paths instead, does not hold. Kempe et al. [25] proved that the vertex-variant of the former analogue to Menger's Theorem holds true if the underlying graph excludes a fixed minor. Mertzios et al. [30] proved another analogue of Menger's Theorem: the maximum number of strict temporal (s, z)-path which never leave the same vertex at the same time equals the minimum number of node departure times needed to separate s from z, where a node departure time (v, t) is the vertex v at time point t.

Michail and Spirakis [32] introduced the time-analogue of the famous Traveling Salesperson problem and studied the problem on temporal graphs of dynamic diameter $d \in \mathbb{N}$, that is, informally speaking, on temporal graphs where every two vertices can reach each other in at most d time steps at any time. Erlebach et al. [14] studied the same problem on temporal graphs where the underlying graph has bounded degree, bounded treewidth, or is planar. Additionally, they introduced a class of temporal graphs with regularly present edges, that is, temporal graphs where each edge is associated with two integers upper- and lower-bounding consecutive time steps of edge absence. Axiotis and Fotakis [5] studied the problem of finding the smallest temporal subgraph of a temporal graph such that single-source temporal connectivity is preserved on temporal graphs where the underlying graph has bounded treewidth. In companion work, we recently studied the computational complexity of (non-strict) temporal separation on several other restricted temporal graphs [18].

2 Preliminaries

Let $\mathbb{N}$ denote the natural numbers without zero. For $n \in \mathbb{N}$, we use $[n] := [1, n] = \{1, \ldots, n\}$.

Static graphs. We use basic notations from (static) graph theory [13]. Let $G = (V, E)$ be an *undirected, simple graph*. We use $V(G)$ and $E(G)$ to denote the set of vertices and set of edges of G, respectively. We denote by $G - V' := (V \setminus V', \{\{v, w\} \in E \mid v, w \in V \setminus V'\})$ the graph G without the vertices in $V' \subseteq V$. For $V' \subseteq V$, $G[V'] := G - (V \setminus V')$ denotes the *induced subgraph* of G by V'. A *path* of length ℓ is sequence of edges $P = (\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_\ell, v_{\ell+1}\})$ where $v_i \neq v_j$ for all $i, j \in [\ell + 1]$ with $i \neq j$. We set $V(P) = \{v_1, v_2, \ldots, v_{\ell+1}\}$. Path P is an (s, z)-*path* if $s = v_1$ and $z = v_{\ell+1}$. A set $S \subseteq V \setminus \{s, z\}$ of vertices is an (s, z)-*separator* if there is no (s, z)-path in $G - S$.

Temporal graphs. Let $\boldsymbol{G} = (V, \boldsymbol{E}, \tau)$ be a temporal graph. The graph $G_i(\boldsymbol{G}) = (V, E_i(\boldsymbol{G}))$ is called *layer* i of the temporal graph $\boldsymbol{G} = (V, \boldsymbol{E}, \tau)$ where $\{v, w\} \in E_i(\boldsymbol{G}) \Leftrightarrow (\{v, w\}, i) \in \boldsymbol{E}$. The *underlying graph* $G_\downarrow(\boldsymbol{G})$ of a temporal graph $\boldsymbol{G} = (V, \boldsymbol{E}, \tau)$ is defined as $G_\downarrow(\boldsymbol{G}) := (V, E_\downarrow(\boldsymbol{G}))$, where $E_\downarrow(\boldsymbol{G}) = \{e \mid (e, t) \in \boldsymbol{E}\}$. (We write G_i, E_i, $G_\downarrow$, and $E_\downarrow$ for short

if $\boldsymbol{G}$ is clear from the context.) For $X \subseteq V$ we define the *induced temporal subgraph* of X by $\boldsymbol{G}[X] := (X, \{(\{v, w\}, t) \in \boldsymbol{E} \mid v, w \in X\}, \tau)$. We say that $\boldsymbol{G}$ is *connected* if its underlying graph $G_{\downarrow}$ is connected. For surveys concerning temporal graphs we refer to [9, 31, 24, 26, 23].

Strict and non-strict temporal separators. Throughout the paper we assume that the underlying graph of the temporal input graph $\boldsymbol{G}$ is connected and that there is no time-edge between s and z. Furthermore, in accordance with Wu et al. [37] we assume that the time-edge set $\boldsymbol{E}$ is ordered by ascending time stamps. Moreover, we can assume that the number of layers is at most the number of time-edges:

▶ **Lemma 2.1 (⋆).** *Let $\mathcal{I} = (\boldsymbol{G} = (V, \boldsymbol{E}, \tau), s, z, k)$ be an instance of* (Strict) Temporal (s, z)-Separation. *There is an algorithm which computes in $\mathcal{O}(|\boldsymbol{E}|)$ time an instance $\mathcal{I}' = (\boldsymbol{G}' = (V, \boldsymbol{E}', \tau'), s, z, k)$ of* (Strict) Temporal (s, z)-Separation *which is equivalent to $\mathcal{I}$, where $\tau' \leq |\boldsymbol{E}'|$.*

Regarding our two models, we have the following connection:

▶ **Lemma 2.2 (⋆).** *There is a linear-time computable many-one reduction from* Strict Temporal (s, z)-Separation *to* Temporal (s, z)-Separation *that maps any instance $(\boldsymbol{G} = (V, \boldsymbol{E}, \tau), s, z, k)$ to an instance $(\boldsymbol{G}' = (V', \boldsymbol{E}', \tau'), s, z, k')$ with $k' = k$ and $\tau' = 2 \cdot \tau$.*

3 Hardness Dichotomy Regarding the Number of Layers

In this section we settle the complexity dichotomy of both Temporal (s, z)-Separation and Strict Temporal (s, z)-Separation regarding the number τ of time steps. We observe that both problems are strongly related to the following NP-complete [10, 35] problem:

Length-Bounded (s, z)-Separation (LBS)
Input: An undirected graph $G = (V, E)$, distinct vertices $s, z \in V$, and $k, \ell \in \mathbb{N}$.
Question: Is there a subset $S \subseteq V \setminus \{s, z\}$ such that $|S| \leq k$ and there is no (s, z)-path in $G - S$ of length at most ℓ?

Length-Bounded (s, z)-Separation is NP-complete even if the lower bound ℓ for the path length is five [6] and W[1]-hard with respect to the postulated separator size [20]. We obtain the following, improving a result by Kempe et al. [25] who showed NP-completeness of Temporal (s, z)-Separation and Strict Temporal (s, z)-Separation for all $\tau \geq 12$.

▶ **Theorem 3.1 (⋆).** Temporal (s, z)-Separation *is* NP*-complete for every maximum label $\tau \geq 2$ and* Strict Temporal (s, z)-Separation *is* NP*-complete for every $\tau \geq 5$. Moreover, both problems are* W[1]*-hard when parameterized by the solution size k.*

We only present the construction of the NP-hardness reduction for Temporal (s, z)-Separation, which is inspired by Baier et al. [6], and postpone the rest to the long version.

Proof (Construction). To show NP-completeness of Temporal (s, z)-Separation for $\tau = 2$ we present a reduction from the Vertex Cover problem where, given a graph $G = (V, E)$ and an integer k, the task is to determine whether there exists a set $V' \subseteq V$ of size at most k such that $G - V'$ does not contain any edge. Let $(G = (V, E), k)$ be an instance of Vertex Cover. We say that $V' \subseteq V$ is a *vertex cover* in G of size k if $|V'| = k$ and V' is a solution to $(G = (V, E), k)$. We refine the gadget of Baier et al. [6, Theorem 3.9] and reduce from Vertex Cover to Temporal (s, z)-Separation. Let $\mathcal{I} := (G = (V, E), k)$ be a Vertex Cover instance and $n := |V|$. We construct a Temporal (s, z)-Separation

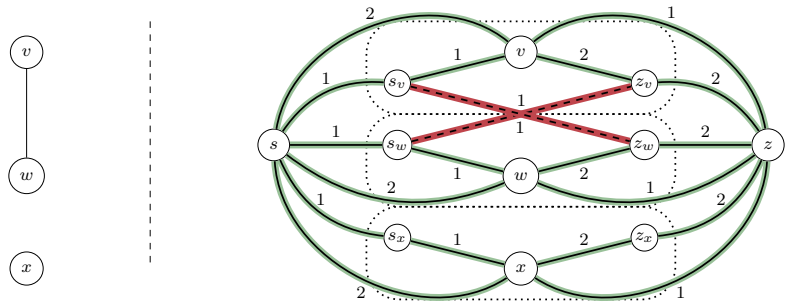


Figure 2 The Vertex Cover instance $(G, 1)$ (left) and the corresponding Temporal (s, z)-Separation instance from the reduction of Theorem 3.1 (right). The edge-edges are dashed (red), the vertex-edges are solid (green), and the vertex gadgets are in dotted boxes.

instance $\mathcal{I}' := (\boldsymbol{G}' = (V', \boldsymbol{E}', 2), s, z, n + k)$, where $V' := V \cup \{s_v, t_v \mid v \in V\} \cup \{s, z\}$ are the vertices and the time-edges are defined as

$$\boldsymbol{E}' := \overbrace{\{(\{s, s_v\}, 1), (\{s_v, v\}, 1), (\{v, z_v\}, 2), (\{z_v, z\}, 2), (\{s, v\}, 2), (\{v, z\}, 1) \mid v \in V\}}^{\text{vertex-edges}} \cup \underbrace{\{(\{s_v, z_w\}, 1), (\{s_w, z_v\}, 1) \mid \{v, w\} \in E\}}_{\text{edge-edges}}.$$

Note that $|V'| = 3 \cdot n + 2$, $|\boldsymbol{E}'| = 6 \cdot |V'| + 2 \cdot |E|$, and $\mathcal{I}'$ can be computed in polynomial time. For each vertex $v \in V$ there is a *vertex gadget* which consists of three vertices s_v, v, z_v and six vertex-edges. In addition, for each edge $\{v, w\} \in E$ there is an *edge gadget* which consists of two edge-edges $\{s_v, z_w\}$ and $\{z_v, s_w\}$. See Figure 2 for an example. ◀

In the remainder of this section we prove that the bound on τ is tight in the strict case (for the non-strict case the tightness is obvious). This is the first case where we can observe a significant difference between the strict and the non-strict variant of our separation problem.

▶ **Theorem 3.2.** Strict Temporal (s, z)-Separation *for maximum label $\tau \leq 4$ can be solved in $\mathcal{O}(k \cdot |\boldsymbol{E}|)$ time, where k is the solution size.*

As a subroutine hidden in several of our algorithms, we need to solve the Single-Source Shortest Strict Temporal Paths problem on temporal graphs: find shortest strict paths from a source vertex s to all other vertices in the temporal graph. Herein, we say that a strict temporal (s, z)-path is *shortest* if there is no strict temporal (s, z)-path of length $\ell' < \ell$. Indeed, we provide a linear-time algorithm for this. We believe this to be of independent interest; it improves (with few adaptations to the model; for details we refer to the long version) previous results by Wu et al. [37], but in contrast to the algorithm of Wu et al. [37] our subroutine cannot be adjusted to the non-strict case.

▶ **Proposition 3.3** ($\star$). Single-Source Shortest Strict Temporal Paths *is solvable in $\Theta(|\boldsymbol{E}|)$ time.*

Our algorithm behind Theorem 3.2 executes the following steps:

1. As a preprocessing step, remove unnecessary time-edges and vertices from the graph.
2. Compute an auxiliary graph called *directed path cover graph* of the temporal graph.
3. Compute a separator for the directed path cover graph.

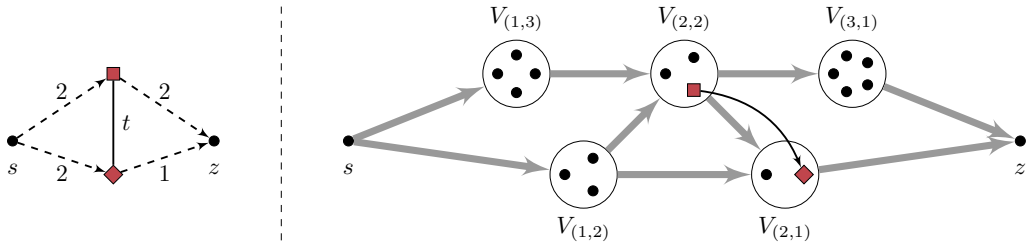


Figure 3 The left side depicts an excerpt of a reduced temporal graph with maximum time-edge label $\tau = 4$. Dashed arcs labeled with a number x indicate a shortest strict temporal path of length x. The right side depicts the directed path cover graph D from s to z of the reduced temporal graph. A gray arc from vertex set $V_{(i,j)}$ to vertex set $V_{(i',j')}$ denotes that for two vertices $v \in V_{(i,j)}$ and $w \in V_{(i',j')}$ there can be an arc from v to w in D. Take as an example the square-shaped vertex in $V_{(2,2)}$ and the diamond-shaped vertex in $V_{(2,1)}$.

In the following, we explain each of the steps in more detail.

The preprocessing *reduces* the temporal graph such that it has the following properties. A temporal graph $\boldsymbol{G} = (V, \boldsymbol{E}, \tau)$ with two distinct vertices $s, z \in V$ is *reduced* if

(i) the underlying graph $\boldsymbol{G}_{\downarrow}$ is connected,
(ii) for each time-edge $e \in \boldsymbol{E}$ there is a strict temporal (s, z)-path which contains e, and
(iii) there is no strict temporal (s, z)-path of length at most two in $\boldsymbol{G}$.

This preprocessing step can be performed in polynomial time:

▶ **Lemma 3.4** (⋆). *Let $\mathcal{I} = (\boldsymbol{G} = (V, \boldsymbol{E}, \tau), s, z, k)$ be an instance of* Strict Temporal (s, z)-Separation. *In $\mathcal{O}(k \cdot |\boldsymbol{E}|)$ time, one can either decide $\mathcal{I}$ or construct an instance $\mathcal{I}' = (\boldsymbol{G}' = (V', \boldsymbol{E}', \tau), s, z, k')$ of* Strict Temporal (s, z)-Separation *such that $\mathcal{I}'$ is equivalent to $\mathcal{I}$, $\boldsymbol{G}'$ is reduced, $|V'| \leq |V|$, $|\boldsymbol{E}'| \leq |\boldsymbol{E}|$, and $k' \leq k$.*

Lovász et al. [28] showed that the minimum size of an (s, z)-separator for paths of length at most four in a graph is equal to the number of vertex-disjoint (s, z)-paths of length at most four in a graph. We adjust their idea to strict temporal paths on temporal graphs. The proof of Lovász et al. [28] implicitly relies on the transitivity of connectivity in static graphs. This does not hold for temporal graphs; hence, we have to extend their result to the temporal case. To this end, we define a directed auxiliary graph.

▶ **Definition 3.5** (Directed Path Cover Graph). Let $\boldsymbol{G} = (V, \boldsymbol{E}, \tau = 4)$ be a reduced temporal graph with $s, z \in V$. The *directed path cover graph* from s to z of $\boldsymbol{G}$ is a directed graph $D = (V, \vec{E})$ such that $(v, w) \in \vec{E}$ if and only if

(i) $v, w \in V$,
(ii) $(\{v, w\}, t) \in \boldsymbol{E}$ for some $t \in [\tau]$, and
(iii) $v \in V_{(i,j)}$ and $w \in V_{(i',j')}$ such that $i < i'$, $v \in V_{(2,2)}$ and $w \in V_{(2,1)}$, $v = s$ and $w \in V_{(1,j)}$, or $w = z$ and $v \in V_{(i,1)}$ for some $i, j \in \{2, 3\}$.

Herein, a vertex $x \in V$ is in the set $V_{(i,j)}$ if the shortest strict temporal (s, x)-path is of length i and the shortest strict temporal (x, z)-path is of length j.

Figure 3 depicts a generic directed path cover graph of a reduced temporal graph with $\tau = 4$. Note that due to the definition of reduced temporal graphs, one can prove that the set $V_{(1,1)}$ is always empty, and hence not depicted in Figure 3. This is a crucial property that allows us to prove the following.

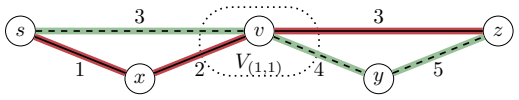


Figure 4 A reduced temporal graph with maximum label $\tau = 5$ where the vertex set $V_{(1,1)}$ of the directed path cover graph is not empty. The solid (red) and dashed (green) edges are strict temporal paths and show that edges $(\{s, v\}, 3)$ and $(\{v, z\}, 3)$ are not removed when the graph is reduced. Furthermore, v is not removed since $((\{s, v\}, 3), (\{v, z\}, 3))$ is not a strict temporal path.

▶ **Lemma 3.6** ($\star$). *Let $\boldsymbol{G} = (V, \boldsymbol{E}, \tau = 4)$ be a reduced temporal graph with $s, z \in V$. Then the directed path cover graph D from s to z of $\boldsymbol{G}$ can be computed in $\mathcal{O}(|\boldsymbol{E}|)$ time and $S \subseteq V \setminus \{s, z\}$ is a strict temporal (s, z)-separator in $\boldsymbol{G}$ if and only if S is an (s, z)-separator in D.*

Figure 4 shows that if $\tau = 5$, then we can construct a reduced temporal graph where the set $V_{(1,1)}$ is not empty. This indicates why our algorithm fails for $\tau = 5$.

Finally, with Lemmata 3.4 and 3.6 we can prove Theorem 3.2.

Proof of Theorem 3.2. Let $\mathcal{I} := (\boldsymbol{G} = (V, \boldsymbol{E}, \tau = 4), s, z, k)$ be an instance of Strict Temporal (s, z)-Separation. First, apply Lemma 3.4 in $\mathcal{O}(k \cdot |\boldsymbol{E}|)$ time to either decide $\mathcal{I}$ or to obtain an instance $\mathcal{I}' = (\boldsymbol{G}' = (V', \boldsymbol{E}', \tau), s, z, k')$ of Strict Temporal (s, z)-Separation. In the second case, compute the directed path cover graph D of $\boldsymbol{G}'$ from s to z in $\mathcal{O}(|\boldsymbol{E}'|)$ time (by Lemma 3.6). Next, check whether D has an (s, z)-separator of size at most k' in $\mathcal{O}(k' \cdot |\boldsymbol{E}'|)$ time by a folklore result [19]. By Lemma 3.6, D has an (s, z)-separator of size k' if and only if $\boldsymbol{G}'$ has a strict temporal (s, z)-separator of size k'. Since by Lemma 3.4 we have that $\boldsymbol{G}'$ is reduced, $|V'| \leq |V|$, $|\boldsymbol{E}'| \leq |\boldsymbol{E}|$, and $k' \leq k$, the overall running time is $\mathcal{O}(k \cdot |\boldsymbol{E}|)$. ◀

4 On Temporal Graphs with Planar Underlying Graph

In this section, we study our problems on *planar temporal graphs*, that is, temporal graphs that have a planar underlying graph. We show that both Temporal (s, z)-Separation and Strict Temporal (s, z)-Separation remain NP-complete on planar temporal graphs. On the positive side, we show that on planar temporal graphs with a constant number of layers, Strict Temporal (s, z)-Separation can be solved in $\mathcal{O}(|\boldsymbol{E}| \cdot \log |\boldsymbol{E}|)$ time.

In order to prove our hardness results, we first prove NP-hardness for Length-Bounded (s, z)-Separation on planar graphs – a result which we consider to be of independent interest; note that NP-completeness on planar graphs was only known for the edge-deletion variant of Length-Bounded (s, z)-Separation on undirected graphs [17] and weighted directed graphs [33].

▶ **Theorem 4.1.** Length-Bounded (s, z)-Separation *on planar graphs is* NP*-hard.*

Proof. We give a many-one reduction from the NP-complete [17] edge-weighted variant of Length-Bounded (s, z)-Cut, referred to as Planar Length-Bounded (s, z)-Cut, where the input graph $G = (V, E)$ is planar, has edge costs $c : E \to \{1, k + 1\}$, has maximum degree $\Delta = 6$, the degree of s and z is three, and s and z are incident to the outer face. Since the maximum degree is constant, one can replace a vertex with a planar grid-like gadget.

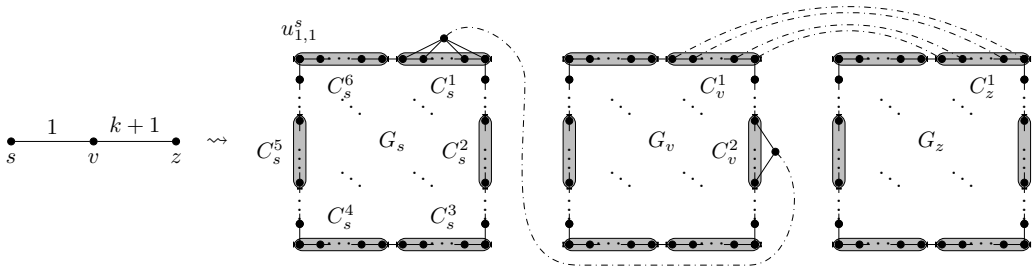


Figure 5 A simple planar graph G (left) with edge costs (above edges) and the obtained graph G' in the reduction from Theorem 4.1. The connector sets are highlighted in gray. The edge-gadgets are indicated by dash-dotted lines.

Let $\mathcal{I} := (G = (V, E, c), s, z, \ell, k)$ be an instance of Planar Length-Bounded (s, z)-Cut, and we assume k to be even[7]. We construct an instance $\mathcal{I}' := (G', s', z', \ell', k)$ of Length-Bounded (s, z)-Separation as follows (refer to Figure 5 for an illustration).

Construction. For each vertex $v \in V$, we introduce a *vertex-gadget* G_v which is a grid of size $(2k + 2) \times (2k + 2)$, that is, a graph with vertex set $\{u^v_{i,j} \mid i, j \in [2k + 2]\}$ and edge set $\{\{u^v_{i,j}, u^v_{i',j'}\} \mid |i - i'| + |j - j'| = 1\}$. There are six pairwise disjoint subsets $C^1_v, \ldots, C^6_v \subseteq V(G_v)$ of size $k + 1$ that we refer to as *connector sets*. As we fix an orientation of G_v such that $u^v_{1,1}$ is in the top-left, there are two connector sets on the top of G_v, two on the bottom of G_v, one on the left of G_v, and one on the right of G_v. Formally, $C^1_v = \{u^v_{1,k+2}, \ldots, u^v_{1,2k+2}\}$, $C^2_v = \{u^v_{k/2,2k+2}, \ldots, u^v_{3k/2,2k+2}\}$, $C^3_v = \{u^v_{2k+2,k+2}, \ldots, u^v_{2k+2,2k+2}\}$, $C^4_v = \{u^v_{2k+2,1}, \ldots, u^v_{2k+2,k+1}\}$, $C^5_v = \{u^v_{k/2,1}, \ldots, u^v_{3k/2,1}\}$, and $C^6_v = \{u^v_{1,1}, \ldots, u^v_{1,k+1}\}$.

Note that all (x, y)-paths are of length at most $k' := (2k + 2)^2 - 1$, for all $x, y \in V(G_v)$, because there are only $(2k + 2)^2$ vertices in $V(G_v)$.

Let $\phi(G)$ be a plane embedding of G. We say that an edge e incident with vertex $v \in V$ is *at position i on v* if e is the ith edge incident with v when counted clockwise with respect to $\phi(G)$.

For each edge $e = \{v, w\}$, we introduce an *edge-gadget* G_e that differs on the weight of e, as follows. Let e be at position $i \in \{1, \ldots, 6\}$ on v and at position $j \in \{1, \ldots, 6\}$ on w.

If $c(e) = 1$, then G_e is constructed as follows. Add a path consisting of $(\ell + 1) \cdot k' - 1$ vertices and connect one endpoint with each vertex in C^i_v by an edge and connect the other endpoint with each vertex in C^j_w by an edge.

If $c(e) = k + 1$, then G_e is constructed as follows. We introduce a planar matching between the vertices in C^i_v and C^j_w. That is, for instance, we connect vertex $u^v_{1,k+2+p}$ with vertex $u^w_{1,2k+2-p}$ for each $p \in \{0, \ldots, k\}$, if $i = j = 1$, or we connect vertex $u^v_{1,1+p}$ with vertex $u^w_{2k+2,3k/2-p}$ for each $p \in \{0, \ldots, k\}$, if $i = 6$ and $j = 2$ (we omit the remaining cases). Then, replace each edge by a path of length at least $(\ell + 1) \cdot k' + 1$ where its endpoints are identified with the endpoints of the replaced edge. Hence, a path between two vertex-gadgets has length at least $(\ell + 1) \cdot k' + 1$.

Next, we choose connector sets $C^{i'}_s$ and $C^{j'}_z$ such that no vertex $v \in C^{i'}_s \cup C^{j'}_z$ is adjacent to a vertex from an edge-gadget. Such i' and j' always exist because the degrees of s and z are both three. Now, we add two special vertices s' and z' and edges between s' and each vertex in $C^{i'}_s$, as well as between z' and each vertex in $C^{j'}_z$.

[7] If k is odd, since s and z are incident to the outer face, then we can add a path of length $\ell - 1$ with endpoints s and z and set the budget for edge deletions to $k + 1$.

Finally, we set $\ell' := 2 + (\ell+1) \cdot k' + \ell\,((\ell+1) \cdot k' + 1)$. Note that G' can be computed in polynomial time. Moreover, one can observe that G' is planar by obtaining an embedding from ϕ. This concludes the description of the construction.

Correctness. We claim that $\mathcal{I}$ is a yes-instance if and only if $\mathcal{I}'$ is a yes-instance.

$\Rightarrow$: Let $\mathcal{I}$ be a yes-instance. Thus, there is a solution $C \subset E$ with $c(C) \leq k$ such that there is no (s,z)-path of length at most ℓ in $G - C$. We construct a set $S \subset V(G')$ of size at most k by taking for each $\{v,w\} \in C$ one arbitrary vertex from the edge-gadget $G_{\{v,w\}}$ into S. Note that since $c(C) \leq k$, each edge in C is of cost one.

Assume towards a contradiction that there is a shortest (s',z')-path P' of length at most ℓ' in $G' - S$. Since a path between two vertex-gadgets has length at least $(\ell+1) \cdot k' + 1$, we know that P' goes through at most ℓ edge-gadgets. Otherwise P' would be of length at least $2 + (\ell+1) \cdot [(\ell+1) \cdot k' + 1] = 2 + (\ell+1) \cdot k' + \ell \cdot [(\ell+1) \cdot k' + 1] + 1 = \ell' + 1$. Now, we reconstruct an (s,z)-path P in G corresponding to P' by taking an edge $e \in E$ into P if P' goes through the edge-gadget G_e. Hence, the length of P is at most ℓ. This contradicts that there is no (s,z)-path of length at most ℓ in $G - C$. Consequently, there is no (s',z')-path of length at most ℓ' in $G' - S$ and $\mathcal{I}'$ is a yes-instance.

$\Leftarrow$: Let $\mathcal{I}'$ be a yes-instance. Thus, there is a solution $S \subseteq V(G')$ of minimum size (at most k) such that there is no (s',z')-path of length at most ℓ' in $G' - S$. Since S is of minimum size, it follows from the following claim that $V(G_v) \cap S = \emptyset$ for all $v \in V$.

▶ **Claim 4.2.** *Let G_v be a vertex-gadget and $i,j \in \{1,\ldots,6\}$ with $i \neq j$. Then, for each vertex set $S \subseteq V(G_v)$ of size at most k it holds that there are $v_1 \in C_v^i \setminus S$ and $v_2 \in C_v^j \setminus S$ such that there is a (v_1,v_2)-path of length at most k' in $G_v - S$.*

Proof of Claim 4.2. Let C_v^i, C_v^j two connector sets of a vertex-gadget G_v, where $i,j \in \{1,\ldots,6\}$ and $i \neq j$. We add vertices a and b and edges $\{a,a'\}$ and $\{b,b'\}$ to G_v, where $a' \in C_v^i$ and $b' \in C_v^j$. There are $\binom{6}{2}$ different cases in which $i \neq j$. It is not difficult to see that in each case there are $k+1$ vertex-disjoint (a,b)-paths. The claim then follows by Menger's Theorem [29]. ◀

Note that by minimality of S, it holds that $V(G_e) \cap S = \emptyset$ for all $e \in E$ with $c(e) = k+1$. We construct an edge set $C \subseteq E$ of cost at most k by taking $\{v,w\} \in E$ into C if there is a $y \in V(G_{\{v,w\}}) \cap S$.

Assume towards a contradiction that there is a shortest (s,z)-path P of length at most ℓ in $G - C$. We reconstruct an (s',z')-path P' in G' which corresponds to P as follows. First, we take an edge $\{s',v\} \in E(G')$ such that $v \in C_s^{i'} \setminus S$. Such a v always exists, because $|C_s^{i'}| = k+1$ and $|S| \leq k$. Let $\{s,w\} \in E$ be the first edge of P and at position i on w. Then we add a (v,v')-path P_s in $G_s - S$, such that $v' \in C_s^i \setminus S$. Due to Claim 4.2, such a (v,v')-path P_s always exists in $G_s - S$ and is of length at most k'.

We take an edge-gadget G_e into P' if e is in P. Recall, that an edge-gadget is a path of length $(\ell+1) \cdot k' + 1$. Due to Claim 4.2, we can connect the edge-gadgets $G_{\{v_1,v_2\}}, G_{\{v_2,v_3\}}$ of two consecutive edges $\{v_1,v_2\},\{v_2,v_3\}$ in P by a path of length at most k' in G_{v_2}. Let $\{v_p,z\}$ be the last edge in P, be at position j on z, $v \in C_z^j$, and $v' \in C_z^{j'}$. We add a (v,v')-path of length k' in $G_z - S$ (Claim 4.2). Note that P' visits at most $\ell+1$ vertex-gadgets and ℓ edge-gadgets. The length of P' is at most $2 + (\ell+1) \cdot k' + \ell\,[(\ell+1) \cdot k' + 1] = \ell$. This contradicts that S forms a solution for $\mathcal{I}'$.

It follows that there is no (s,z)-path of length at most ℓ in $G - C$ and $\mathcal{I}$ is a yes-instance. ◀

From the proofs of Theorem 3.1 and Lemma 2.2 (planarity-preserving reductions for the underlying graph), together with Theorem 4.1 we get the following:

▶ **Corollary 4.3.** *Both* Temporal (s,z)-Separation *and* Strict Temporal (s,z)-Separation *on planar temporal graphs are* NP*-complete.*

In contrast to the case of general temporal graphs, Strict Temporal (s,z)-Separation on planar temporal graphs is efficiently solvable if the maximum label τ is any constant. To this end, we employ the optimization variant of Courcelle's Theorem [4, 11].

▶ **Proposition 4.4** ($\star$)**.** Strict Temporal (s,z)-Separation *on planar temporal graphs can be solved in* $\mathcal{O}(|\boldsymbol{E}| \cdot \log |\boldsymbol{E}|)$ *time, if the maximum label* τ *is constant.*

Due to space constrains, we only sketch how one can develop MSO formulas over temporal graphs and postpone the full proof to the long version.

Proof (Sketch). Let $\mathcal{I} = (\boldsymbol{G} = (V, \boldsymbol{E}, \tau), s, z, k)$ be an instance of Strict Temporal (s,z)-Separation, where the underlying graph $G_{\downarrow}$ of $\boldsymbol{G}$ is planar. We define the edge-labeled graph $L(\boldsymbol{G})$ to be $G_{\downarrow}$ with the edge-labeling $\omega : E(G_{\downarrow}) \to [2^{\tau}-1]$ with $\omega(\{v,w\}) = \sum_{i=1}^{\tau} \mathbb{1}_{\{v,w\}\in E_i} \cdot 2^{i-1}$, where $\mathbb{1}_{\{v,w\}\in E_i} = 1$ if and only if $(\{v,w\}, i) \in \boldsymbol{E}$, and 0 otherwise. Observe that in binary representation, the i-th bit of $\omega(\{v,w\})$ is 1 if and only if $\{v,w\}$ exists at time point i.

We define the optimization variant of Strict Temporal (s,z)-Separation in MSO on $L(\boldsymbol{G})$. First, the MSO formula $\mathrm{layer}(e,t) := \bigvee_{i=1}^{\tau} \bigvee_{j\in\sigma(i,2^{\tau}-1)} \big(t = i \ \wedge \omega(e) = j\big)$ checks whether an edge e is present in the layer t, where $\sigma(i,j) := \{x \in [j] \mid i\text{-th bit of } x \text{ is } 1\}$. Second, we can write an MSO formula $\mathrm{tempadj}(v,w,t) := \exists_{e\in E}\big(\mathrm{inc}(e,v) \wedge \mathrm{inc}(e,w) \wedge \mathrm{layer}(e,t)\big)$ to determine whether two vertices v and w are adjacent at time point t. Third, there is an MSO formula

$$\mathrm{path}(S) := \exists_{x_1,\ldots,x_{\tau+1}\in V\setminus S}\Big(x_1 = s \wedge x_{\tau+1} = z \wedge \bigwedge_{i=1}^{\tau}\big(x_i = x_{i+1} \vee \mathrm{tempadj}(x_i, x_{i+1}, i)\big)\Big)$$

to check whether there is a strict temporal (s,z)-path which does not visit any vertex in S. Note that the length of $\mathrm{layer}(e,t)$, and hence the length of $\mathrm{path}(S)$, is upper-bounded by some function in $2^{\mathcal{O}(\tau)}$. The facts that the length of a strict temporal (s,z)-path is at most τ and the treewidth of a planar graph can be bounded in its diameter (see Flum and Grohe [16]), together with an application of Courcelle's Theorem (optimization variant, see long version) on the MSO formula $\phi(S) := S \subseteq (V \setminus \{s,z\}) \wedge \neg\mathrm{path}(S)$ complete the proof. ◀

5 On Temporal Graphs with Small Temporal Cores

In this section, we investigate the complexity of deciding (Strict) Temporal (s,z)-Separation on temporal graphs where the number of vertices whose incident edges change over time is small. We call the set of such vertices the *temporal core* of the temporal graph.

▶ **Definition 5.1** (Temporal core)**.** For a temporal graph $\boldsymbol{G} = (V, \boldsymbol{E}, \tau)$, the vertex set $W = \{v \in V \mid \exists\{v,w\} \in (\bigcup_{i=1}^{\tau} E_i) \setminus (\bigcap_{i=1}^{\tau} E_i)\} \subseteq V$ is called the *temporal core*.

A temporal graph is often composed of a public transport system and an ordinary street network. Here, the temporal core consists of vertices involved in the public transport system.

For Strict Temporal (s,z)-Separation, we can observe that the hardness reduction described in the proof of Theorem 3.1 produces an instance with an empty temporal core. In stark contrast, we show that Temporal (s,z)-Separation is fixed-parameter tractable when parameterized by the size of the temporal core[8]. We reduce an instance to Node

[8] Note that we can compute the temporal core in $\mathcal{O}(|\boldsymbol{E}| \log |\boldsymbol{E}|)$ time.

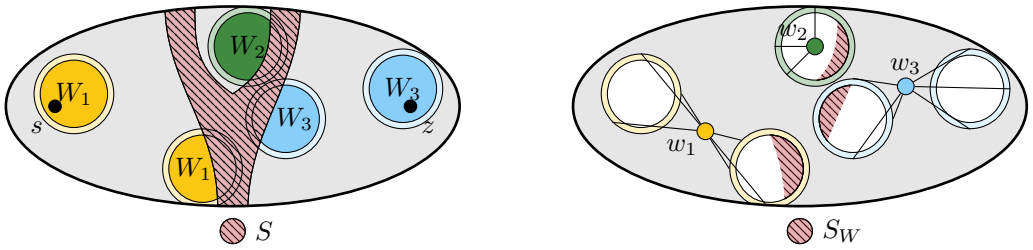


Figure 6 Illustration of the idea behind the proof of Theorem 5.2. *Left-hand side*: Sketch of a temporal graph $\boldsymbol{G}$ (enclosed by the ellipse) with temporal (s, z) separator S (red hatched) and induced partition $\{S_W, W_1, W_2, W_3\}$ of the temporal core W, where $S_W = W \cap S$. The outer rings of W_1, W_2, W_3 contain the open neighborhood of the sets. *Right-hand side*: Sketch of the constructed graph G' (enclosed by the ellipse). The partition $\{S_W, W_1, W_2, W_3\}$ is guessed in steps (1) and (2). The vertices w_1, w_2, w_3 with edges to the neighborhood of W_1, W_2, W_3, respectively, are created in step (3).

Multiway Cut (NWC) in such a way that we can use an above lower bound FPT-algorithm due to Cygan et al. [12] for NWC as a subprocedure in our algorithm for Temporal (s, z)-Separation. Note that the above lower bound parameterization is crucial to obtain the desired FPT-running time bound. Recall the definition of NWC:

Node Multiway Cut (NWC)
Input: An undirected graph $G = (V, E)$, a set of terminals $T \subseteq V$, and an integer k.
Question: Is there a set $S \subseteq (V \setminus T)$ of size at most k such there is no (t_1, t_2)-path for every distinct $t_1, t_2 \in T$?

We remark that Cygan et al.'s algorithm can be modified to obtain a solution S. Formally, we show the following.

▶ **Theorem 5.2.** Temporal (s, z)-Separation *can be solved in* $2^{|W|\cdot(\log|W|+2)} \cdot |V|^{\mathcal{O}(1)} + \mathcal{O}(|\boldsymbol{E}| \log |\boldsymbol{E}|)$ *time, where* W *denotes the temporal core of the input graph.*

Proof. Let instance $\mathcal{I} = (\boldsymbol{G} = (V, \boldsymbol{E}, \tau), s, z, k)$ of Temporal (s, z)-Separation with temporal core $W \subseteq V$ be given. Without loss of generality, we can assume that $s, z \in W$, as otherwise we add two vertices one being incident only with s and the other being incident only with z, both only in layer one. Furthermore, we need the notion of a *maximal static subgraph* $\widehat{G}$ of a temporal graph $\boldsymbol{G} = (V, \boldsymbol{E})$: It contains all edges that appear in every layer, more specifically $\widehat{G} = (V, \widehat{E})$ with $\widehat{E} = \bigcap_{i \in [\tau]} E_i$. Our algorithm works as follows.

(1) Guess a set $S_W \subseteq (W \setminus \{s, z\})$ of size at most k.
(2) Guess a number r and a partition $\{W_1, \ldots, W_r\}$ of $W \setminus S_W$ such that s and z are not in the same W_i, for some $i \in [r]$.
(3) Construct the graph G' by copying $\widehat{G} - W$ and adding a vertex w_i for each part W_i. Add edge sets $\{\{v, w_i\} \mid v \in N_{\widehat{G}}(W_i) \setminus W\}$ for all $i \in [r]$ and for all $i, j \in [r]$ add an edge $\{w_i, w_j\}$ if $N_{\widehat{G}}(W_i) \cap W_j \neq \emptyset$.
(4) Solve the NWC instance $\mathcal{I}' = (G', \{w_1, \ldots, w_r\}, k - |S_W|)$.
(5) If a solution S' is found for $\mathcal{I}'$ and $S' \cup S_W$ is a solution for $\mathcal{I}$, then output yes.
(6) If all possible guesses in (1) and (2) are considered without finding a solution for $\mathcal{I}$, then output no.

See Figure 6 for a visualization of the constructed graph G'. Since we do a sanity check in step (5) it suffices to show that if $\boldsymbol{G}$ has a temporal (s, z)-separator of size at most k, then there is a partition $\{S_W, W_1, \ldots, W_r\}$ of W where s and z are in different parts such that

(i) the NWC instance $\mathcal{I}'$ has a solution of size at most $k - |S_W|$, and
(ii) if S' is a solution to $\mathcal{I}'$, then $S_W \cup S'$ is a temporal (s,z)-separator in $\boldsymbol{G}$.

Let S be a temporal (s,z)-separator of size at most k in $\boldsymbol{G}$. First, we set $S_W = S \cap W$. Let $C_1, \ldots, C_r$ be the connected components of $\widehat{G} - S$ with $C_i \cap W \neq \emptyset$ for all $i \in [r]$. Now we construct a partition $\{S_W, W_1, \ldots, W_r\}$ of W such that $W_i = W \cap C_i$ for all $i \in [r]$. It is easy to see that s and z are in different parts of this partition. Observe that for $i, j \in [r]$ with $i \neq j$ the vertices $v \in W_i$ and $u \in W_j$ are in different connected components of $\widehat{G} - S$. Hence, $w_1, \ldots, w_r$ are in different connected components of $G' - (S \setminus S_W)$. Thus $S \setminus S_W$ is a solution of size at most $k - |S_W|$ of the NWC instance $\mathcal{I}' = (G', \{w_1, \ldots, w_r\}, k - |S_W|)$, proving (i).

For the correctness, it remains to prove (ii). Let S' be a solution of size at most $k - |S_W|$ of the NWC instance $\mathcal{I}'$. We need to prove that $S' \cup S_W$ forms a temporal (s,z)-separator in $\boldsymbol{G}$. Clearly, if $S' = S \setminus S_W$, we are done by the arguments before. Thus, assume $S' \neq S \setminus S_W$. Since S' is a solution to $\mathcal{I}'$, we know that $w_1, \ldots, w_r$ are in different connected components of $G' - S'$. Hence, for $i, j \in [r]$ with $i \neq j$ the vertices $v \in W_i$, $u \in W_j$ are in different connected components of $\widehat{G} - (S' \cup S_W)$.

Now assume towards a contradiction that there is a temporal (s,z)-path P in $\boldsymbol{G} - (S' \cup S_W)$. Observe that $\{s,z\} \subseteq V(P) \cap W$. Hence, we have two different vertices $u_1, u_2 \in V(P) \cap W$ such that there is no temporal (u_1,u_2)-path in $\boldsymbol{G} - S$ and all vertices that are visited by P between u_1 and u_2 are contained in $V \setminus W$: Take the furthest vertex in P that is also contained in W and is reachable by a temporal path from s in $\boldsymbol{G} - S$ as u_1, and take the next vertex (after u_1) in P that is also contained in W as u_2. Note that u_1 and u_2 are disconnected in $\widehat{G} - S$, and hence there are $i, j \in [r]$ with $i \neq j$ such that $u_1 \in W_i$ and $u_2 \in W_j$. Since P does not visit any vertices in $(S' \cup S_W)$ we can conclude that u_1 and u_2 are connected in $\widehat{G} - (S' \cup S_W)$, and hence w_i and w_j are connected in $G' - S'$. This contradicts the fact that S' is a solution for $\mathcal{I}'$.

Running time: It remains to show that the our algorithm runs in the proposed time. For the guess in step (1) there are at most $2^{|W|}$ many possibilities. For the guess in step (2) there are at most $B_{|W|} \leq 2^{|W| \cdot \log(|W|)}$ many possibilities, where B_n is the n-th Bell number. Step (3) and the sanity check in step (5) can clearly be done in polynomial time.

Let L be a minimum (s,z)-separator in $\widehat{G} - (W \setminus \{s,z\})$. If $k \geq |W \setminus \{s,z\}| + |L|$, then $(W \setminus \{s,z\}) \cup L$ is a temporal (s,z)-separator of size at most k for $\boldsymbol{G}$. Otherwise, we have that $k - |L| < |W|$. Cygan et al. [12] showed that NWC can be solved in $2^{k-b} \cdot |V|^{\mathcal{O}(1)}$ time , where $b := \max_{x \in T} \min\{|S| \mid S \subseteq V \text{ is an } (x, T \setminus \{x\})\text{-separator}\}$. Since s and z are not in the same W_i for any $i \in [r]$, we know that $|L| \leq b$. Hence, $k - b \leq k - |L| < |W|$ and step (4) can be done in $2^{|W|} \cdot |V|^{\mathcal{O}(1)}$ time. Thus we have an overall running time of $2^{|W| \cdot (\log |W| + 2)} \cdot |V|^{\mathcal{O}(1)} + \mathcal{O}(|\boldsymbol{E}| \log |\boldsymbol{E}|)$. ◀

We conclude that the strict and the non-strict variant of Temporal (s,z)-Separation behave very differently on temporal graphs with a constant-size temporal core. While the strict version stays NP-complete, the non-strict version becomes polynomial-time solvable.

6 Conclusion

The temporal path model strongly matters when assessing the computational complexity of finding small separators in temporal graphs. This phenomenon has so far been neglected in the literature. We settled the complexity dichotomy of Temporal (s,z)-Separation and Strict Temporal (s,z)-Separation by proving NP-hardness on temporal graphs

with $\tau \geq 2$ and $\tau \geq 5$, respectively, and polynomial-time solvability if the number of layers is below the respective constant. The mentioned hardness results also imply that both problem variants are W[1]-hard when parameterized by the solution size k. When considering the parameter combination $k + \tau$, it is easy to see that Strict Temporal (s, z)-Separation is fixed-parameter tractable [38]: There is a straightforward search-tree algorithm that branches on all vertices of a strict temporal (s, z)-path which has length at most τ. Whether the non-strict variant is fixed-parameter tractable regarding the same parameter combination remains open.

We showed that (Strict) Temporal (s, z)-Separation on temporal graphs with planar underlying graphs remains NP-complete. However, for the planar case we proved that if additionally the number τ of layers is a constant, then Strict Temporal (s, z)-Separation is solvable in $\mathcal{O}(|\boldsymbol{E}| \cdot \log |\boldsymbol{E}|)$ time. We leave open whether Temporal (s, z)-Separation admits a similar result. Finally, we introduced the notion of a temporal core as a temporal graph parameter. We proved that on temporal graphs with constant-size temporal core, while Strict Temporal (s, z)-Separation remains NP-hard, Temporal (s, z)-Separation is solvable in polynomial time.

References

1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, 1993.

2 Eleni C Akrida, Jurek Czyzowicz, Leszek Gąsieniec, Łukasz Kuszner, and Paul G Spirakis. Temporal flows in temporal networks. In *Proceedings of the 10th International Conference on Algorithms and Complexity (CIAC '17)*, pages 43–54. Springer, 2017.

3 Eleni C Akrida, Leszek Gąsieniec, George B Mertzios, and Paul G Spirakis. On temporally connected graphs of small cost. In *Proceedings of the 13th International Workshop on Approximation and Online Algorithms (WAOA '15)*, pages 84–96. Springer, 2015.

4 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

5 Kyriakos Axiotis and Dimitris Fotakis. On the size and the approximability of minimum temporally connected subgraphs. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP '16)*, pages 149:1–149:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

6 Georg Baier, Thomas Erlebach, Alexander Hall, Ekkehard Köhler, Petr Kolman, Ondřej Pangrác, Heiko Schilling, and Martin Skutella. Length-bounded cuts and flows. *ACM Transactions on Algorithms*, 7(1):4:1–4:27, 2010.

7 Kenneth A Berman. Vulnerability of scheduled networks and a generalization of Menger's Theorem. *Networks*, 28(3):125–134, 1996.

8 Stefano Boccaletti, Ginestra Bianconi, Regino Criado, Charo I Del Genio, Jesús Gómez-Gardenes, Miguel Romance, Irene Sendina-Nadal, Zhen Wang, and Massimiliano Zanin. The structure and dynamics of multilayer networks. *Physics Reports*, 544(1):1–122, 2014.

9 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

10 H.W Corley and David Y Sha. Most vital links and nodes in weighted networks. *Operations Research Letters*, 1(4):157–160, 1982.

11 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-order Logic: a Language-theoretic Approach*. Cambridge University Press, 2012.

12 Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. *ACM Transactions on Computation Theory*, 5(1):3:1–3:11, 2013.

13 Reinhard Diestel. *Graph Theory, 5th Edition*, volume 173 of *Graduate Texts in Mathematics*. Springer, 2016.

14 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP '15)*, pages 444–455. Springer, 2015.

15 Afonso Ferreira. Building a reference combinatorial model for MANETs. *IEEE Network*, 18(5):24–29, 2004.

16 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer-Verlag, Berlin, 2006.

17 Till Fluschnik, Danny Hermelin, André Nichterlein, and Rolf Niedermeier. Fractals for kernelization lower bounds. *SIAM Journal on Discrete Mathematics*, 32(1):656–681, 2018.

18 Till Fluschnik, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Temporal graph classes: A view through temporal separators. *arXiv preprint arXiv:1803.00882*, 2018. To appear in *Proceedings 44th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '18)*.

19 Lester R Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956.

20 Petr A. Golovach and Dimitrios M. Thilikos. Paths of bounded length and their cuts: Parameterized complexity and algorithms. *Discrete Optimization*, 8(1):72–86, 2011.

21 Anne-Sophie Himmel. Algorithmic investigations into temporal paths. Masterthesis, TU Berlin, April 2018. URL: `http://fpt.akt.tu-berlin.de/publications/theses/MA-anne-sophie-himmel.pdf`.

22 Anne-Sophie Himmel, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. Adapting the Bron-Kerbosch algorithm for enumerating maximal cliques in temporal graphs. *Social Network Analysis and Mining*, 7(1):35:1–35:16, 2017.

23 Petter Holme. Modern temporal network theory: a colloquium. *European Physical Journal B*, 88(9):234, 2015.

24 Petter Holme and Jari Saramäki. Temporal networks. *Physics Reports*, 519(3):97–125, 2012.

25 David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002.

26 Matthieu Latapy, Tiphaine Viard, and Clémence Magnien. Stream graphs and link streams for the modeling of interactions over time. *arXiv preprint arXiv:1710.04073*, 2017.

27 Qingkai Liang and Eytan Modiano. Survivability in time-varying networks. *IEEE Transactions on Mobile Computing*, 16(9):2668–2681, 2017.

28 László Lovász, Víctor Neumann-Lara, and Michael Plummer. Mengerian theorems for paths of bounded length. *Periodica Mathematica Hungarica*, 9(4):269–276, 1978.

29 Karl Menger. Zur allgemeinen Kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927.

30 George B Mertzios, Othon Michail, Ioannis Chatzigiannakis, and Paul G Spirakis. Temporal network optimization subject to connectivity constraints. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP '13)*, pages 657–668. Springer, 2013.

31 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016.

32 Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theoretical Computer Science*, 634:1–23, 2016.

33 Feng Pan and Aaron Schild. Interdiction problems on planar graphs. *Discrete Applied Mathematics*, 198:215–231, 2016.

34 Salvatore Scellato, Ilias Leontiadis, Cecilia Mascolo, Prithwish Basu, and Murtaza Zafer. Evaluating temporal robustness of mobile networks. *IEEE Transactions on Mobile Computing*, 12(1):105–117, 2013.

35 Baruch Schieber, Amotz Bar-Noy, and Samir Khuller. The complexity of finding most vital arcs and nodes. Technical report, University of Maryland at College Park, College Park, MD, USA, 1995.

36 Tiphaine Viard, Matthieu Latapy, and Clémence Magnien. Computing maximal cliques in link streams. *Theoretical Computer Science*, 609:245–252, 2016.

37 Huanhuan Wu, James Cheng, Yiping Ke, Silu Huang, Yuzhen Huang, and Hejun Wu. Efficient algorithms for temporal path computation. *IEEE Transactions on Knowledge and Data Engineering*, 28(11):2927–2942, 2016.

38 Philipp Zschoche. On finding separators in temporal graphs. Masterthesis, TU Berlin, August 2017. URL: `http://fpt.akt.tu-berlin.de/publications/theses/MA-philipp-zschoche.pdf`.

The Complexity of Transducer Synthesis from Multi-Sequential Specifications

Léo Exibard[1]
Aix-Marseille Université, Marseille, France
Université libre de Bruxelles, Brussels, Belgium
leo.exibard@ulb.ac.be

Emmanuel Filiot[2]
Université libre de Bruxelles, Brussels, Belgium
efiliot@ulb.ac.be

Ismaël Jecker[3]
Université libre de Bruxelles, Brussels, Belgium
ismael.jecker@ulb.ac.be

Abstract

The transducer synthesis problem on finite words asks, given a specification $S \subseteq I \times O$, where I and O are sets of finite words, whether there exists an implementation $f : I \to O$ which (1) fulfils the specification, i.e., $(i, f(i)) \in S$ for all $i \in I$, and (2) can be defined by some input-deterministic (aka sequential) transducer $\mathcal{T}_f$. If such an implementation f exists, the procedure should also output $\mathcal{T}_f$. The realisability problem is the corresponding decision problem.

For specifications given by synchronous transducers (which read and write alternately one symbol), this is the finite variant of the classical synthesis problem on ω-words, solved by Büchi and Landweber in 1969, and the realisability problem is known to be ExpTime-c in both finite and ω-word settings. For specifications given by asynchronous transducers (which can write a batch of symbols, or none, in a single step), the realisability problem is known to be undecidable.

We consider here the class of multi-sequential specifications, defined as finite unions of sequential transducers over possibly incomparable domains. We provide optimal decision procedures for the realisability problem in both the synchronous and asynchronous setting, showing that it is PSpace-c. Moreover, whenever the specification is realisable, we expose the construction of a sequential transducer that realises it and has a size that is doubly exponential, which we prove to be optimal.

2012 ACM Subject Classification Theory of computation → Logic and verification, Theory of computation → Transducers

Keywords and phrases Transducers, Multi-Sequentiality, Synthesis

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.46

Acknowledgements We warmly thank the anonymous reviewers for their helpful comments and Christof Löding for pointing us to some related references.

[1] L. Exibard is a PhD student funded by a FRIA fellowship from the F.R.S.-FNRS.
[2] E. Filiot is a research associate of F.R.S.- FNRS. He is supported by the ARC Project Transform Fédération Wallonie-Bruxelles and the FNRS CDR project J013116F.
[3] I. Jecker is an "aspirant FNRS" PhD student, funded by the F.R.S.-FNRS.

1 Introduction

The realisability and synthesis problem. In general, the realisability problem is given by some input and output data domains $D_{\mathtt{i}}, D_{\mathtt{o}}$, a *specification* $S \subseteq D_{\mathtt{i}} \times D_{\mathtt{o}}$ defining for every $d \in D_{\mathtt{i}}$ the set of allowed outputs $\{d' \in D_{\mathtt{o}} \mid (d, d') \in S\}$ (assumed to be non-empty) and a class of target *implementations* $\mathcal{I}$ consisting of (total) functions $D_{\mathtt{i}} \to D_{\mathtt{o}}$. It asks whether there exists $f \in \mathcal{I}$ such that for all $d \in D_{\mathtt{i}}$, $(d, f(d)) \in S$, i.e., the implementation f satisfies the specification. The synthesis problem asks to generate (a representation of) f. So, instead of designing f and verifying its correctness *a posteriori*, a synthesis algorithm automatically generates f from it, making it *correct by construction*. The underlying idea behind synthesis is that the specification may be written in a high-level language, e.g. a logic, and an implementation is a low-level computational model e.g. an automaton. It is based on the assumption that it is less error-prone to design a specification, i.e. to describe *what* a system has to do, than designing the system itself, i.e. describing *how* it must do it.

Synchronous transducer synthesis. In the original setting defined by Church [7, 29], $D_{\mathtt{i}}, D_{\mathtt{o}}$ are sets of ω-words over two alphabets $\Sigma_{\mathtt{i}}, \Sigma_{\mathtt{o}}$ respectively, and the specification S is given by an ω-language $L \subseteq (\Sigma_{\mathtt{i}} \times \Sigma_{\mathtt{o}})^\omega$ as follows: $S = \{(\pi_1(w), \pi_2(w)) \mid w \in L\}$, where π_i is the projection on the ith component. The language L is represented by an MSO-sentence or, equivalently, an automaton. Such automata are also called (non-deterministic) *synchronous transducers*, as they can be seen as machines alternately reading one input symbol and synchronously producing one output symbol. In Church's setting, the target implementations are synchronous *sequential* transducers (also called *input-deterministic*): they alternately read one input symbol and deterministically produce a symbol to output. Determinism is required because implementations are required to use only finite-memory. The Church's instance of the realisability problem is decidable if the specification is given in MSO and ExpTime-c if it is given by a synchronous transducer [21]. For LTL specifications, it is 2ExpTime-c [27]. Motivated by reactive systems, the synthesis problem from LTL specifications has been revisited recently with efficient symbolic methods [20, 28, 14, 10, 18]. The synthesis problem in general has also motivated an active research on infinite games [1, 9, 5].

Asynchronous transducer synthesis. In the asynchronous setting, specifications may not strictly alternate between input and output symbols, hence they can no longer be seen as languages over $\Sigma_{\mathtt{i}} \times \Sigma_{\mathtt{o}}$. Similarly, the target implementations may not be synchronous: the system can delay its production of outputs, or produce several symbols at once. Transducers, in contrast to synchronous transducers, are by definition asynchronous: their transitions are labelled by pairs (i, w) where $i \in \Sigma_{\mathtt{i}}$ is a symbol and $w \in \Sigma_{\mathtt{o}}^*$ a word, possibly empty. Since they are generally non-deterministic, to a single input word may correspond several output words, and thus transducers define subsets of $\Sigma_{\mathtt{i}}^* \times \Sigma_{\mathtt{o}}^*$. Therefore, they are well-suited to represent (asynchronous) specifications, and in their sequential version, asynchronous implementations. Any asynchronous specification is realisable by some unambiguous (functional) asynchronous transducer [24, 11, 3]. However, evaluating unambiguous transducers on arbitrarily long or even infinite input words may require arbitrarily large memory. Therefore, just as in Church's setting, a sequentiality requirement can be put on implementations for efficient memory usage. However, the realisability of asynchronous specifications by (asynchronous) sequential transducers, which is called the *sequential uniformisation problem* in transducer-theoretic terms, is undecidable for finite words [4, 12]. If the specification is finite-valued (i.e. any input word has a constant number of output words), the problem is in 3ExpTime [12]. The

proof of [12] is based on Ramsey's theorem and word combinatorics arguments, and it is not clear how to reduce the complexity. This raises the question of whether there are natural and non-trivial subclasses of asynchronous specifications with better complexity.

Multi-sequential specifications. In this paper, we consider a class of specifications S on finite words, i.e. $S \subseteq \Sigma_{\mathtt{i}}^* \times \Sigma_{\mathtt{o}}^*$, which strictly restricts the class of finite-valued specifications to so-called multi-sequential specifications. Such class is obtained by closure under finite unions of graphs of sequential functions. Precisely, $S = \bigcup_{i=1}^{n} S_i$ where S_i is the graph of a (partial) function $f_i : \Sigma_{\mathtt{i}}^* \to \Sigma_{\mathtt{o}}^*$ defined by a sequential transducer. Likewise, a transducer is multi-sequential if it is a union of state-disjoint sequential transducers. For instance, consider the specification S consisting of the pairs (w, w') such that w' is a subword of w of fixed length k. This specification is multi-sequential: $S = \bigcup_{w' \in \Sigma_{\mathtt{i}}^k} S_{w'}$ where $S_{w'} = \{(w, w') \mid w' \text{ subword of } w\}$. Clearly, $S_{w'}$ can be represented by a sequential transducer, because w' is fixed: once the first symbol of w' is met in w, output it, and proceed to the second symbol of w', etc., until the last symbol of w' is produced, reject otherwise. The notion of multi-sequentiality has been introduced for functions in [6] and studied for relations in [19]. An important property of the class of multi-sequential specifications is its decidability in PTime: Given a transducer, it is decidable in PTime whether it defines a multi-sequential specification [19]. This fact and their natural definition as closure of sequential functions under union make multi-sequential specifications a good candidate for a class of specifications with better complexity than the known results of the literature.

Contributions. We investigate the complexity of sequential transducer synthesis from specifications defined by multi-sequential transducers on finite words. We show that both in the synchronous and asynchronous settings, the realisability problem is PSpace-complete. To the best of our knowledge, it is the first non-trivial class of specifications which admits a realisability test below ExpTime. If the specification is realisable, we show how to extract an implementation as a winning strategy in a two-player game called the *synthesis game*. It is parameterised by a value $k \in \mathbb{N}$ which bounds the maximal number of output symbols which can be queued before being outputted, allowing for an incremental synthesis algorithm. To keep track of such output symbols, we use the notion of *delay* [2].

Difficulties and examples. Let us briefly explain what are the main difficulties to overcome. Consider $S = \bigcup_i S_i$ a multi-sequential specification. If all of the S_i have disjoint domains, then S is a function, which is realisable by a sequential transducer iff it is sequential. The latter can be tested in PTime [2]. The problem becomes more interesting and challenging when the S_i have domains which are not necessarily disjoint. Consider for example the 2-sequential transducer $\mathcal{D}_1 \cup \mathcal{D}_2$ of Fig. 1. The transducer $\mathcal{D}_1$ accepts the words containing at least two a's, and replaces b's with a's, and $\mathcal{D}_2$ accepts the words containing at least one b, and replaces a's with b's. This specification can be realised by a sequential transducer which waits two steps before outputting something, since it then knows whether the input contains at least one b or two a's. It then behaves as $\mathcal{D}_1$ in the first case, and as $\mathcal{D}_2$ in the second.

This example shows that a sequential realiser may have to wait before reacting, keeping in memory what remains to be output in the future. Take on the contrary the 2-sequential transducer $\mathcal{D}'_1 \cup \mathcal{D}'_2$ of Fig. 1, which is the same as $\mathcal{D}_1 \cup \mathcal{D}_2$ except that it can additionally read and copy c's at any moment. In that case, a sequential realiser would have to store arbitrary long sequences of c's before outputting them, for instance when processing words in $ac^*\{a, b, c\}^*$. In particular, this specification is not sequentially realisable.

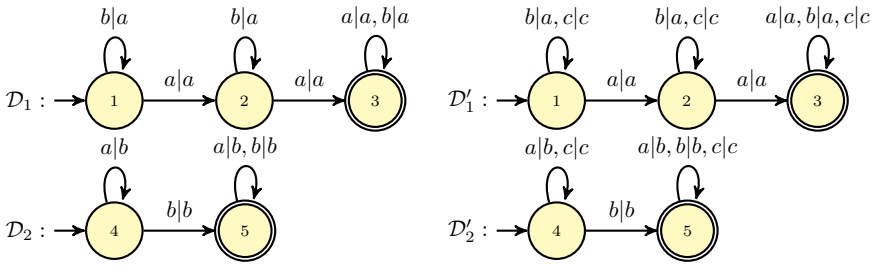


Figure 1 Four synchronous sequential transducers.

Structure of the paper. After a formal definition of transducer synthesis (Section 2), we solve the synchronous case and provide a characterisation of realisable synchronous multi-sequential specifications, decidable in PSpace (Section 3). Then, we present the notion of synthesis game (Section 4), which is a useful tool for the proofs and also to get a synthesis procedure. For the asynchronous setting, we define a recursive characterisation of realisable multi-sequential specifications and show that it can be decided in PSpace (Section 5).

Related work. Games with delays have been used in [4, 15]. Perhaps the closest formulation to ours is that of [4]. However, it is tailored to automatic relation. Our game structure is more general, as it is defined for uniformising any transducer (defining a rational relation). In particular, our game structure is exponentially larger that the one of [4].

We would also like to mention an interesting related line of works on ω-words, where the specification is synchronous, but the implementation may be asynchronous [15, 17, 22, 23, 30, 31, 32]. Unlike the setting where the specification and implementations are both asynchronous, the realisability problem is decidable here, for ω-regular specifications (i.e., regular ω-languages over $\Sigma_{\mathtt{i}} \times \Sigma_{\mathtt{o}}$), and ExpTime-c if the specification is given by a parity automaton [22]. In this setting, the authors often consider a notion of delay games. In these games, the delay is a quantitative notion, corresponding to the waiting time before outputting a symbol, while for us, a delay is a word that still remains to be output (this is a standard terminology in transducer theory). It is known in particular that constant "waiting time" (depending on the specification) is always sufficient to win, for ω-regular specifications. This is different to our setting: for instance, the function f mapping any word of the form $a^n\sigma$, for $n > 0$ and $\sigma \in \{a, b\}$, to σ is realisable by a sequential transducer, but the production of a and b might have to be delayed for an unbounded amount of time.

2 Transducer synthesis problem

Words. For an alphabet Σ, we denote by Σ^* the set of finite words over it, and by ϵ the empty word. The length $|w|$ of a word w is its number of symbols. For $k \in \mathbb{N}$, we denote by Σ^k (resp. $\Sigma^{\leq k}$) the set of words of length k (resp. at most k). For $u, v \in \Sigma^*$, we write $u \preceq v$ if u is a prefix of v, and denote by $u^{-1}v$ the word such that $u(u^{-1}v) = v$. For $L \subseteq \Sigma^*$, the residual language $u^{-1}L$ is $u^{-1}L = \{u' \mid uu' \in L\}$. Given $S \subseteq \Sigma^* \times \Gamma^*$, and $(u, v) \in \Sigma^* \times \Gamma^*$, the residual relation $(u, v)^{-1}S$ is defined by $(u, v)^{-1}S = \{(u', v') \mid (uu', vv') \in S\}$.

Automata. In this paper, finite (non-deterministic) automata over an alphabet Σ are denoted as tuples $\mathcal{A} = (\Sigma, Q, I, F, \Delta)$ where Σ is the alphabet, Q the set of states, among which I (resp. F) denotes the initial (resp. final or accepting) states, and $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation. $\mathcal{A}$ is deterministic if there is only one initial state and for all $(q, \sigma) \in Q \times \Sigma$, there exists at most one $q' \in Q$ such that $(q, \sigma, q') \in \Delta$.

A *run* of $\mathcal{A}$ on a word $w = \sigma_1 \dots \sigma_n$ consists in either a single state $q \in Q$ if $n = 0$, or a sequence $r \in \Delta^*$ of n transitions $t_1 \dots t_n$ such that the target state of t_i equals the source state of t_{i+1} for all $1 \leq i < n$. It is said to be initial if the source state of t_1 is initial, and accepting if the target state of t_n is accepting. If p is the source state of t_1 and q the target state of t_n, we may write $p \xrightarrow{w}_{\mathcal{A}} q$ to mean that there exists a run from p to q on w. The language accepted by an automaton $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words admitting an accepting run. A state $q \in Q$ is reachable (resp. co-reachable) if there is a run from an initial state (resp. to a final state) for some $u \in \Sigma^*$. A state is said to be *useful* if it is both reachable and co-reachable, and $\mathcal{A}$ is said to be *trim* if all its states are useful. It is well-known that any automaton can be transformed into an equivalent trim automaton in PTime. Given two automata $\mathcal{A}_1 = (\Sigma, Q_1, I_1, F_1, \Delta_1)$ and $\mathcal{A}_2 = (\Sigma, Q_2, I_2, F_2, \Delta_2)$, their disjoint union $\mathcal{A}_1 \uplus \mathcal{A}_2$ is the automaton $(\Sigma, Q_1 \uplus Q_2, I_1 \uplus I_2, F_1 \uplus F_2, \Delta_1 \uplus \Delta_2)$.

Transducers. A *transducer*[4] over two alphabets Σ, Γ is a tuple $\mathcal{T} = (\mathcal{A}, \rho, \tau)$ such that $\mathcal{A} = (\Sigma, Q, I, F, \Delta)$ is an automaton over Σ, called the *input automaton*, $\rho : \Delta \to \Gamma^*$ is a mapping, called the output function, associating with every transition an output word, and $\tau : F \to \Gamma^*$ is a terminal function associating with every accepting state an output word. Given a run r of $\mathcal{A}$ on a word w, its output $\text{out}(r) \in \Gamma^*$ is defined by ϵ if $w = \epsilon$, and by $\rho(t_1) \dots \rho(t_n)$ if $r = t_1 \dots t_n$ for some $n \geq 1$. We write $p \xrightarrow{u|v}_{\mathcal{T}} q$ whenever there exists a run r of $\mathcal{A}$ on u from p to q, such that $v = \text{out}(r)$, and say that r *produces* v. The *relation* defined by $\mathcal{T}$ is the set $[\![\mathcal{T}]\!]$ of pairs $(u, v\tau(q)) \in \Sigma^* \times \Gamma^*$ such that $p \xrightarrow{u|v}_{\mathcal{T}} q$ for $p \in I$ and $q \in F$. We define $\text{dom}(\mathcal{T})$ by $\text{dom}(\mathcal{T}) = \text{dom}([\![\mathcal{T}]\!]) = L(\mathcal{A})$.

A transducer is *trim* if its input automaton is trim. It is called *sequential* if its input automaton is deterministic, and *functional* if $[\![\mathcal{T}]\!]$ is a function, i.e. for all $u \in \text{dom}(\mathcal{T})$, there exists at most one pair $(u, v) \in [\![\mathcal{T}]\!]$. In that case we let $\mathcal{T}(u) = v$. Note that any sequential transducer is functional. A transducer $\mathcal{T} = (\mathcal{A}, \rho, \tau)$ is called *synchronous* (or sometimes *letter-to-letter* in the literature) if, whenever it reads an input symbol, it produces exactly one output symbol, i.e. for all transition t, $|\rho(t)| = 1$, and $\tau(q) = \epsilon$ for all accepting state q. For example, consider the transducer $\mathcal{D}_1$ on Fig. 1 (the terminal function is assumed to output ϵ and is not depicted). It is sequential and synchronous. Its domain is $L = b^*ab^*a(a + b)^*$.

Two transducers are said to be *equivalent* if they define the same relation. Finally, the disjoint union of transducers is naturally defined as the disjoint union of their input automata and the disjoint union of their output functions (seen as graphs). For all transducers $\mathcal{T}_1, \mathcal{T}_2$, we have $[\![\mathcal{T}_1 \uplus \mathcal{T}_2]\!] = [\![\mathcal{T}_1]\!] \cup [\![\mathcal{T}_2]\!]$.

Transducer Synthesis Problem. Let $\Sigma_{\mathtt{i}}, \Sigma_{\mathtt{o}}$ be two alphabets of input and output symbols respectively. They may not necessarily be disjoint. A *specification* is a subset of $\Sigma_{\mathtt{i}}^* \times \Sigma_{\mathtt{o}}^*$, and an *implementation* is a function, possibly partial, from $\Sigma_{\mathtt{i}}^*$ to $\Sigma_{\mathtt{o}}^*$. The transducer *realisability* problem asks, given a specification S defined by a transducer $\mathcal{T}$, i.e. $S = [\![\mathcal{T}]\!]$, whether there exists a sequential transducer $\mathcal{I}$ such that (1) $\text{dom}(\mathcal{I}) = \text{dom}(\mathcal{T})$ and (2) for all $u \in \text{dom}(\mathcal{T})$, $(u, \mathcal{I}(u)) \in [\![\mathcal{T}]\!]$. In that case, we say that $\mathcal{I}$ realises S (or $\mathcal{T}$), and that S is realisable by a sequential transducer, or sequentially realisable. We also say that $\mathcal{I}$ is a *realiser* of S. The synthesis problem asks to output $\mathcal{I}$. The realisability problem is undecidable in general [4, 12], but decidable, in 3ExpTime, if $\mathcal{T}$ is finite-valued, i.e. there exists $k \in \mathbb{N}$ such that for all $u \in \text{dom}(\mathcal{T})$, $|\{v \mid (u, v) \in [\![\mathcal{T}]\!]\}| \leq k$ [12].

[4] Our definition is sometimes called *real-time* transducer in the literature, in contrast to transducers with ϵ-input transitions. For the purpose of this paper, this does not make a difference.

Multi-sequential specifications. A transducer $\mathcal{T}$ is called *k-sequential* if it is the disjoint union of k sequential transducers. It is called *multi-sequential* if it is k-sequential for some k. Observe that when the k sequential transducers have pairwise disjoint domains, then $\mathcal{T}$ is functional, but it may not be the case in general. Deciding whether given a transducer $\mathcal{T}$, there exists an equivalent multi-sequential transducer $\mathcal{T}'$, can be done in PTime; however, $\mathcal{T}'$ may be exponentially larger than $\mathcal{T}$ [19]. Minimising the number of sequential transducers of the disjoint union is also doable: deciding whether $\mathcal{T}$ is equivalent to some k-sequential transducer for k given in unary is decidable in PSpace [8]. In this paper, we consider *multi-sequential specification*, i.e. relations $S \subseteq \Sigma_{\mathtt{i}}^* \times \Sigma_{\mathtt{o}}^*$ defined by multi-sequential transducers.

PSpace-hardness. In both the synchronous and asynchronous case, the realisability problem of multi-sequential specifications by (a)synchronous sequential transducers is PSpace-hard.

We build a reduction from the emptiness problem of the intersection of n DFA $\mathcal{A}_1, \dots, \mathcal{A}_n$ on some alphabet Σ, proven PSpace-c in [25]. We define a specification S over $\Sigma \cup \{\#, a, b\}$ by $S = \bigcup_{i=1}^{n}(S_i \cup N_i)$ where $S_i = \{(w\#^m\sigma, w\sigma\#^m) \mid \sigma \in \{a,b\}, m \geq 0, w \in L(\mathcal{A}_i)\}$ and $N_i = \{(w\#^m\sigma, w\#^m\sigma) \mid \sigma \in \{a,b\}, m \geq 0, w \notin L(\mathcal{A}_i)\}$. If there exists $w \in \bigcap_{i=1}^{n} L(\mathcal{A}_i)$, then on the domain $w\#^*\{a,b\}$, the specification is a function which is not definable by any sequential transducer, thus not sequentially realisable, since it would imply counting the #s (in the synchronous setting, it suffices to take $m = 1$ since a synchronous transducer would be forced to guess the future). Conversely, if $\bigcap_{i=1}^{n} L(\mathcal{A}_i) = \varnothing$, then the identity function (trivially definable by a synchronous sequential transducer) realises the specification.

It is readily seen that each S_i (resp. N_i) is definable by a 2-(resp. 1-)sequential transducer, hence S is multi-sequential, concluding the proof.

3 The synchronous setting

In this section, we consider first the synchronous setting, where the specification is given as a disjoint union of synchronous sequential transducers, and the target implementations are synchronous sequential transducers. Not only is this setting interesting in itself, but it helps to understand the asynchronous setting. First, we characterise the realisable specifications through a property called the *residual property*, then we show it is decidable in PSpace.

Residual property. Let $\mathcal{T} = \biguplus_{i=1}^{n} \mathcal{D}_i$ be an n-sequential transducer on $\Sigma_{\mathtt{i}}, \Sigma_{\mathtt{o}}$. Intuitively, the residual property says that if on some input prefix u, two sequential transducers of the union disagree on their outputs, i.e. produce different outputs, then a synchronous realiser of $[\![\mathcal{T}]\!]$ must "drop" one of the two transducers. However, it must do so while preserving the residual domain $u^{-1}\mathrm{dom}(\mathcal{T})$, i.e., the realiser must still accept any word of $u^{-1}\mathrm{dom}(\mathcal{T})$. For example, consider again Fig. 1 and the specification defined by $\mathcal{D}_1 \uplus \mathcal{D}_2$. On input a, the two transducers disagree, hence, since we want a synchronous realiser, a choice has to be made and therefore one of the two transducers must be dropped. However, by doing so, the residual domain will not be fully covered by the remaining transducer. For example, if a realiser chooses to output a when reading a, the residual language b^* is not covered anymore. As a matter of fact, $\mathcal{D}_1 \uplus \mathcal{D}_2$ is not realisable by any sequential and synchronous transducer.

Formally, let $u \in \Sigma_{\mathtt{i}}^*$ and let r_i, r_j be runs of some $\mathcal{D}_i, \mathcal{D}_j$ respectively, on u. We say that r_i and r_j agree on their output if $\mathrm{out}(r_i) = \mathrm{out}(r_j)$. Now, u is called *smooth* if every $\mathcal{D}_i$ admits an initial run on input u, and all these runs agree on the corresponding output. The word u is called *critical* if it is not smooth.

We say that $\mathcal{T}$ satisfies the *residual property* if for every critical prefix $u \in \Sigma_{\mathtt{i}}^*$ of a word of $\mathrm{dom}(\mathcal{T})$, there exists a subset $P \subsetneq \{1, \dots, n\}$ satisfying:

1. All the transducers $\mathcal{D}_i$, $i \in P$, produce the same output $\phi(u)$ on u;
2. $u^{-1}\mathrm{dom}(\mathcal{T}) = \bigcup_{i \in P} u^{-1}\mathrm{dom}(\mathcal{D}_i)$;
3. $\biguplus_{i \in P}(u, \phi(u))^{-1}[\![\mathcal{D}_i]\!]$ is realisable by a synchronous and sequential transducer.

▶ **Theorem 1.** *A specification S defined by a synchronous multi-sequential transducer $\mathcal{T}$ is realisable by a synchronous sequential transducer iff $\mathcal{T}$ satisfies the residual property.*

Sketch. If $[\![\mathcal{T}]\!]$ is realised by a synchronous sequential transducer $\mathcal{U}$, for every critical prefix u, let P be the set of i such that $\mathcal{D}_i$ and $\mathcal{U}$ map the same output to u. Property 1 is satisfied by definition, and the other two follow from the fact that $\mathcal{U}$ is sequential and realises $[\![\mathcal{T}]\!]$.

Conversely, if the residual property is satisfied, we can construct a synchronous and sequential realiser. The idea is to make a synchronised product of all the transducers $\mathcal{D}_i$, and, whenever on some input symbol σ at least two of them disagree on the output, we know by the residual property that there exists a subset P of them having the good properties $1, 2, 3$. Then, the realiser just goes on simulating all the transducers $\mathcal{D}_i$ corresponding to P in parallel.

It also shows that if the property is satisfied, then we can synthesise a realiser, which might however be exponentially larger than $\mathcal{T}$. ◀

▶ **Theorem 2.** *The realisability problem of synchronous multi-sequential specifications by synchronous sequential transducers is PSpace-complete.*

Sketch. The PSpace-hardness is obtained by reducing the problem from the emptiness problem of the intersection of n DFAs (*cf* Section 2 p. 6).

To show membership to PSpace, given a transducer $\mathcal{T} = \biguplus_{i=1}^{n} \mathcal{D}_i$, we show that the residual property can be tested by a non-deterministic algorithm running in polynomial space. First, we bound the size of witnesses of the negation of the property: roughly, if there is such witness, namely a critical prefix u, then there exists a critical prefix v of exponential length (in $\mathcal{T}$) such that for any subset $P \subsetneq \{1, \dots, n\}$, one of the conditions $1, 2, 3$ is falsified. Then, the algorithm guesses the prefix v on the fly, simulating all transducers $\mathcal{D}_i$ in parallel and keeping their states in memory (it also needs a counter for the length of v). As soon as the transducers disagree on an output symbol, for each subset $P \subsetneq \{1, \dots, n\}$ (they can obviously be enumerated using only polynomial space), the algorithm checks whether property 1, 2 or 3 is falsified. Checking property 1 is easy: it suffices to look at the symbols produced when reading the last input symbol. Checking property 2 can be done using the current set of states reached by the transducers on input v, and by using any PSpace algorithm for automata inclusion. Finally, to check property 3, it suffices to recursively apply the PSpace algorithm described so far on a smaller set of transducers. The stack of recursive calls is linear in n, hence the memory used by the whole procedure remains polynomial. ◀

4 The synthesis game

We now define a 2-player safety game from a transducer $\mathcal{T}$ such that if Eve wins the game then $\mathcal{T}$ is realisable by a sequential transducer. This game notion will prove useful to show the correctness of the characterisation of Theorem 5, and may also be used as a practical way to synthesise implementations, as winning strategies of this game. In the asynchronous setting, two different runs of a transducer on the same input word may not only produce different outputs, but also the same output at different rates (i.e. one run is ahead, output-wise, of the other for some time). This leads us to the notion of delays, a classical tool to compare outputs in transducer theory. Let us define this notion formally.

Delays. Given two words $u_1, u_2 \in \Sigma^*$, their longest common prefix ℓ is denoted by $u_1 \wedge u_2$. The *delay* between u_1 and u_2 is an element of $\Sigma^* \times \Sigma^*$ defined by $\mathrm{del}(u_1, u_2) = (\ell^{-1}u_1, \ell^{-1}u_2)$. Intuitively, if a transducer produces u_1 and another one produces u_2, then $u_1 \wedge u_2$ is what can safely be output by the two transducers and $\mathrm{del}(u_1, u_2)$ what remains to be produced by each of them respectively. This notion is naturally extended to tuples of words: $\mathrm{del}(u_1, \ldots, u_n) = (\ell^{-1}u_1, \ldots, \ell^{-1}u_n)$ where $\ell = \bigwedge_{i=1}^{n} u_i$.

We now introduce notations that are useful when comparing the outputs of different runs on the same input of a transducer $\mathcal{T} = (\mathcal{A}, \rho, \tau)$ over $\Sigma_{\mathtt{i}}, \Sigma_{\mathtt{o}}$ with $\mathcal{A} = (\Sigma_{\mathtt{i}}, Q, I, F, \Delta)$. Given a pair $(q, w) \in Q \times \Sigma_{\mathtt{o}}^*$, where w is intended to be some delay associated with state q, given a transition $t = (q, \sigma, q') \in \Delta$ and some output word u prefix of $w\rho(t)$, we denote by $\mathit{next}((q, w), t, u)$ the "next" pair (state,delay), assuming that u is output, i.e. $\mathit{next}((q, w), t, u) = (q', u^{-1}w\rho(t))$. More generally, given a (total) function $D : Q \to 2^{\Sigma_{\mathtt{o}}^*}$ associating each state with a set of delays, we let $\mathit{live}(D) = \{q \in Q \mid D(q) \neq \varnothing\}$. For $\sigma \in \Sigma_{\mathtt{i}}$, $\mathit{next}(D, \sigma)$ maps every state which can be reached from $\mathrm{dom}(D)$ by reading σ to the corresponding delays obtained by outputting the longest common prefix of the words that can be formed from the previous delays and the output on these transitions. Formally, we call *safe output of D for σ* the word $\ell = \bigwedge\{w\rho(t) \mid q \in \mathit{live}(D), w \in D(q), t = (q, \sigma, q') \in \Delta\}$. Then $\mathit{next}(D, \sigma) = \{\mathit{next}((q, w), t, \ell) \mid q \in \mathit{live}(D), w \in D(q), t = (q, \sigma, q') \in \Delta\}$.

The synthesis game. In the synchronous setting, synthesis problems are classically solved by reduction to two-player games in which the players alternately choose one input symbol (the adversary, whom we call Adam) and one output symbol (the protagonist, called Eve). Their interaction induces a pair of input and output words by concatenating their respective symbols, and the protagonist wins if such pair satisfies the specification, or if the input word is out of the domain. Then, a finite-memory winning strategy in the game corresponds to an implementation of the specification.

In the asynchronous setting, the protagonist may choose arbitrary output words at each round instead of a single symbol, and one needs to introduce output delays in the game in order to define the winning condition in a regular manner. The game we now present follows this idea. Given a transducer $\mathcal{T} = (\mathcal{A}, \rho, \tau)$ with $\mathcal{A} = (\Sigma_{\mathtt{i}}, Q, I, F, \Delta)$, $\rho : \Delta \to \Sigma_{\mathtt{o}}^*$ and $\tau : F \to \Sigma_{\mathtt{o}}^*$, we build a two-player safety game $G_{\mathcal{T}} = (V_\forall, V_\exists, A_\forall, A_\exists, T_\forall, T_\exists, \mathsf{Safe})$, called the *synthesis game*, whose vertices keep track of the runs in $\mathcal{T}$ and the associated delays. More precisely, it consists of two disjoint sets of vertices $V_\forall = 2^Q \times (Q \to 2^{\Sigma_{\mathtt{o}}^*})$ and $V_\exists = V_\forall \times \Sigma_{\mathtt{i}}$, respectively controlled by Adam and Eve. The initial vertex is $v_0 = (I, D_0) \in V_\forall$ where $D_0(q) = \varnothing$ if $q \notin I$, and $D_0(q) = \{\epsilon\}$ otherwise.

Eve's vertices are Adam's vertices extended with the last input symbol picked by Adam. Suppose now that the game has been played for some rounds and is currently in some vertex (C, D) of Adam. Along these rounds, Adam has chosen a sequence u of input symbols, and Eve has chosen a set of runs over u from the initial states. C is the set of states in which these runs end. Each run induces some delays compared to the longest common prefix of all the outputs they can produce. D maps each state to the delays of the runs ending in it. Eve's actions consist in selecting some of these runs to prevent some delays to grow too high, i.e., she can drop from any set $D(q)$ some of its elements. By restricting the set of possible runs, Eve can be in a situation where some state q of C is accepting while none of the states of $\mathit{live}(D)$ is, in which case she loses, as none of the runs she has selected accepts the input word chosen by Adam. Such vertices constitute the set of *unsafe* vertices she needs to avoid.

More precisely, the set of Adam's *transitions* $T_\forall$ and Eve's transitions $T_\exists$ are defined as follows. From any game position (C, D), Adam can pick a symbol $\sigma \in \Sigma_{\mathtt{i}}$ and the game evolves to the position (C, D, σ). From (C, D, σ), Eve's actions is a subset $\alpha \subseteq \mathit{next}(D, \sigma)$

(she can "drop" some pairs of $next(D, \sigma)$), and the game evolves to (C', D_α) where C' is the set of states reached from C by reading σ, and D_α maps any $q \in Q$ to the set $\{w \mid (q, w) \in \alpha\}$.

Given $K \in \mathbb{N}$, we define the *K-synthesis game* as the restriction of $G_{\mathcal{T}}$ to delays of length at most K: $G_{\mathcal{T},K} = (V_\forall^K, V_\exists^K, v_0^K, A_\forall^K, A_\exists^K, T_\forall^K, T_\exists^K, \mathsf{Safe}^K)$, where $V_\forall^K = 2^Q \times (Q \to 2^{\Sigma_\mathfrak{o}^{\leq K}})$, $A_\exists \subseteq Q \times \Sigma_\mathfrak{o}^{\leq K}$, etc. There is no deadlock in $G_{\mathcal{T},K}$ because Eve can always play $\varnothing$, at the risk of going to an unsafe position.

Example. First, note that by definition of the game, any reachable vertex (C, D) or (C, D, σ) satisfies $live(D) \subseteq C$. Figure 2 represents the 1-synthesis game for $\mathcal{D}_1 \cup \mathcal{D}_2$ (*cf* Figure 1). The states depicted in a vertex correspond to C, together with their values by D (thanks to the previous remark, there is no need to represent the values D assigns for the states outside C). The circle vertices are Eve's positions, whose labels are not depicted, as they are just the label of their predecessor vertex extended with Adam's action. Bold nodes correspond to the unsafe states. Let us now explain how the game proceeds in more detail. First, since both $\mathcal{D}_1$ and $\mathcal{D}_2$ are complete and sequential, for each state (C, D) of Adam, C contains exactly one state of $\mathcal{D}_1$ and one state of $\mathcal{D}_2$. Eve's actions in the synthesis game, which consist in dropping a subset of pairs (state,delay), actually correspond here to "dropping" one of the two sequential transducers: at any moment, she can choose to drop $\mathcal{D}_2$, which leads her into the red part of the game, or to drop $\mathcal{D}_1$, which leads her into the blue part of the game. Note that once she has dropped one of the transducers, she is stuck in the corresponding part.

The initial vertex, owned by Adam, corresponds to being in the initial states of both $\mathcal{D}_1$ and $\mathcal{D}_2$, with no delays. If Adam chooses to play a as the first input letter, Eve has four choices. Either she keeps both transducers, with a delay of length 1, or she drops one of them, or both (not depicted). If Eve chooses to drop $\mathcal{D}_2$, respectively $\mathcal{D}_1$, Adam can then play a b, respectively an a, which leads her into an unsafe state. Note that this proves that Eve cannot win the 0-synthesis game corresponding to $\mathcal{D}_1 \cup \mathcal{D}_2$. If Eve keeps both, she has to drop one of them once Adam plays a second letter, since otherwise the delay would grow larger than 1. However, in both cases she has a move which ensures her a win: if Adam plays a second a, Eve can safely drop $\mathcal{D}_2$ since the accepting state of $\mathcal{D}_1$ has been reached, and if Adam plays b, Eve can safely drop $\mathcal{D}_1$ since the accepting state of $\mathcal{D}_2$ has been reached. If Adam chooses to play a b in the first place, Eve can immediately drop $\mathcal{D}_1$ and win. Hence, Eve wins the 1-synthesis game associated to $\mathcal{D}_1 \cup \mathcal{D}_2$. The described strategy then directly induces a sequential transducer realising the specification.

▶ **Proposition 3.** *Let S be a specification defined by some transducer $\mathcal{T}$. If Eve wins the K-synthesis game $G_{\mathcal{T},K}$ for some K, then S is realisable by a sequential transducer.*

Sketch. If Eve wins the K-synthesis game, then, since it is a safety game, she can win with a *positional* strategy. Thus, her actions only depend on the last visited vertex. This allows to reconstruct a realiser for S, whose states are the possible vertices of Adam visited by the strategy. Then, when Adam chooses an input symbol σ in a vertex (C, D) and Eve decides to go to some vertex (E, F) from (C, D, σ), then in the realiser, we add a transition from (C, D) to (E, F) on σ, outputting the safe output of D for σ. ◀

Synthesis algorithm. It is worth noting that the synthesis game allows for a synthesis procedure: for ascending values of K, test whether Eve wins the K-synthesis game (this can be done in PTime in the size of the game). If it is the case, then by Proposition 3 the specification is realisable, and we can even extract an implementation corresponding to a winning strategy of Eve. If it is not the case, then increment K and try again, until K

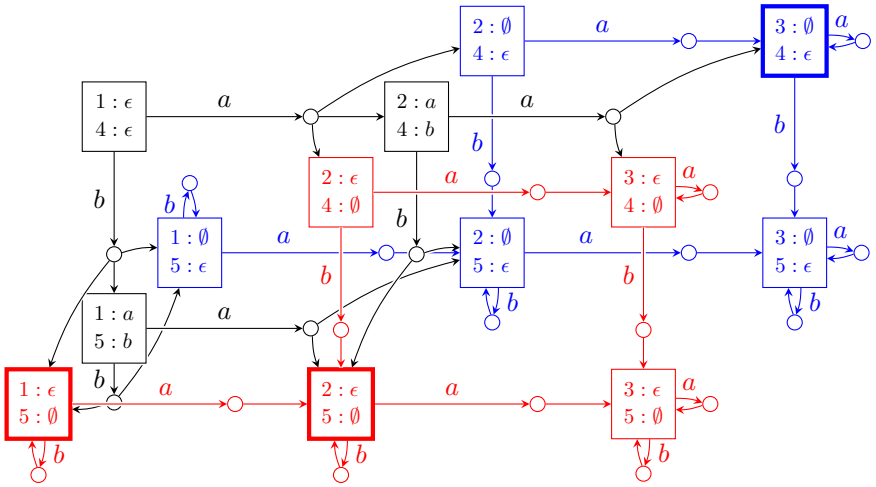


Figure 2 The 1-synthesis game corresponding to the union of $\mathcal{D}_1$ and $\mathcal{D}_2$ (*cf* Figure 1).

reaches some given upper bound B. The K-synthesis game is exponentially large in general (in the transducer defining the specification, and in K). Solving this game efficiently, using for instance symbolic methods, as done for LTL synthesis in the synchronous case [10, 13], is beyond the scope of this paper, but is an interesting research direction.

This algorithm is not complete in general: it is shown for instance in [12] that some specifications defined by transducers are realisable by sequential transducers while Eve has no winning strategy in $G_{\mathcal{T},K}$ for any K. Still, the converse of Proposition 3 holds for some subclasses of specifications $[\![\mathcal{T}]\!]$. For example, in the synchronous setting, where we want to synthesise a synchronous sequential transducer, it suffices to take $K = 0$. This gives an ExpTime procedure to check the realisability of $[\![\mathcal{T}]\!]$ by a synchronous sequential transducer. If $\mathcal{T}$ is *finite-valued*, then by taking K large enough (triply exponential in $\mathcal{T}$), we get completeness [12]. Finally, if $\mathcal{T}$ is functional, then Eve wins $G_{\mathcal{T},K}$ for some K iff $\mathcal{T}$ is equivalent to a sequential transducer, and a polynomial K (in $\mathcal{T}$) suffices [2].

In this paper, we obtain completeness for multi-sequential specifications by taking K exponential in $\mathcal{T}$ (Proposition 6). While this allows us to decide realisability using the game approach, the time complexity will not be optimal (2ExpTime). We indeed devise, in Section 5, a PSpace realisability-checking procedure based on an effective characterisation of realisable multi-sequential specifications. If the PSpace procedure concludes that the specification is realisable, one can run the former game-solving procedure to synthesise a realiser, for ascending values of K. This way, one may hope to synthesise a "small" realiser.

5 The asynchronous setting

We first characterise recursively the multi-sequential specifications which are sequentially realisable (Theorem 5). Then, we provide an equivalent characterisation, non-recursive and easier to check algorithmically, but more technical.

Similarly to the synchronous case, we define a notion of critical situation to which a realiser must react. In the synchronous case, it was just a prefix on which at least two sequential transducers were producing different outputs. In the asynchronous case, two sequential transducers may produce different outputs on the same prefix, but this may not be problematic in the case where one is ahead of the other, i.e., the output of one run is

a prefix of the output of the other. A critical situation is rather a prefix where the delays between all the outputs of the sequential transducers are too large. Since no bound is known a priori to define "too large", we formalise a critical situation as a prefix of the form uv, such that at least two sequential transducers loop on v, and have a different delay before and after the loop. By iterating this loop, i.e. by taking a prefix uv^n, the delay between these two transducers will grow unboundedly when n increases. For such loops, the situation will get critical if a realiser does not react.

▶ **Definition 4** (critical loop). Let $\mathcal{T} = \biguplus_{i=1}^{n} \mathcal{D}_i$ be an n-sequential transducer. A *critical loop* for $\mathcal{T}$ is a triple $(u, v, \mathcal{X}) \in \Sigma_{\mathtt{i}}^* \times \Sigma_{\mathtt{i}}^* \times 2^{\{1,\dots,n\}}$ such that

1. for all $i \in \mathcal{X}$, there exists an initial run $p_i \xrightarrow{u|\alpha_i} q_i \xrightarrow{v|\beta_i} q_i$ of $\mathcal{D}_i$ on uv;
2. for all $i \in \{1, \dots, n\} \setminus \mathcal{X}$, there is no run of $\mathcal{D}_i$ on u;
3. There exists $i, j \in \mathcal{X}$ such that $\mathrm{del}(\alpha_i, \alpha_j) \neq \mathrm{del}(\alpha_i\beta_i, \alpha_j\beta_j)$.

Our characterisation echoes the one of the synchronous setting. It says that whenever there is a critical situation (a critical loop), a realiser must be able to drop some of the sequential transducers, in order to prevent the delays to grow unboundedly, while preserving the residual domain. Formally:

▶ **Theorem 5** (recursive characterisation). *Let $\mathcal{T} = \biguplus_{i=1}^{n} \mathcal{D}_i$ be a multi-sequential transducer over $\Sigma_{\mathtt{i}}, \Sigma_{\mathtt{o}}$. Then $[\![\mathcal{T}]\!]$ is realisable by a sequential transducer, iff, for all critical loops $(u, v, \mathcal{X})$, there exists $\mathcal{Y} \subsetneq \mathcal{X}$ such that*

1. *$\forall i, j \in \mathcal{Y}$, $del(\alpha_i, \alpha_j) = del(\alpha_i\beta_i, \alpha_j\beta_j)$ (following the notations of Definition 4),*
2. *$u^{-1}dom(\mathcal{T}) = \bigcup_{i\in\mathcal{Y}} u^{-1}dom(\mathcal{D}_i)$,*
3. *$\bigcup_{i\in\mathcal{Y}} (u, \ell)^{-1}[\![\mathcal{D}_i]\!]$ is realisable by a sequential transducer, where $\ell = \bigwedge_{i\in\mathcal{X}} \alpha_i$.*

Sketch. $\Rightarrow$ Let $\mathcal{U}$ be a sequential transducer realising $[\![\mathcal{T}]\!]$. For every critical loop $(u, v, \mathcal{X})$ of $\mathcal{T}$, the corresponding set $\mathcal{Y}$ is obtained by getting rid of all the transducers that stray arbitrarily far from $\mathcal{U}$ on the input words of the form uv^*. Then, the first property is immediate, and the other two follow from the fact that $\mathcal{U}$ is sequential and realises $[\![\mathcal{T}]\!]$.

$\Leftarrow$ Conversely, assuming that whenever a critical loop is met there exists a set $\mathcal{Y}$ satisfying the three conditions, we prove by induction on the degree n of sequentiality of $[\![\mathcal{T}]\!]$ that Eve has a winning strategy in the $K_{\mathcal{T}}$-synthesis game for $\mathcal{T}$, for some well-chosen value $K_{\mathcal{T}}$ depending on $\mathcal{T}$. By Proposition 3, this entails the existence of a sequential realiser.

Note that in the synthesis game, since $\mathcal{T}$ is a union of sequential transducers, for each accessible vertex (C, D) of Adam, and for every $i \in \{1, \dots, n\}$, there is at most one state q_i of $\mathcal{D}_i$ occurring in $live(D)$, and if there exists such a state, $|D(q_i)| = 1$. As a consequence, Eve's actions in the synthesis game, which consist in dropping a subset of pairs (state,delay), actually correspond here to "dropping" a subset of sequential transducers.

If $n = 1$, then $\mathcal{T}$ is sequential, and the strategy of Eve that consists in never dropping $\mathcal{T}$ is winning. Now, suppose that $n > 1$. In order to demonstrate that Eve has a winning strategy, we show that for every input word chosen by Adam, either Eve can keep track of all the transducers in the $K_{\mathcal{T}}$-synthesis game, which ensures her a win, or she can drop some transducers on the way, while reaching a state from which she has a winning strategy.

Let $u \in \Sigma_{\mathtt{i}}^*$, and let (C_0, D_0) be the state reached by Eve on input u if she drops nothing. If (C_0, D_0) is not part of the $K_{\mathcal{T}}$-synthesis game, i.e., for some $q \in C_0$, $D_0(q) = \{w\}$ with $|w| > K_{\mathcal{T}}$, this implies the existence of a decomposition $u_1u_2u_3$ of u such that $(u_1, u_2, \mathcal{X})$ is a critical loop for some $\mathcal{X} \subseteq \{1, \dots, n\}$. Then, by hypothesis, there exists a subset $\mathcal{Y} \subsetneq \mathcal{X}$ which satisfies the three conditions of the theorem, hence $\mathcal{T}' = \biguplus_{i\in\mathcal{Y}} (u_1, \ell)^{-1}[\![\mathcal{D}_i]\!]$ is realisable by a sequential transducer. In particular, $\mathcal{T}'$ satisfies the conditions on critical loops (implication

$\Rightarrow$ shown before), and, by the induction hypothesis (since $\mathcal{T}'$ is $|\mathcal{Y}|$-sequential and $|\mathcal{Y}| < n$), Eve has a winning strategy in the $K_{\mathcal{T}'}$-synthesis game for $\mathcal{T}'$ from the initial vertex. Lifting this strategy to the $K_{\mathcal{T}}$-synthesis game for $\mathcal{T}$ yields a winning strategy for Eve from the state (C, D'), where (C, D) is the state reached by Eve on input u_1u_2 if she drops nothing, and D' is obtained from D by dropping all the transducers that are not part of $\mathcal{Y}$. ◀

The proof of Theorem 5 shows that if a multi-sequential specification is realisable, Eve wins the K-synthesis game for K computable from the specification, as stated in Proposition 6. As explained in Section 4, solving the k-synthesis game for ascending values of k then provides a practical way to synthesise a realiser, but the complexity is not optimal.

▶ **Proposition 6** (bounded delay). *Let S be a specification defined by an n-sequential transducer $\mathcal{T}$. Then S is realisable by some sequential transducer iff Eve wins the K-synthesis game for $K = L(6M)^{n^2}$, where L is the longest output occurring on a transition of $\mathcal{T}$, and M is the maximal number of states of a sequential transducer of $\mathcal{T}$.*

▶ **Theorem 7.** *A realisable specification S defined by a multi-sequential transducer $\mathcal{T}$ with m states admits a realiser of size doubly exponential in m. Moreover, there exists a family $(S_n)_{n\in\mathbb{N}}$ of realisable specifications such that for every $n \in \mathbb{N}$, S_n is definable by a multi-sequential transducer of size polynomial in n, and every sequential transducer realising S_n has a size that is doubly exponential in n.*

Proof. Let $S \subset \Sigma^* \times \Gamma^*$ be a realisable specification defined by an n-sequential transducer $\mathcal{T}$ with a set of states Q of size m. Note that $n \leq m$, hence, by Proposition 6, Eve wins the K-synthesis game for some K exponential in m. Then, the construction presented in the proof of Proposition 3 exposes a realiser whose set of states Q' consists of Adam's vertices that are reachable in the K-synthesis game. For every such vertex $(C, D) \in 2^Q \times (Q \to 2^{\Gamma^*})$, since $\mathcal{T}$ is n-sequential, there is at most n sates $q \in Q$ satisfying $D(q) \neq \emptyset$. Moreover, for every such state we have $D(q) = \{w\}$ for some $w \in \Gamma^*$ satisfying $|w| \leq K$. Therefore, the size of Q' is bounded by $2^m(m(|\Gamma|^{K+1}))^n$, which is doubly exponential in m.

In order to expose the family $(S_n)_{n\in\mathbb{N}}$, we use the notion of j-pairs, presented in [22]. For every $n \in \mathbb{N}$, let us consider the alphabet $I_n = \{1, \ldots, n\}$. A *bad j-pair* of a word $u = i_1 \ldots i_m \in I_n^*$ is a pair of positions $1 \leq k < k' \leq m$ such that $i_k = i_{k'} = j$, and for all $k < \ell < k'$, $i_\ell \leq j$. Then every $u \in I_n^*$ satisfying $|u| \geq 2^n$ admits a bad j-pair for some $1 \leq j \leq n$, and there exists a word, denoted by ψ_n, that has size $2^n - 1$, and contains no j-pair (see [22]). We now consider the finite alphabet $\Sigma = \{a, b\}$. For every $n \in \mathbb{N}$, let Σ_n denote the alphabet $I_n \times \Sigma$. We denote by $\pi_1 : \Sigma_n^* \to I_n^*$ and $\pi_2 : \Sigma_n^* \to \Sigma^*$ the projections on the first, respectively second component. Let $f : \Sigma_n^* \to \Sigma^*$ be the function mapping $w \in \Sigma_n^*$ to the word obtained by taking the last letter of $\pi_2(w)$ and putting it at the beginning, i.e., $f(w) = \sigma v$ where $\sigma \in \Sigma$ and $v \in \Sigma^*$ satisfy $\pi_2(w) = v\sigma$. We consider the specification

$$S_n = \{(w, f(w)) | w \in \Sigma_n^*\} \cup \{(w, \epsilon) | w \in \Sigma_n^* \text{ contains a bad } j\text{-pair for some } 1 \leq j \leq n\}.$$

Then S_n is definable by an $(n+2)$-sequential transducers with $3(n+2)$ states, since the function f is definable by the union of 2 sequential transducers of size 3, and for every $1 \leq j \leq n$, the set of words $w \in I_n^*$ containing a bad j-pair is recognisable by a deterministic automaton of size 3. Moreover, since every word $u \in I_n^*$ of size greater than 2^n admits a bad j-pair for some j, S_n is realised by the sequential transducer mapping every word $w \in \Sigma_n^*$ satisfying $|w| < 2^n$ to $f(w)$, and every $w \in \Sigma_n^*$ satisfying $|w| \geq 2^n$ to ϵ.

We now show that every sequential transducer $\mathcal{D}$ realising S_n has at least 2^{2^n-1} states. Let $\mathcal{D} = ((\Sigma, Q, I, F, \Delta), \rho, \tau)$ be a sequential transducer realising S_n. For every $v \in \Sigma^*$ such that $|v| = 2^n - 1$, let $\psi_v \in \Sigma_n^*$ denote the word satisfying $\pi_1(\psi_v) = \psi_n$ and $\pi_2(\psi_v) = v$. We

now show that for every pair of distinct words $v_1, v_2 \in \Sigma^*$ of size $2^n - 1$, the states reached by $\mathcal{D}$ on input ψ_{v_1} and ψ_{v_2} are distinct. This allows us to conclude the proof, since Σ^* contains 2^{2^n-1} such words. Given $v \in \Sigma^*$ satisfying $|v| = 2^n - 1$, let $\rho_v : p_0 \xrightarrow{\psi_v|v'} p_v$ denote the accepting run of $\mathcal{D}$ on input ψ_v. Then $v' = \epsilon$, since if the first letter of v' was an a, $\mathcal{D}$ would not be able to produce an acceptable output on input $\psi_v \cdot (1, b)$, and a similar contradiction would be reached if the first letter of v' was a b. Therefore, the output associated to ψ_v is produced by the terminal function of $\mathcal{D}$, i.e., $\tau(p_v) = f(\psi_v)$. Since f is injective, for every pair of distinct words $v_1, v_2 \in \Sigma^*$ of size $2^n - 1$, $p_{v_1} \neq p_{v_2}$. ◀

We are now ready to show how to decide the realisability of multi-sequential specifications in PSpace. Consider the characterisation given in Theorem 5. We rely on the notion of witness for the non-satisfaction of this characterisation, and we show how to decide the existence of a witness, using a reduction to the emptiness of reversal-bounded counter machines.

The notion of witness intuitively consists in the following ingredients: (1) an unfolding (modeled as a tree) of the recursive characterisation of Theorem 5 and (2) an explicit formulation of delay differences using simple properties of words. Formally, given an n-sequential transducer $\mathcal{T} = \biguplus_{i=1}^{n} \mathcal{D}_i$, where each $\mathcal{D}_i$ is sequential, a *witness* for $\mathcal{T}$ is a finite tree t whose nodes are labelled in $\Sigma_{\mathtt{i}}^* \times \Sigma_{\mathtt{i}}^* \times (2^{\{1,\dots,n\}} \setminus \{\varnothing\})$. For any node x of t, we denote by (u_x, v_x, S_x) its label. For all nodes x, y, z of t, it is required that:

1. (maximality) if x is the root, $S_x = \{1, \dots, n\}$;
2. (consistency) S_x can be split into two disjoint sets N_x, L_x such that for all $i \in N_x$ there is no run of $\mathcal{D}_i$ on u_x, and for all $i \in L_x$ there is a run of $\mathcal{D}_i$ on $u_x v_x$ from its initial state q_0^i, of the form $q_0^i \xrightarrow{u_x|\alpha_{x,i}} p_{x,i} \xrightarrow{v_x|\beta_{x,i}} p_{x,i}$;
3. (monotonicity) if y is a child of x, then $S_y \subsetneq L_x$ and u_x is a prefix of u_y;
4. (partition) if Y is the set of children of x, then $\{S_y \mid y \in Y\}$ partitions L_x;
5. (delays) if y and z are different children of x, for all $i \in S_y$ and $j \in S_z$, either $|\beta_{x,i}| \neq |\beta_{x,j}|$ or, $\beta_{x,i}\beta_{x,j} \neq \epsilon$ and, $\alpha_{x,i}$ and $\alpha_{x,j}$ mismatch[5];
6. (leaves) if x is a leaf, then there is $w \in \Sigma_{\mathtt{i}}^*$ such that $u_x w \in \mathrm{dom}(\mathcal{T})$ and $u_x w \notin \mathrm{dom}(\mathcal{D}_i)$ for all $i \in S_x$.

Intuitively, conditions 2 and 5 require that the words u_x, v_x are critical loops. The delay difference required in the definition of critical loops is not explicit here, but rather replaced by simple properties of words (condition 5), which are easier to check algorithmically. These properties are not strictly equivalent to delay difference, but up to iterating the loop on v_x a sufficient number of times, they are. Conditions $1, 3, 4$ correspond to properties of the subsets met when unfolding the recursive characterisation of Theorem 5. They also allow us to bound linearly the number of nodes of a witness. As announced, all these conditions characterise the unrealisable multi-sequential specifications:

▶ **Lemma 8.** *A multi-sequential specification defined by a trim transducer $\mathcal{T}$ is not realisable by a sequential transducer if and only if there exists a witness for $\mathcal{T}$.*

▶ **Theorem 9.** *The realisability problem by some sequential transducer of a specification defined by a multi-sequential transducer is PSpace-c.*

Sketch. PSpace-hardness has been shown in Section 2. To show PSpace-easyness, we reduce the problem to deciding the emptiness of the language of a counter machine, whose counters make at most 1 reversal (i.e. move from increasing to decreasing mode). This is known to

[5] Two words u, v mismatch if there is a position i such that $i \leq |u|, |v|$ and the ith letter of u differs from the ith letter of v.

be in NLogSpace [16]. Our machine is exponentially large (in the transducer defining the specification), but can be constructed on the fly, hence we get PSpace.

A bit more precisely, we first define the notion of *skeleton* s, which is a witness without the words u_x, v_x, hence there are finitely many skeletons, each one of polynomial size. Given an enumeration $x_1 \dots x_n$ in depth-first order of the nodes of s, we construct a counter machine M_s which recognises sequences of the form $x_1 w_{x_1} \# v_{x_1} \dots x_n w_{x_n} \# v_{x_n}$ such that if we extend any label of a node x in s with the pair of words $(w_{y_1} \dots w_{y_k}, v_x)$, where $y_1 \dots y_k$ is the path from the root to x, we get a witness. Hence, there exists a witness iff there exists a skeleton s such that $L(M_s)$ is non-empty. Our algorithm non-deterministically guesses a skeleton and runs a procedure to check in PSpace the emptiness of M_s.

Let us intuitively explain how M_s works. Conditions $1, 3, 4$ and 6 are regular, so no counter is needed there. Counters are only necessary to check Condition 5, for instance to compute the length of the words $\beta_{x,i}$, and to check the existence of a mismatch between a word $\alpha_{x,i}$ and a word $\alpha_{x,j}$. First, a mismatch position m is guessed, by incrementing for some time two counters $c_{i,x}$ and $c_{j,x}$ in parallel. Then, they are decremented according to the length of outputs produced by simulating the transitions of $\mathcal{D}_i$ and $\mathcal{D}_j$ respectively. When one of them reaches 0, say $c_{i,x}$, we store the mth symbol of the output of $\mathcal{D}_i$ on u_x in memory. We do the same for $c_{j,x}$ and later on check that the two stored symbols are different. ◀

6 Conclusion

We have identified a class of specifications (whose membership is decidable in PTime), for which the sequential realisability problem is PSpace-c, both in the asynchronous and synchronous settings. This is in contrast to the general case, which is ExpTime-c for synchronous specifications, and undecidable in the asynchronous case. Our procedure allows to synthesise a sequential transducer whenever the specification is realisable, and allows for incremental testing, via the solvability of a two-player game parameterised by the longest output allowed to be queued by a realiser before being output.

While the class of multi-sequential specifications is natural, as the closure of graphs of sequential functions under finite unions, we believe that it may also be interesting for practical applications. In particular, Vardi and Lustig have defined the concept of synthesis from component libraries [26], in the synchronous setting, over infinite words. In this setting, given a set of components (synchronous sequential transducers over finite words), a specification S over infinite words, the question is whether the components can be arranged in such a way which realises the specification (by linking the final states of the components to the initial state of another component). This problem was shown to be decidable. We would like to investigate another way of reusing existing components, which is tightly related to multi-sequential specifications: given components $C_1, \dots, C_n$ represented as sequential transducers and a specification S, decide whether there exists a sequential function f such that f and S have the same domain, $f \subseteq \bigcup_i C_i$ and f satisfies S. This is beyond the scope of this paper but we plan to investigate further this question in the near future.

References

1 Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002. doi:10.1145/585265.585270.

2 Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *Theoretical Computer Science*, 292(1):45–63, 2003.

3 Jean Berstel and Luc Boasson. Transductions and context-free languages. *Ed. Teubner*, pages 1–278, 1979.

4 Arnaud Carayol and Christof Löding. Uniformization in Automata Theory. In *Proceedings of the 14th Congress of Logic, Methodology and Philosophy of Science Nancy, July 19-26, 2011*, pages 153–178, London, 2014. College Publications.

5 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy logic. *Information and Computation*, 208(6):677–693, 2010. doi:10.1016/j.ic.2009.07.004.

6 Christian Choffrut and Marcel Paul Schützenberger. Décomposition de fonctions rationnelles. In *2nd Annual Symposium on Theoretical Aspects of Computer Science, STACS*, pages 213–226, 1986.

7 Church, Alonzo. Logic, arithmetic and automata. In *International Congress of Mathematics*, pages 23–35, Stockholm, 1962.

8 Laure Daviaud, Ismaël Jecker, Pierre-Alain Reynier, and Didier Villevalois. Degree of sequentiality of weighted automata. In Javier Esparza and Andrzej S. Murawski, editors, *Proceedings of the 20th International Conference on Foundations of Software Science and Computation Structures, FOSSACS 2017, Uppsala, Sweden, April 22-29*, pages 215–230. Springer Berlin Heidelberg, 2017. doi:10.1007/978-3-662-54458-7_13.

9 Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. Symbolic algorithms for infinite-state games. In *Proceedings of the 12th International Conference in Concurrency Theory, CONCUR 2001, Aalborg, Denmark, August 20-25*, pages 536–550, 2001. doi:10.1007/3-540-44685-0_36.

10 Rüdiger Ehlers. Symbolic bounded synthesis. In *Proceedings of the 22nd International Conference on Computer Aided Verification, CAV 2010, Edinburgh, UK, July 15-19*, volume 6174 of *Lecture Notes in Computer Science*, pages 365–379. Springer, 2010.

11 Samuel Eilenberg. *Automata, Languages, and Machines*. Academic Press, 1974.

12 Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On equivalence and uniformisation problems for finite transducers. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, Rome, Italy*, pages 125:1–125:14, 2016. doi:10.4230/LIPIcs.ICALP.2016.125.

13 Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Exploiting structure in LTL synthesis. *International Journal on Software Tools for Technology Transfer*, 2011.

14 Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Antichains and compositional algorithms for LTL synthesis. *Formal Methods in System Design*, 39(3):261–296, 2011.

15 Wladimir Fridman, Christof Löding, and Martin Zimmermann. Degrees of lookahead in context-free infinite games. In *Computer Science Logic, 25th International Workshop / 20th Annual Conference of the EACSL, CSL 2011, September 12-15, 2011, Bergen, Norway, Proceedings*, pages 264–276, 2011. doi:10.4230/LIPIcs.CSL.2011.264.

16 Eitan M. Gurari and Oscar H. Ibarra. The complexity of decision problems for finite-turn multicounter machines. *Journal of Computer and System Science*, 22(2):220–229, 1981. doi:10.1016/0022-0000(81)90028-3.

17 Michael Holtmann, Lukasz Kaiser, and Wolfgang Thomas. Degrees of lookahead in regular infinite games. In C.-H. Luke Ong, editor, *Proceedings of the 13th International Conference on Foundations of Software Science and Computational Structures, FOSSACS 2010, Paphos, Cyprus, March 20-28*, volume 6014 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 2010.

18 Swen Jacobs, Roderick Bloem, Romain Brenguier, Rüdiger Ehlers, Timotheus Hell, Robert Könighofer, Guillermo A. Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup, and Adam Walker. The first reactive synthesis competition (SYNTCOMP 2014). *STTT*, 19(3):367–390, 2017. doi:10.1007/s10009-016-0416-3.

19 Ismaël Jecker and Emmanuel Filiot. Multi-sequential word relations. In *Proceedings of the 19th International Conference on Developments in Language Theory, DLT 2015, Liverpool, UK, July 27-30*, pages 288–299, 2015. doi:10.1007/978-3-319-21500-6_23.

20 B. Jobstmann, S. Galler, M. Weiglhofer, and R. Bloem. Anzu: A tool for property synthesis. In *Computer Aided Verification, CAV*, pages 258–262, 2007.

21 J.R. Büchi and L.H. Landweber. Solving sequential conditions finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

22 Felix Klein and Martin Zimmermann. How much lookahead is needed to win infinite games? *Logical Methods in Computer Science*, 12(3), 2016. doi:10.2168/LMCS-12(3:4)2016.

23 Felix Klein and Martin Zimmermann. Prompt delay. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, Chennai, India*, pages 43:1–43:14, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.43.

24 Kojiro Kobayashi. Classification of formal languages by functional binary transductions. *Information and Control*, 15(1):95–109, 1969.

25 Dexter Kozen. Lower bounds for natural proof systems. In *FOCS*, pages 254–266. IEEE Computer Society, 1977. URL: http://dblp.uni-trier.de/db/conf/focs/focs77.html#Kozen77.

26 Yoad Lustig and Moshe Y. Vardi. Synthesis from component libraries. *STTT*, 15(5-6):603–618, 2013.

27 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *ACM Symposium on Principles of Programming Languages, POPL*. ACM, 1989.

28 Sven Schewe and Bernd Finkbeiner. Bounded synthesis. In *Automated Technology for Verification and Analysis*, volume 4762 of *Lecture Notes in Computer Science*, pages 474–488. Springer Berlin Heidelberg, 2007.

29 Wolfgang Thomas. Church's problem and a tour through automata theory. In *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *Lecture Notes in Computer Science*, pages 635–655. Springer, 2008.

30 Martin Zimmermann. Delay games with WMSO+U winning conditions. *RAIRO - Theoretical Informatics and Applications*, 50(2):145–165, 2016. doi:10.1051/ita/2016018.

31 Martin Zimmermann. Finite-state strategies in delay games. In *Proceedings 8th International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2017, Roma, Italy, 20-22 September*, pages 151–165, 2017. doi:10.4204/EPTCS.256.11.

32 Martin Zimmermann. Games with costs and delays. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005125.

Pricing Problems with Buyer Preselection

Vittorio Bilò
Univ. of Salento, Lecce, Italy
vittorio.bilo@unisalento.it

Michele Flammini
GSSI Institute, L'Aquila, Italy & Univ. of L'Aquila, L'Aquila, Italy
michele.flammini@univaq.it

Gianpiero Monaco
Univ. of L'Aquila, L'Aquila Italy
gianpiero.monaco@univaq.it

Luca Moscardelli
Univ. of Chieti-Pescara, Pescara, Italy
luca.moscardelli@unich.it

Abstract

We investigate the problem of preselecting a subset of buyers participating in a market so as to optimize the performance of stable outcomes. We consider four scenarios arising from the combination of two stability notions, item and bundle envy-freeness, with the two classical objective functions, i.e., the social welfare and the seller's revenue. When adopting the notion of item envy-freeness, we prove that, for both the two objective functions, the problem cannot be approximated within $n^{1-\varepsilon}$ for any $\varepsilon > 0$, and provide tight or nearly tight approximation algorithms. We also prove that maximizing the seller's revenue is NP-hard even for a single buyer, thus closing an open question. Under bundle envy-freeness, instead, we show how to transform in polynomial time any stable outcome for a market involving only a subset of buyers to a stable one for the whole market without worsening its performance, both for the social welfare and the seller's revenue. Finally, we consider multi-unit markets, where all items are of the same type and are assigned the same price. For this specific case, we show that buyer preselection can improve the performance of stable outcomes in all of the four considered scenarios, and we design corresponding approximation algorithms.

2012 ACM Subject Classification Theory of computation → Computational pricing and auctions

Keywords and phrases Pricing problems, Envy-freeness, Revenue maximization, Social Welfare maximization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.47

Related Version A two-page extended abstract [8] of a preliminary version of this paper appeared in the proceedings of AAMAS 2018.

1 Introduction

Determining an efficient pricing strategy is a fundamental problem in many business activities, as it affects both the seller's revenue and the customers' or buyers' satisfaction. Usually, optimal prices are the result of a challenging counterbalancing process: selecting low prices, for instance, may be profitable for the seller when it attracts considerably more customers, but, at the same time, in case of limited supply, it may leave some buyers unsatisfied, thus

generating discontent. In particular, this happens when a customer is negated the right to buy her preferred set of items, or even any item at all, despite the fact that she is willing to pay for the posted prices. In this case she is often called *loser*, as opposed to a customer receiving items, called *winner*. For such a reason, pricing problems are traditionally considered under the hypothesis of **envy-freeness** [25, 32], which prescribes that, once a pricing strategy has been established, items have to be allocated to buyers in such a way that no one would prefer a different set of items.

However, if from the one hand safeguarding the losers' interests shelters the seller from possible future losses due to their dissatisfaction, on the other hand, a result by [7] shows that, in certain markets, an intrinsic and unavoidable hurdle to the construction of a good quality envy-free solution may come from the presence of a set of "disturbing" customers, that is, a set of buyers such that at least one of them gets envious in any assignment of sufficiently high revenue. This observation naturally leads to the following intriguing question: *"What happens if envy-freeness is restricted to apply only to the set of winners? Can the seller raise enough more revenue (with certainty) today to compensate the (uncertain) future loss of potential customers?"*. Such a relaxed form of envy-freeness models indeed the situation in which the seller is allowed the freedom to discard any subset of buyers from the given instance, so as to get rid of envious losers in the assignment she would like to propose. In addition, one might consider such buyers "preselection" as a situation in which the seller advertises about the existence of the market only targeted buyers that, once involved, will all consider the allocation fair; the excluded ones then are simply unaware and won't feel any unfairness.

1.1 Our Contribution

Motivated by the above discussion, we introduce and investigate the *buyer preselection problem* in which, given a pricing problem P with n buyers and m items, we are interested in computing the best possible envy-free solution that can be achieved by removing any arbitrary subset of buyers from P. We consider four scenarios arising from the combination of two stability notions, called item and bundle envy-freeness, respectively, with the two classical objective functions, namely, the social welfare and the seller's revenue.

In an item envy-free allocation, given a pricing of the items, each buyer gets the subset maximizing her utility among all possible subsets that can be created from the set of available items; in a bundle envy-free allocation, no buyer gets a better utility by receiving the bundle allocated to any winner. Observe that these allocations are always guaranteed to exist, as it suffices to assign all items an arbitrarily high price, so that no winner is possible.

For item envy-free allocations and both objective functions, we show that the buyer preselection problem cannot be approximated within $n^{1-\epsilon}$ for every $\epsilon > 0$, unless $\mathsf{P} = \mathsf{ZPP}$. On the positive side, under the objective of social welfare, we design an n-approximation algorithm, while, for the case of revenue maximization, we give an $O(n \log m)$-approximation. In particular, these results are obtained as follows: all but one buyer are discarded from the given instance, so that we are left with a pricing problem with a single buyer. While such a problem is solvable in polynomial time under the objective of social welfare, for revenue maximization, it already exhibits challenging combinatorial structures and, to the best of our knowledge, has been considered before only in [5]. In this paper an $O(\log m)$-approximation is provided, but no lower bounds on the problem complexity are given. We show that the problem is NP-hard, thus solving the corresponding longstanding open problem raised by the authors.

We stress that efficient preselection can be profitable under two orthogonal directions: from the one hand, the removal of a subset of pathological envious buyers may increase the value of the optimal solution; from the other hand, even when this does not happen or it has only a modest impact, simplifying the combinatorial structure defined by the valuation functions of the winners may lead to the design of better approximation algorithms. In fact, for the two preselection problems obtained by considering bundle envy-free allocations, we show how to transform in polynomial time any allocation which is bundle envy-free only for the subset of the winners to a bundle envy-free allocation for all buyers without worsening its performance. Hence, although this transformation implies that, in this case, buyer preselection cannot improve the performance of stable outcomes, it can be used to map any bundle envy-free allocation for a subset of winners obtained through preselection back to a bundle envy-free allocation involving all buyers. Therefore, it can be used as an algorithmic tool for computing good stable outcomes when preselection is not allowed. In fact, it can be first exploited to simplify the combinatorics of the problem, and then for mapping back the computed solution to one encompassing all the buyers.

To this respect, consider for instance the case in which buyers can be partitioned in two (or more) sets A and B (each of which, for instance, containing only unit-demand buyers or only single minded ones or only multi-demand buyers with additive valuations), and such that (i) valuations of players in A and B, if considered separately, allow to compute an r_A-approximate (resp. r_B-approximate) solution of the problem restricted to set A (resp. B) by exploiting known or new simpler algorithms/techniques and (ii) considering the whole instance would make difficult the direct application of such algorithms/technique for solving P. By performing preselection we can easily obtain an $r = 2\max(r_A, r_B)$-approximation for BP(P) (selecting the best solution among the one only for A and the one only for B), and thus, by the transformation, also for the initial problem P.

Finally, we consider the multi-unit case, where all items are of the same type and are assigned the same price. We show how preselection can improve the revenue and the social welfare of both item and bundle envy-free solutions. In particular, for item envy-free allocations, we show a tight multiplicative factor of m for both objective functions. For bundle envy-free allocations, we show a lower multiplicative bound of 2 for both the revenue and the social welfare, and prove that it is tight for the objective of revenue maximization. We also provide tight results on the complexity of computing optimal solutions for the buyer preselection problem under envy-freeness.

Due to space limitations, some proofs are only sketched or omitted.

1.2 Related Work

The literature on envy-free pricing is so vast that it cannot be exhaustively covered here. For such a reason, we simply refer to the achievements which are mostly related to the model of [28] we consider in this paper.

For the social welfare maximization, the VCG mechanism [33, 19, 26] provides an optimal solution to the envy-free pricing problem. However, while this mechanism is efficiently computable in markets with unit-demand buyers, yet for single-minded ones its computation becomes NP-hard. Approximate solutions are still possible in this case thanks to the results of [4]. Also Walrasian Equilibria [34] provide an optimal solution to the problem [6]; however, they are guaranteed to exist only under very stringent hypothesis on the buyers' valuation functions [27].

For the revenue maximization, [28, 29, 18, 5, 10] design logarithmic approximation algorithms for various special cases of the problem. Relative hardness results have been given by [9, 11, 13, 12, 20]. Further variants have been considered by [14, 16, 22, 3, 7, 15].

[23] propose an interesting relaxation of the notion of Walrasian Equilibrium, called Combinatorial Walrasian Equilibrium (CWE), obtained by grouping items into bundles so as to induce a "reduced market" to which, then, applying the notion of Walrasian Equilibrium. They show the existence of a CWE yielding a 2-approximation of the optimal social welfare and that of a CWE yielding a logarithmic approximation of the optimal revenue.

Furthermore, [31] study the case of revenue maximization in markets with multi-unit items under both item and bundle envy-freeness when allowing both item and bundle pricing. Such setting has been extended in [24] where the authors consider a social graph of the buyers and envies can arise only between neighbors.

In our model, by preselecting buyers, we basically require that all the winners are envy-free. Settings in which, in a similar way, envy-freeness is not guaranteed for all buyers, but only for the winners, are studied in [1, 2, 17]. In particular, [17] considers "weak" Walrasian equilibirum, a relaxed version of Walrasian equilibirum in which the goal is that of maximizing the number of envy-free buyers, with the condition that all the winners must be envy-free. [1, 2] consider a relaxed version of envy-freeness: in their model, identical items have to be sold to buyers, with every buyer constituting a node of a given unweighted graph; adjacent winning buyers have to pay similar prices for the received item, while the losers cannot envy. This feature is exploited to achieve higher revenue with respect to the classical case in which there cannot be losers, even if it makes the computational problem harder.

2 Model and Definitions

2.1 Markets

A *market* is a tuple $\Gamma = (N, M, (v_i)_{i \in N})$, where N is a set of n buyers, M is a set of m items, and for every buyer $i \in N$, $v_i : 2^M \to \mathbb{R}_{\geq 0}$ is a *valuation function* expressing, given a set of items $X \subseteq M$, the amount of money that buyer i is willing to pay for X; we assume that $v_i(\emptyset) = 0$ for every buyer $i \in N$.

Depending on the definition of the valuation functions, different types of markets can be modeled. In the most general case, called *market with combinatorial valuations*, function v_i is completely arbitrary for every buyer $i \in N$. In a *market with unit-demand buyers*, $v_i(X) = 0$ for every $i \in N$ and $X \subseteq M$ with $|X| > 1$, that is, every buyer is only interested in singleton sets. In a *market with single-minded buyers*, for every $i \in N$, there exists a unique set of items $X \subseteq M$ such that $v_i(X) \neq 0$, that is, every buyer is only interested in a particular set of items. In a *market with additive valuations*, $v_i(X) = \sum_{j \in X} v_i(\{j\})$ for every $i \in N$ and $X \subseteq M$. We stress that, while the representation of a market with combinatorial valuations may require $\Omega(n2^m)$ bits, $\Theta(nm)$ bits suffice to represent the last three types of markets. Finally, in a *market with multi-unit items*, all the m items are of the same type and so, for every $i \in N$, the valuation function becomes of the form $v_i : \{0, 1, \ldots, m\} \to \mathbb{R}_{\geq 0}$, since it is only required to specify how much a buyer evaluates a set of k items, for every $k \in \{1, \ldots, m\}$ (clearly, $v_i(0) = 0$); thus, also in this case, the market can be represented with $\Theta(nm)$ bits.

2.2 Stable Outcomes

Fix a market Γ. A *price vector* is an m-tuple $\mathbf{p} = (p_1, \ldots, p_m)$ such that, for every $j \in M$, $p_j \geq 0$ is the price of item j.[1] We denote by $\mathbf{0}^m$ the price vector assigning price 0 to all items. Given a price vector $\mathbf{p}$ and a set of items $X \subseteq M$, $u_i(X, \mathbf{p}) = v_i(X) - \sum_{j \in X} p_j$ is

[1] For the case of markets with multi-unit items, it is only required to fix the price of a single item so that vector $\mathbf{p}$ collapses to a real number $p \geq 0$.

the *utility* of buyer i when buying X. The *demand set* of buyer i for the price vector $\mathbf{p}$ is the set $D_i(\mathbf{p}) = \text{argmax}_{X \subseteq M} u_i(X, \mathbf{p})$ of subsets of items maximizing i's utility according to the prices specified by $\mathbf{p}$. An *allocation vector* is an n-tuple $\mathbf{X} = (X_1, \ldots, X_n)$ such that $X_i \subseteq M$ is the set of items sold to buyer i. The allocation vector $\mathbf{X} = (X_1, \ldots, X_n)$ is *feasible* if $X_i \cap X_{i'} = \emptyset$ for each $i \neq i' \in N$. An *outcome* is a pair $(\mathbf{X}, \mathbf{p})$ such that $\mathbf{X}$ is feasible. Denote with $\mathsf{OUT}(\Gamma)$ the set of outcomes of Γ. An outcome $(\mathbf{X}, \mathbf{p})$ is *individually-rational* if $u_i \geq 0$ for every $i \in N$.

Denote as $M(\mathbf{X}) = \bigcup_{i \in N} X_i$ the set of items sold to some buyer according to a feasible allocation vector $\mathbf{X}$. Buyer i is a *winner* if $X_i \neq \emptyset$ and $W(\mathbf{X})$ denotes the set of all winners in $\mathbf{X}$. For an item $j \in M(\mathbf{X})$, denote with $b_{\mathbf{X}}(j)$ the buyer $i \in W(\mathbf{X})$ such that $j \in X_i$. When the allocation vector is clear from the context, we simply write $b(j)$.

The following concepts define two types of stable outcomes for Γ.

▶ **Definition 1.** An individually-rational outcome $(\mathbf{X}, \mathbf{p})$ is **item envy-free** if $u_i(X_i, \mathbf{p}) \geq u_i(T, \mathbf{p})$ for every buyer $i \in N$ and $T \subseteq M$, that is, $X_i \in D_i(\mathbf{p})$ for every $i \in N$.

▶ **Definition 2.** An individually-rational outcome $(\mathbf{X}, \mathbf{p})$ is **bundle envy-free** if $u_i(X_i, \mathbf{p}) \geq u_i(X_j, \mathbf{p})$ for every two buyers $i, j \in N$.

Denote with $\mathsf{IEF}(\Gamma)$ and $\mathsf{BEF}(\Gamma)$, the sets of item envy-free and bundle envy-free outcomes for Γ, respectively. Notice that $\mathsf{IEF}(\Gamma) \subseteq \mathsf{BEF}(\Gamma)$ and $\mathsf{IEF}(\Gamma) \neq \emptyset$ (and so also $\mathsf{BEF}(\Gamma) \neq \emptyset$), since the outcome $(\mathbf{X}, \mathbf{p})$ such that $X_i = \emptyset$ for every $i \in N$ and $p_j = \infty$ for every $j \in M$ is individually-rational and item envy-free.

2.3 Pricing Problems

Fix an outcome $(\mathbf{X}, \mathbf{p})$ for a market Γ. The revenue raised by $(\mathbf{X}, \mathbf{p})$ is $\mathsf{REV}(\mathbf{X}, \mathbf{p}) = \sum_{j \in M(\mathbf{X})} p_j$. The social welfare generated by $(\mathbf{X}, \mathbf{p})$ is $\mathsf{SW}(\mathbf{X}, \mathbf{p}) = \sum_{i \in N} u_i(X_i) + \sum_{j \in M(\mathbf{X})} p_j = \sum_{i \in W(\mathbf{X})} v_i(X_i)$. Note that the social welfare does not depend on the price vector $\mathbf{p}$ and that $\mathsf{SW}(\mathbf{X}, \mathbf{p}) \geq \mathsf{REV}(\mathbf{X}, \mathbf{p})$.

Given a market Γ, let $\mathsf{sol}(\Gamma) \subseteq \mathsf{OUT}(\Gamma)$ denote any subset of outcomes for Γ and $\mathsf{obj} : \mathsf{OUT}(\Gamma) \to \mathbb{R}_{\geq 0}$ denote an objective function associating a non-negative value to every outcome for Γ. Let $opt(\Gamma, \mathsf{sol}, \mathsf{obj}) := \text{argmax}_{(\mathbf{X}, \mathbf{p}) \in \mathsf{sol}(\Gamma)} \{\mathsf{obj}(\mathbf{X}, \mathbf{p})\}$ denote the set of outcomes in $\mathsf{sol}(\Gamma)$ maximizing the objective function obj.

▶ **Definition 3.** The **pricing problem** $P = (\Gamma, \mathsf{sol}, \mathsf{obj})$ is an optimization problem which , given a market Γ, a set of outcomes $\mathsf{sol}(\Gamma)$ and an objective function obj, asks for an outcome $o^*(P) \in opt(P)$.

In this paper, we consider the cases in which $\mathsf{sol}(\Gamma) \in \{\mathsf{IEF}(\Gamma), \mathsf{BEF}(\Gamma)\}$ and $\mathsf{obj} \in \{\mathsf{REV}, \mathsf{SW}\}$.

Oracles. Fix a pricing problem $P = (\Gamma, \mathsf{sol}, \mathsf{obj})$. As we have seen, when Γ is a market with combinatorial valuations, any algorithm for P needs to deal with an input of exponential size. In order to circumvent this problem and remain within the realm of polynomial time algorithms, it is usually assumed that functions v_is are not given as an input of the problem and are replaced by a polynomial time (with respect to n and m) *oracle* providing information about a buyer's valuation function. An oracle is usually assumed to answer two types of questions: a *value query* which, given a buyer $i \in N$ and a set of items X, returns the valuation $v_i(X)$, and a *demand query* which, given a price vector $\mathbf{p}$ and a buyer $i \in N$, returns any set in $D_i(\mathbf{p})$.

We remark that all algorithms of this paper exploiting oracle calls are polynomial also in the sense that they call the oracle a polynomial number of times. Therefore, when also the oracle is polynomially computable, the algorithm computation is fully polynomial.

2.4 The Buyer Preselection Problem

Given a market $\Gamma = (N, M, (v_i)_{i \in N})$ and a subset of buyers $N' \subseteq N$, the submarket of Γ induced by N' is the market $\Gamma(N') = (N', M, (v_i)_{i \in N'})$.

▶ **Definition 4.** The **buyer preselection problem** is an optimization problem $\mathsf{BP}(P)$ which, given a pricing problem $P = (\Gamma, \mathsf{sol}, \mathsf{obj})$ with $\Gamma = (N, M, (v_i)_{i \in N})$, asks for a pair $(N^*(\mathsf{BP}(P)), o^*(\mathsf{BP}(P)))$ such that $N^*(\mathsf{BP}(P)) \in \operatorname{argmax}_{N' \subseteq N}\{opt(\Gamma(N'), \mathsf{sol}, \mathsf{obj})\}$ and $o^*(\mathsf{BP}(P)) \in opt(\Gamma(N^*(\mathsf{BP}(P))), \mathsf{sol}, \mathsf{obj})$, that is, $o^*(\mathsf{BP}(P))$ is the best outcome which can be realized in all possible submarkets of Γ.

Clearly, by definition, for every pricing problem P, $\mathsf{obj}(o^*(P)) \leq \mathsf{obj}(o^*(\mathsf{BP}(P)))$, that is, buyer preselection can only improve the quality of the optimal solution.

3 Results for Item Envy-Free Outcomes

In this section, we consider the buyer preselection problem $\mathsf{BP}(\Gamma, \mathsf{IEF}, \mathsf{obj})$ with $\mathsf{obj} \in \{\mathsf{REV}, \mathsf{SW}\}$. We start by providing a lower bound on its approximability attained by exploiting an approximation-preserving reduction from the maximum independent set problem.

▶ **Theorem 5.** *Let $P = (\Gamma, \mathsf{IEF}, \mathsf{obj})$ be a pricing problem with $\mathsf{obj} \in \{\mathsf{REV}, \mathsf{SW}\}$. For every $\epsilon > 0$, the buyer preselection problem $\mathsf{BP}(P)$ cannot be approximated within $n^{1-\epsilon}$, unless $\mathbf{P} = \mathbf{ZPP}$, even when Γ is a market with single-minded buyers.*

Proof. We prove the claim through an approximation-preserving reduction from the maximum independent set problem, in which, given an undirected graph, it is asked for a subset of nodes, no two of which are adjacent, of maximum cardinality. To this aim, consider an instance of the maximum independent set problem defined by a graph $G = (V, E)$ and denote with $\delta_i(G)$ the set of edges incident to node i. We create a market $\Gamma = (N, M, (v_i)_{i \in N})$ with single-minded buyers as follows: we set $N = V$, $M = E$ and, for every $i \in N$, we define the valuation function v_i in such a way that, for every $X \subseteq M$,

$$v_i(X) = \begin{cases} 1 & \text{if } X = \delta_i(G) \\ 0 & \text{otherwise.} \end{cases}$$

Fix any subset of buyers $N' \subseteq N$. Given an individually-rational outcome $o = (\mathbf{X}, \mathbf{p}) \in \mathsf{OUT}(\Gamma(N'))$, define $V_{\mathsf{REV}}(o) := \{i \in V : \sum_{j \in X_i} p_j > 0\}$ and $V_{\mathsf{SW}}(o) := \{i \in V : v_i(X_i) > 0\}$. By construction of the valuation functions, both $V_{\mathsf{REV}}(o)$ and $V_{\mathsf{SW}}(o)$ have to be independent sets for G. Let $\mathsf{obj} \in \{\mathsf{REV}, \mathsf{SW}\}$. Since $\sum_{j \in X_i} p_j \leq 1$ and $v_i(X_i) \leq 1$ for every $i \in N'$, it follows that

$$\mathsf{obj}(o) \leq |V_{\mathsf{obj}}(o)|. \tag{1}$$

Let $V^* \subseteq V$ be a maximum independent set for G. It is easy to see that the outcome $(\mathbf{X}^*, \mathbf{p}^*)$ such that $X_i^* = \delta_i(G)$ for every $i \in V^*$ and

$$p_j^* = \begin{cases} 1/\delta_i(G) & \text{if } j \in \delta_i(G) \\ 0 & \text{otherwise.} \end{cases}$$

is an item envy-free outcome (actually it is also a Walrasian equilibrium because the market clears, i.e, every unsold item is assigned price zero) for market $\Gamma(N')$. This implies

$$\mathsf{obj}(o^*(\mathsf{BP}(P))) \geq \mathsf{obj}(\mathbf{X}^*, \mathbf{p}^*) = |V^*|. \tag{2}$$

Assume, for the sake of contradiction, that there exists an approximation algorithm for $\mathsf{BP}(P)$ returning an outcome o such that $n^{1-\epsilon}\mathsf{obj}(o) \geq \mathsf{obj}(o^*(\mathsf{BP}(P)))$ for some $\epsilon > 0$. Using (2), we get $n^{1-\epsilon}\mathsf{obj}(o) \geq |V^*|$ which combined with (1) implies that $n^{1-\epsilon}|V_{\mathsf{obj}}(o)| \geq |V^*|$: a contradiction to the inapproximability result for the maximum independent set problem. ◀

As a positive result, we show that, by building upon (approximation) algorithms for pricing problems defined on markets with a unique buyer, it is possible to obtain approximation algorithms for the buyer preselection problem. In particular, the following theorem can be proved by considering the buyer providing, when being alone in the market, the best possible outcome with respect to the considered objective function.

▶ **Lemma 6.** *Given a buyer preselection problem* $\mathsf{BP}(P)$*, where* $P = (\Gamma, \mathsf{IEF}, \mathsf{obj})$ *with* $\mathsf{obj} \in \{\mathsf{REV}, \mathsf{SW}\}$*, if there exists a polynomial time algorithm* $\mathcal{A}$ *returning an outcome for the pricing problem defined on markets with a unique buyer whose objective value is at least an* α *fraction of the optimal social welfare, then* $\mathsf{BP}(P)$ *admits an* αn*-approximation algorithm.*

Proof. Assume that $o^*(\mathsf{BP}(P)) := (\mathbf{X}^*, \mathbf{p}^*)$. For every $i \in N^*(\mathsf{BP}(P))$, define

$$\mathsf{obj}_i(o^*(\mathsf{BP}(P))) = \begin{cases} \sum_{j \in X_i^*} p_j^* & \text{if } \mathsf{obj} = \mathsf{REV}, \\ v_i(X_i^*) & \text{if } \mathsf{obj} = \mathsf{SW}. \end{cases} \tag{3}$$

Consider the preselection algorithm which, given Γ, returns a buyer $i^* \in N$ such that $i^* \in \mathrm{argmax}_{i \in N} \{\max_{X \subseteq M}\{v_i(X)\}\}$. Let $o \in \mathsf{IEF}(\Gamma(\{i^*\}))$ be the outcome returned by $\mathcal{A}$ when executed on the pricing problem $P' = (\Gamma(\{i^*\}), \mathsf{IEF}, \mathsf{obj})$. We have

$$\begin{aligned} \mathsf{obj}(o^*(\mathsf{BP}(P))) &= \sum_{i \in N^*(\mathsf{BP}(P))} \mathsf{obj}_i(o^*(\mathsf{BP}(P))) \\ &\leq \sum_{i \in N^*(\mathsf{BP}(P))} v_i(X_i^*) \\ &\leq n v_{i^*}(X_{i^*}^*) \\ &\leq \alpha n \mathsf{obj}(o), \end{aligned}$$

where the first inequality comes from (3) and the fact that $\sum_{j \in X_i^*} p_j \leq v_i(X_i^*)$ because of individual rationality, the second inequality follows from the definition of i^*, and the third inequality comes from the hypothesis on algorithm $\mathcal{A}$. ◀

As a consequence of Theorem 5 and Lemma 6, we have that the buyer preselection problem $\mathsf{BP}(\Gamma, \mathsf{IEF}, \mathsf{obj})$, with $\mathsf{obj} \in \{\mathsf{REV}, \mathsf{SW}\}$, admits a polynomial time algorithm providing the best possible approximation guarantee, whenever the pricing problem defined on markets with a unique buyer can be solved in polynomial time with respect to the social welfare objective function. For such a reason, in the following subsection, we focus on the solution of the latter problem.

3.1 Pricing Problems Defined on Markets with a Unique Buyer

Throughout this subsection, since there is only one buyer in the market, for the sake of simplicity we remove the pedis 1 from the notation.

As a warmup, we start by considering the simpler case in which $\mathsf{obj} = \mathsf{SW}$.

▶ **Claim 7.** *Let Γ be a market with a single buyer and combinatorial valuations. The pricing problem $(\Gamma, \mathsf{IEF}, \mathsf{SW})$ can be solved in polynomial time.*

In fact, observe that an outcome $(X^*, \mathbf{0}^m)$ such that $X^* \in \operatorname{argmax}_{X \subseteq M} v(X)$ verifies $(X^*, \mathbf{0}^m) \in opt(P)$, and a set $X^* \in \operatorname{argmax}_{X \subseteq M} v(X)$ can be obtained in polynomial time by using the price vector $\mathbf{0}^m$ as the input of an oracle demand query.

We show in the next theorem that the case of $\mathsf{obj} = \mathsf{REV}$ yields an **NP**-hard problem, thus solving a longstanding open problem left by [5] (where in Lemma 7 an approximation algorithm with no hardness result is provided). Theorem 8 can be proved by exploiting a polynomial reduction from 3SAT, in which a given boolean formula ϕ is transformed into a market with a unique buyer, whose items are the literals of ϕ.

▶ **Theorem 8.** *Let Γ be a market with a single buyer and combinatorial valuations. The pricing problem $(\Gamma, \mathsf{IEF}, \mathsf{REV})$ is* **NP***-hard.*

Proof. We prove the claim through a reduction from 3SAT. To this aim, given a boolean formula ϕ, let $\mathsf{V}(\phi)$ denote the set of its variables and $\mathsf{L}(\phi)$ the set of all possible literals on variables in $\mathsf{V}(\phi)$; moreover, denote $\nu = |\mathsf{V}(\phi)|$. Throughout this proof, we assume $\nu \geq 4$. An assignment for ϕ is a function $f : \mathsf{V}(\phi) \to \{0, 1\}$ assigning to each variable of ϕ a boolean value. Denote with $F(\phi)$ the set of all possible assignments for ϕ and with $\phi(f)$ the boolean value obtained by evaluating all literals occurring in ϕ according to f. ϕ is satisfiable if there exists an assignment $f \in F(\phi)$ such that $\phi(f) = 1$ and it is unsatisfiable if, for every assignment $f \in F(\phi)$, $\phi(f) = 0$. Given a set of literals $X \subseteq \mathsf{L}(\phi)$, with a little abuse of notation, we write $X \in F(\phi)$ whenever there exists an assignment $f \in F(\phi)$ such that X contains all and only those literals which are evaluated 0 according to f. A formula ϕ is an instance of 3SAT if ϕ is expressed in Conjunctive Normal Form and each clause is the disjunction of 3 literals, so that ϕ can be completely expressed by listing the set $C = \{c_1, \ldots, c_k\}$ of its clauses, where each clause c_i $(i = 1, \ldots, k)$ is a set of three literals.

Given an instance of 3SAT $\phi := C$, we construct a market $\Gamma = (\{1\}, M, v)$ with a unique buyer such that $M = \mathsf{L}(\phi)$ and the valuation function v is defined as follows:

$$v(X) = \begin{cases} 1 & \text{if } |X| = 1, \\ 3 + \epsilon & \text{if } X \in C, \\ \nu & \text{if } X \in F(\phi), \\ 0 & \text{otherwise,} \end{cases}$$

where $\epsilon > 0$ is arbitrarily small.

Clearly Γ can be constructed in polynomial time with respect to the representation of ϕ. However, in order to complete the reduction, we have to construct an oracle which can answer both demand and value queries in polynomial time. To this aim, observe that, given a set of literals X, checking whether X is a singleton set, or $X \in C$, or $X \in F(\phi)$ can be performed in polynomial time, so that value queries can be efficiently answered. In order to provide an efficient answer to a demand query, we first observe that the cardinalities of sets $\mathsf{L}(\phi)$ and C are polynomial in the representation of ϕ, and therefore, given a pricing vector, a set of items in $\mathsf{L}(\phi)$ and C yielding the highest utility can be efficiently computed by enumeration. Then, in order to compute a set $X \in F(\phi)$ of maximum utility, note that all

the candidate sets have the same valuation so that, in order to return one with the highest utility, we simply need to choose, for each variable in $\mathsf{V}(\phi)$, the related literal having the lowest price. Hence, market Γ can be generated and managed in polynomial time.

Now, in order to complete the proof, we show that there exists an outcome $o \in \mathsf{OUT}(\Gamma)$ such that $\mathsf{REV}(o) = \nu$ if and only if ϕ is satisfiable.

Assume first that ϕ is satisfiable and let f be a satisfying assignment for ϕ. Let X be the set of literals which are evaluated 0 in f and let $\mathbf{p}$ be the pricing vector such that all literals in X are priced 1, while all literals in $\mathsf{L}(\phi) \setminus X$ are priced ∞. By definition, we have $u(X, \mathbf{p}) = 0$, so that $(X, \mathbf{p})$ is individually-rational. Moreover, $(X, \mathbf{p}) \in D(\mathbf{p})$. In fact, for every $X' \neq X$ such that $X' \in F(\phi)$, $u(X', \mathbf{p}) < 0$ since X' has to contain at least one item priced ∞; for every $X' \in C$, $u(X', \mathbf{p}) < 0$ since, as we have $\phi(f) = 1$, X' has to contain at least one item priced ∞; for every $X' \in \mathsf{L}(\phi)$, $u(X', \mathbf{p}) \leq 0$ by construction. Hence, $(X, \mathbf{p}) \in \mathsf{IEF}(\Gamma)$ and $\mathsf{REV}(X, \mathbf{p}) = \nu$.

Secondly, assume that there exists an outcome $(X, \mathbf{p}) \in \mathsf{IEF}(\Gamma)$ such that $\mathsf{REV}(X, \mathbf{p}) = \nu$. By construction of the valuation function, this is possible only if $X \in F(\phi)$ and $\sum_{j \in X} p_j = \nu$, so that $u(X, \mathbf{p}) = 0$. However, since $v(\{\ell\}) = 1$ for every $\ell \in \mathsf{L}(\phi)$, $(X, \mathbf{p}) \in \mathsf{IEF}(\Gamma)$ implies that it must also be $p_j \geq 1$ for each $j \in X$. Hence, we can conclude that $p_j = 1$ for each $j \in X$. Assume, for the sake of contradiction, that ϕ is unsatisfiable. This implies that the assignment induced by all literals in $\mathsf{L}(\phi) \setminus X$ cannot satisfy ϕ, that is, there exists a clause $c_j \in C$ such that $c_j \subseteq X$. This implies $u(c_j, \mathbf{p}) = 3 + \epsilon - 3 = \epsilon > 0$ thus contradicting $(X, \mathbf{p}) \in \mathsf{IEF}(\Gamma)$. Hence, ϕ has to be satisfiable. ◀

On the positive side, [5] derive an $O(\log m)$-approximation algorithm for this problem. An interesting feature of this algorithm is that its performance guarantee holds also with respect to the maximum social welfare which is an upper bound to the maximum revenue, thus allowing the application of Lemma 6.

Hence, by combining Lemma 6 with Claim 7 and the result of [5], we obtain the following upper bounds.

▶ **Theorem 9.** *Let $P = (\Gamma, \mathsf{IEF}, \mathsf{obj})$ be a pricing problem with $\mathsf{obj} \in \{\mathsf{REV}, \mathsf{SW}\}$. The buyer preselection problem $\mathsf{BP}(P)$ admits an n-approximation when $\mathsf{obj} = \mathsf{SW}$, and an $O(n \log m)$-approximation when $\mathsf{obj} = \mathsf{REV}$.*

It is worth noticing that, given the proof of Lemma 6, the preselection claimed in Theorem 9 is someway "oblivious", i.e., it can be obtained by exploiting the minimum possible number of oracle queries, that is only a single oracle (demand) query for each buyer.

In light of the lower bound given in Theorem 5, the upper bounds given in Theorem 9 are asymptotically tight both for $\mathsf{obj} = \mathsf{SW}$ (unless $\mathbf{P} = \mathbf{ZPP}$) and for $\mathsf{obj} = \mathsf{REV}$ when $m = o(n)$ (unless $\mathbf{P} = \mathbf{ZPP}$).

4 Results for Bundle Envy-Free Outcomes

In this section, we consider the buyer preselection problem $\mathsf{BP}(\Gamma, \mathsf{BEF}, \mathsf{obj})$ with $\mathsf{obj} \in \{\mathsf{REV}, \mathsf{SW}\}$. Since we deal with bundle envy-free outcomes, we suppose that the price of any unsold item is infinite. Formally, given an outcome (N, o), where $o = (\mathbf{X}, \mathbf{p})$, for the buyer preselection problem $\mathsf{BP}(\Gamma, \mathsf{BEF}, \mathsf{obj})$ with $\Gamma = (N, M, (v_i)_{i \in N})$, we have that, for any $j \in M \setminus M(\mathbf{X})$, $p_j = \infty$.

We start by considering the buyer preselection problem $\mathsf{BP}(\Gamma, \mathsf{BEF}, \mathsf{REV})$.

The following theorem shows how it is possible to transform a bundle envy-free solution $(\bar{N}, \bar{o})$ with preselection into anther one (without preselection) having a non-smaller revenue.

▶ **Theorem 10.** *Given any solution* $(\bar{N}, \bar{o})$ *for the buyer preselection problem* $\mathsf{BP}(\Gamma, \mathsf{BEF}, \mathsf{REV})$*, with* $\Gamma = (N, M, (v_i)_{i \in N})$*, it is possible to compute in polynomial time an outcome* $o \in \mathsf{BEF}(\Gamma)$ *for problem* P *such that* $\mathsf{REV}(o) \geq \mathsf{REV}(\bar{o})$*.*

Theorem 10 can be proved by exploiting the notion and properties of maximal solutions, i.e, solutions in which no item price can be increased without changing the allocation (notice that an optimal solution is maximal), and by providing a constructive algorithm working on maximal solutions and allocating bundle of items to the players in $N \setminus \bar{N}$ that envy other winners. In particular, we first show that in a maximal solution $(\bar{N}, \bar{o})$ with preselection (i) for any buyer with positive utility, there exists another sold bundle providing her the same utility and (ii) there always exists a winner buyer having utility equal to 0. Given these properties, it is possible to exchange among the buyers the assigned bundles so that an excluded envious buyer can be assigned her preferred sold bundle $\bar{X}_j$ and can be therefore added to the solution without generating envy. Buyer j, getting bundle $\bar{X}_j$ in $(\bar{N}, \bar{o})$, gets another one providing her the same utility (by property (i) such a bundle always exists). This process can be iterated until a winner buyer with utility equal to 0 is reached and removed from the set of winners (it is possible to show that in this process a buyer is never considered twice and therefore by property (ii) it always terminates).

We now consider the buyer preselection problem $\mathsf{BP}(\Gamma, \mathsf{BEF}, \mathsf{SW})$. Analogously to Theorem 10 holding for the revenue maximization case, next theorem shows that a bundle envy-free outcome for the buyer preselection problem can be efficiently transformed in a bundle envy-free outcome, for the corresponding pricing problem (without preselection), having at least the same social welfare.

▶ **Theorem 11.** *Given any solution* $(\bar{N}, \bar{o})$ *for the buyer preselection problem* $\mathsf{BP}(\Gamma, \mathsf{BEF}, \mathsf{SW})$*, with* $\Gamma = (N, M, (v_i)_{i \in N})$*, it is possible to compute in polynomial time an outcome* $o \in \mathsf{BEF}(\Gamma)$ *for problem* P *such that* $\mathsf{SW}(o) \geq \mathsf{SW}(\bar{o})$*.*

Theorem 11 can be proved by considering a new market Γ' with unit-demand buyers, in which the set of buyers is N and there is an item for every bundle sold in $(\bar{N}, \bar{o})$. In fact, given that markets with unit-demand buyers always admit a Walrasian Equilibrium computable in polynomial time and that, by the well known *First Welfare Theorem*, Walrasian equilibria maximize social welfare over all possible outcomes, it can be easily obtained, by suitably setting the price of the items in Γ as a function of those in Γ', a bundle envy-free solution for Γ with no excluded buyer and having the same social welfare of $\bar{o}$.

As a consequence of Theorems 10 and 11, we obtain the following corollary.

▶ **Corollary 12.** *For* $\mathsf{obj} \in \{\mathsf{REV}, \mathsf{SW}\}$*, given an* α*-approximate solution* (N, o) *(with* $\alpha \geq 1$*, notice that when* $\alpha = 1$ *the corollary holds for optimal solutions) for the buyer preselection problem* $\mathsf{BP}(P)$ *with* $P = (\Gamma, \mathsf{BEF}, \mathsf{obj})$ *and* $\Gamma = (N, M, (v_i)_{i \in N})$*, it is possible to compute in polynomial time an outcome* $o' \in \mathsf{BEF}(\Gamma)$ *approximating the optimal solution of* P *by a factor equal to* α*.*

On the one hand, Corollary 12 tells us that, for $\mathsf{obj} \in \{\mathsf{REV}, \mathsf{SW}\}$, any inapproximability result holding for a pricing problem $P = (\Gamma, \mathsf{BEF}, \mathsf{obj})$ directly extends to the buyer preselection problem $\mathsf{BP}(P)$; on the other hand, it tells us that any α-approximation algorithm for $\mathsf{BP}(P)$ is also an α-approximation algorithm for P. Thus, as discussed in the Our Contribution subsection, preselection can be exploited as an algorithmic framework for designing approximation algorithms for the normal market scenario (without preselection).

5 The multi-unit case

In this section we study the multi-unit case, in which all the m items are of the same type. Recall that, for every $i \in N$, the valuation function becomes of the form $v_i : \{1, \ldots, m\} \to \mathbb{R}_{\geq 0}$ and that the market can be represented with $\Theta(nm)$ bits. Furthermore, in this case an allocation vector X can be specified by the number of items assigned to each buyer, i.e. $X = (x_1, \ldots, x_n)$, with $x_i \in \{0, \ldots, m\}$ for any $i = 1, \ldots, n$, and $\sum_{i=1}^{n} x_i \leq m$. Finally, we set the item pricing to p (i.e., all the multi-items have the same price) and the total price for selling x items is px.

A particular situation of multi-unit market, moreover, arises when one assumes *single-minded* buyers; in this case, since, for every player $i = 1, \ldots, n$, valuation function v_i can be completely defined by specifying how much every player valuates the set containing k_i items (being the only set she is interested in), $\Theta(n \log m)$ bits suffice to represent the market.

We first focus on the case of *item envy-free* solutions. Next claim shows that preselection can improve both the revenue and the social welfare of a market with multi-unit items, even in the case of single-minded buyers.

▶ **Claim 13.** *For any $\epsilon > 0$, there exists a market Γ with single-minded buyers and multi-unit items, and a pricing problem $P = (\Gamma, \mathsf{IEF}(\Gamma), \mathsf{obj})$ with $\mathsf{obj} \in \{\mathsf{REV}, \mathsf{SW}\}$, such that $\mathsf{obj}(o^*(\mathsf{BP}(P))) \geq (m - \epsilon)\mathsf{obj}(o^*(P))$.*

In fact, consider a market with only two single-minded buyers. Buyer 1 valuates $1 + \epsilon'$, for a small $\epsilon' > 0$, for receiving any (one) item. Buyer 2 valuates m for receiving all the m items. Notice that, in any item envy-free outcome for the setting without preselection, buyer 2 receives no item. In fact, if buyer 2 receives m items, the item pricing p must be at most 1. Thus, buyer 1 would be envious since she would get no item (i.e., there are no enough items). Therefore, for the market without preselection, the only feasible solutions are selling one item to buyer 1 at price at least 1 and at most $1 + \epsilon'$. Thus the optimal revenue and the optimal social welfare are at most $1 + \epsilon'$. However, if we consider the submarket with only buyer 2, i.e., we exclude buyer 1, we can sell m items at item pricing 1, thus obtaining a revenue and social welfare of m.

We notice that the above bound is tight. It is easy to see that preselection cannot improve the revenue and social welfare of envy-free solutions by a factor greater than m. In fact, it is sufficient to sell items to the buyer i such that $(i, j) = \text{argmax}_{i=1,\ldots,n;j=1,\ldots,m} \frac{v_i(j)}{j}$.

Now we focus on the computation of optimal or approximate solution for the preselection problem in the case of item envy-free outcomes, both for the social welfare and the revenue objective functions.

▶ **Theorem 14.** *Given a pricing problem $P = (\Gamma, \mathsf{IEF}(\Gamma), \mathsf{obj})$, where $\Gamma = (N, M, (v_i)_{i \in N})$ is a market with multi-unit items and $\mathsf{obj} \in \{\mathsf{REV}, \mathsf{SW}\}$, the buyer preselection problem $\mathsf{BP}(P)$ can be optimally solved in polynomial time.*

Proof. Recall that solving the buyer preselection problem $\mathsf{BP}(P)$ corresponds to find a subset N^* of players to admit to the market and an optimal outcome $o^* = (X^*, p^*)$, in which in the considered case of multi-unit items p^* is just the price of a single item.

In order to prove the claim, we first show that it is possible to compute in polynomial time a set $\mathcal{P}$ containing the optimal price p^* for problem $\mathsf{BP}(P)$.

Consider the set $\mathcal{P}$ defined as follows:

$$\begin{aligned} \mathcal{P} \quad = \quad & \{y | v_i(k) - yk = v_i(k') - yk', i \in N, \\ & k, k' \in \{0, 1, \ldots, m\}, k \neq k'\}. \end{aligned}$$

Roughly speaking, for all buyers $i \in N$ and for all couples (k, k') of integers belonging to $\{0, 1, \ldots, m\}$ such that $k \neq k'$, $\mathcal{P}$ contains the solution y of equality $v_i(k) - yk = v_i(k') - yk'$. Clearly, $\mathcal{P}$ can be computed in $O(nm^2)$ time, i.e., in polynomial time in the size of the instance. Now, assume by contradiction that the optimal solution (N^*, o^*) for $\mathsf{BP}(P)$ with the highest possible item price assign to an item price $p^* \notin \mathcal{P}$, and let $p' \in \mathcal{P}$ the smallest element of $\mathcal{P}$ such that $p' > p^*$. Notice that this element $p' \in \mathcal{P}$ has to exist, because p^* must verify, for any $i = 1, \ldots, n$, $v_i(\bar{x}_i) - p^*\bar{x}_i \geq 0$ and y verifying equality $v_i(\bar{x}_i) - y\bar{x}_i = 0$ belongs to $\mathcal{P}$. Since p^* induces an envy-free solution, it has to verify, for every $i = 1, \ldots, n$, constraint $v_i(\bar{x}_i) - p^*\bar{x}_i \geq v_i(j) - p^*j$ for any $j = 0, \ldots, m$; since, $p^* \notin \mathcal{P}$, given how $\mathcal{P}$ is defined, it follows that all above constraints are not verified in a strict manner, i.e. it is verified that, for every $i = 1, \ldots, n$, $v_i(\bar{x}_i) - p^*\bar{x}_i > v_i(j) - p^*j$ for any $j = 0, \ldots, m$. Therefore, p' still continues to verify all envy-free constraints (with some constraints possibly become strict). It follows that we have found a new envy-free outcome o' such that $\mathsf{SW}(o') = \mathsf{SW}(o^*)$ and $\mathsf{REV}(o') > \mathsf{REV}(o^*)$: a contradiction to the fact that o^* was the optimal outcome with the highest possible item price.

Therefore, since the number of values that can be assigned to p^* in order to obtain an optimal outcome is polynomial in the size of the instance, it remains to show that, given a fixed price p^*, it is possible to optimally compute in polynomial time a subset N^* of players to admit to the market and an allocation X^*. In fact, the optimal solution of $\mathsf{BP}(P)$ is given by the best solution among the ones obtained for all the candidate prices belonging to $\mathcal{P}$.

The *0-1 Multiple-Choice Knapsack Problem (0-1 MCKP)* is a generalization of the classical Knapsack problem introduced in [30]. In this problem, we are given α classes $C_1, C_2, \ldots, C_\alpha$ of elements to pack in some knapsack of capacity c. For every $i = 1, \ldots, n$, each element $e \in C_i$ (let $class(e) = i$ be the index of the class to which e belongs) has a profit β_e and a volume γ_e, and the problem is to choose a set E containing *at most* one element from each class such that the profit sum is maximized without the volume sum exceeding capacity c. In [21] it is shown that it can be optimally solved in pseudo-polynomial time, i.e., in time $O(c)$.

We now provide a polynomial reduction from our problem to 0-1 MCKP. Given an instance I of $\mathsf{BP}(P)$ and fixed a price p^*, we construct an instance I' of 0-1 MCKP as follows: We have $\alpha = n$ classes (one class for each buyer) and the capacity $c = m$. Consider, for every buyer $i = 1, \ldots, n$, the number of items providing her with the highest possible utility: let $\bar{U}_i = \arg\max_{k=1,\ldots,m} v_i(k) - kp^*$ be the set containing these values. It is easy to check that, in every item envy-free solution, allocation X must satisfy $x_i \in \bar{U}_i$ for every player $i = 1, \ldots, n$. For every $i = 1, \ldots, n$, consider set $\bar{U}_i$: we add to class C_i an element e for every $k \in \bar{U}_i$ such that $\gamma_e = k$ and

- $\beta_e = k$ if $\mathsf{obj} = \mathsf{REV}$;
- $\beta_e = v_i(k)$ if $\mathsf{obj} = \mathsf{SW}$.

Given a solution for I' with total profit $\beta = \sum_{e \in E} \beta_e$, it is possible to obtain a solution for I with fixed price p^*, i.e., a subset $\bar{N}$ of players to admit to the market and an allocation $\bar{X}$, as follows: $\bar{N}$ contains the players associated to classes containing an element belonging to E, i.e., $\bar{N} = \{i | C_i \cap E \neq \emptyset\}$, and the allocation vector $\bar{X} = (\bar{x}_1, \ldots, \bar{x}_n)$ is such that, for every $i = 1, \ldots, n$, $x_i = \gamma_e$ if there exists an element $e \in C_i \cap E$. Clearly, $\mathsf{obj}((\bar{X}, p^*)) = \beta$. Furthermore, by the way $\bar{U}_i$ is defined, outcome $(\bar{X}, p^*)$ is item envy-free.

Conversely, given a subset $\bar{N}$ of preselected players and an outcome $o = (X, p^*)$ for I, it is possible to obtain a solution for I' as follows: for every $i \in \bar{N}$, add to E element $e \in C_i$ such that $\gamma_e = x_i$ (by recalling the definition of $\bar{U}_i$, it holds that this element e belongs to C_i because outcome o is item envy-free). Clearly, the total profit $\beta = \sum_{e \in E} \beta_e = \mathsf{obj}((X, p^*))$.

The claim follows by noticing that, since $c = m$, the pseudo-polynomial algorithm of [21] is in fact polynomial with respect to the size of the instance of problem $\mathsf{BP}(P)$. ◀

For the special case of single-minded buyers in which an instance of the buyer preselection problem can be represented by $\Theta(n \log m)$ bits, by exploiting the same ideas used for proving Theorem 14, the following theorem provides an **FPTAS**.

▶ **Theorem 15.** *Given a pricing problem* $P = (\Gamma, \mathsf{IEF}(\Gamma), \mathsf{obj})$*, where* $\Gamma = (N, M, (v_i)_{i \in N})$ *is a market with single-minded buyers and multi-unit items, for* $\mathsf{obj} \in \{\mathsf{REV}, \mathsf{SW}\}$*, the buyer preselection problem* $\mathsf{BP}(P)$ *admits a* fully polynomial approximation scheme*.*

We complement the result of Theorem 15 by showing a tight lower bound to the problem of computing the maximum revenue to the case of item envy-free with single-minded buyers and multi-unit items. The following claim can be proved by exploiting a reduction from the Subset Sum problem.

▶ **Claim 16.** *Given a pricing problem* $P = (\Gamma, \mathsf{IEF}(\Gamma), \mathsf{REV})$*, where* $\Gamma = (N, M, (v_i)_{i \in N})$ *is a market with single-minded buyers and multi-unit items, the buyer preselection problem* $\mathsf{BP}(P)$ *is* **NP***-Hard.*

Proof. We use a reduction from the Subset Sum problem, that is defined as follows: given a set of integers and an integer s, does any non-empty subset sum to s? Let n be the number of elements $\{a_1, a_2, \ldots, a_n\}$ in the given instance of the Subset sum problem, and let s be the required sum. We assume that $\sum_{i=1}^{n} a_i > s$, as otherwise the problem is trivial. For all a_i in the input of the Subset sum problem, we create a corresponding buyer i with the following valuation. Buyer i has valuation a_i for receiving a_i items, and zero otherwise. Moreover, we set the number of items $m = s$. In such case, if there is a solution for the Subset sum problem, then by setting the item pricing $p = 1$ and selling to corresponding buyers gives us a feasible and envy-free outcome in which the revenue equals to m. It is easy to see that m is an upper bound to the maximum revenue. On the other hand, if the revenue of the optimal outcome to the buyer preselection problem is equal to m, we can obtain the solution to the subset sum problem. Notice that we can obtain a revenue of m only if we sell exactly m items at item pricing $p = 1$. In fact, at item pricing $p > 1$, we sell no item, and at item pricing $p < 1$, we do not get an optimal solution. ◀

We now focus on the case of *bundle envy-free* solutions. We first notice that the results of Section 4 claiming that bundle envy-free solutions do not improve the quality of outcomes (with respect neither to social welfare nor to revenue maximization) do not hold for the multi-unit case, because in this case it is not possible to change the price of some item without influencing the other ones. We start by showing that preselection can improve both the revenue and the social welfare of a market with multi-unit items, even in the case of single-minded buyers.

▶ **Claim 17.** *For any* $\epsilon > 0$*, there exists a market* Γ *with single-minded buyers and multi-unit items, and a pricing problem* $P = (\Gamma, \mathsf{BEF}(\Gamma), \mathsf{obj})$ *with* $\mathsf{obj} \in \{\mathsf{REV}, \mathsf{SW}\}$*, such that* $\mathsf{obj}(o^*(\mathsf{BP}(P))) \geq (2 - \epsilon)\mathsf{obj}(o^*(P))$*.*

In fact, consider a market with $x + 1$ single-minded buyers. Buyer 1 valuates x for receiving x items. Buyer i, for any $i = 2, 3, \ldots, x + 1$, valuates $1 + \epsilon'$, for a small $\epsilon' > 0$, for receiving any (one) item. Finally, the number of items is $2x - 1$. Notice that, in any bundle envy-free outcome for the setting without preselection, if buyers 1 receives x items, it implies that the item pricing p must be at most 1. It further implies that, in such outcome, no buyer i, for any $i = 2, 3, \ldots, x + 1$, can get items. The reason is that such buyers have positive utility for receiving one item, but the number of items is not sufficient to satisfy all of them. Therefore, the only chance is selling no bundle of one item. Thus, on one hand, without preselection,

the best revenue and social welfare is $x(1+\epsilon')$. It can be obtained by selling one item to buyers i, for any $i = 2, 3, \ldots, x+1$, at price $1+\epsilon'$. On the other hand, if we consider the submarket with only buyers $1, 2, \ldots, x$ (i.e., we exclude one buyer that valuates $1+\epsilon'$ for receiving any (one) item), we can sell $2x-1$ items at item pricing 1, that is x items to buyer 1, and one item to buyers i, for any $i = 2, \ldots, x$, thus obtaining a revenue and social welfare of $2x-1$.

We now show that, for the market with multi-unit items and general valuations, preselection can improve the revenue by a multiplicative factor of at most 2, thus closing in a tight way the previous bound.

▶ **Theorem 18.** *Given a solution $(\bar{N}, \bar{o})$ for the buyer preselection problem $\mathsf{BP}(\Gamma, \mathsf{BEF}, \mathsf{REV})$, with $\Gamma = (N, M, (v_i)_{i\in N})$ being a market with multi-unit items, it is possible to compute in polynomial time an outcome $o \in \mathsf{BEF}(\Gamma)$ for problem P such that $\mathsf{REV}(\bar{o}) \leq 2\mathsf{REV}(o)$.*

Proof. Let $\bar{o} = (\bar{X}, \bar{p})$. In the following we will show how to compute in polynomial time an outcome $o \in \mathsf{BEF}(\Gamma)$ with $o = (X, p)$ such that (i) $p \geq \bar{p}$ and (ii) $|M(X)| \geq \frac{|M(\bar{X})|}{2}$, i.e., outcome o sells at least one half of the items sold by outcome $\bar{o}$ at a price at least equal to $\bar{p}$. Clearly, this directly implies that $\mathsf{REV}(o) \geq \frac{\mathsf{REV}(\bar{o})}{2}$.

For every $j \in \{1, \ldots, m\}$, let a_j be the number of items that could be sold to some buyers in bundles of cardinality j at price $\bar{p}$ (we require that these buyers obtain a non-negative utility for a bundle of j items). More formally, for every $j = 1, \ldots, m$, let $B_j(\bar{X}) = \{i | v_i(j) - j\bar{p} \geq 0\}$ be the subset of players obtaining a non-negative utility for a bundle with j items; then, $a_j = j|B_j(\bar{X})|$.

We divide the proof in two disjoint cases.

- If there exists $j \in \{1, \ldots, m\}$ such that $a_j \geq \frac{|M(\bar{X})|}{2}$, outcome $o = (X, p)$ is such that $x_k = j$ for every $k \in B_j(\bar{X})$, and $x_k = 0$ otherwise. By setting $p = \bar{p}$, by the definition of a_j, we know that a_j items could be sold without generating envy.
  If $a_j \leq m$, we are done.
  If $j \geq \frac{|M(\bar{X})|}{2}$, we can increase the price p so that only buyer i, with i such that $v_i(j) = \max_{k=1}^{n} v_k(j)$, is assigned the bundle (notice that in this way no other buyer is envious).
  It remains to deal with the subcase in which $a_j > m$ and $j < \frac{|M(\bar{X})|}{2}$: We increment price p until the number of buyers x with positive utility is such that $xj \leq m$. We then assign bundles of j items to all buyers with positive utility and to as many buyers with zero utility as possible. Notice that (i) since $j < \frac{|M(\bar{X})|}{2}$ implies that $m - j > \frac{|M(\bar{X})|}{2}$, this process leads to obtain at least $\frac{|M(\bar{X})|}{2}$ assigned items and (ii) again no buyer is envious.
- If for all $j \in \{1, \ldots, m\}$ it holds that $a_j < \frac{|M(\bar{X})|}{2}$, outcome $o = (X, \bar{p})$ (with the same price of outcome $\bar{o}$) is computed as follows.
  For any $i = 1, \ldots, n$ and $k = 1, \ldots, m$, let $u_i^{*k} = \max_{t=1}^{k} v_i(t) - t\bar{p}$ be the maximum possible utility of buyer i for bundles of at most k items and $b_i^{*k} = \min\{j | v_i(j) - j\bar{p} = u_i^{*k}\}$ the minimum size of a bundle of maximum utility for buyer i. Moreover, for any $k = 1, \ldots, m$ and $j = 1, \ldots, k$, let $B_{k,j} = \{i | b_i^{*k} = j\}$ be the set of buyers having j items as their best bundle of maximum utility, among all bundles made up to k items, and resolving ties by selecting the bundle of minimum size.
  Clearly, $B_{k,j}$ can be computed in time $O(poly(n, m))$. For $k = 1, \ldots, m$, consider allocation $X^k = (x_1^k, \ldots, x_n^k)$ such that, for every $i = 1, \ldots, n$, $x_i^j = j$ if $i \in B_{k,j}$ and $x_i^j = 0$ otherwise. By the definition of $B_{k,j}$ it follows that allocation X^k is envy-free. Moreover, it can be easily verified that $|M(X^1)| = a_1$ and, for any $j = 2, \ldots, m$, $|M(X^j)| - |M(X^{j-1})| \leq a_j$.

If $|M(X^m)| \leq m$, the claim trivially follows by setting $X = X^m$ because we are allocating at least all items allocated in $\bar{o}$.
Otherwise, let k' be the minimum value of $k = 2, \ldots, m$ such that $|M(X^{k'})| > m$: the claim follows by setting $X = X^{k'-1}$. In fact, since $|M(X^{k'})| - |M(X^{k'-1})| \leq a_{k'} \leq \frac{|M(\bar{X})|}{2}$, it follows that $|M(X^{k'-1})| \geq m - \frac{|M(\bar{X})|}{2} \geq \frac{|M(\bar{X})|}{2}$. ◀

It is worth noticing that, on the one hand, preselection can be exploited as an algorithmic framework for designing good approximation algorithms (loosing only a moltiplicative factor of 2) for the normal market scenario without preselection; on the other hand, since the optimal revenue with preselection is at most twice the one without, an α-approximation algorithm for the normal market without preselection, is a 2α-approximation one for market with preselection.

6 Final remarks and Future work

Many results holding for the item envy-free outcomes and social welfare objective function extend to the notion of Walrasian equilibria, that are item envy-free outcomes with the additional requirement that the market clears, i.e., every unsold item is assigned price zero. In particular, the inapproximability result of Theorem 5 and the n-approximation algorithm for the buyer preselection problem of Theorem 9 directly extend to Walrasian equilibria. Notice also that for the remaining uncovered cases, that is when the goal is that of optimizing the seller's revenue, there is no reason for requiring market clearance, a condition clearly limiting the power of setting prices so as to maximize the revenue.

The main left open problems are: for markets with a unique buyer, closing the gap between the **NP**-hardness and the logarithmic approximation for the case of revenue maximization and item envy-free solutions; for the multi-unit case with bundle envy-free outcomes, determining an upper bound to the social welfare improvement achievable by preselection and setting the complexity of computing optimal solutions, for both the revenue and the social welfare cases.

References

1 Noga Alon, Yishay Mansour, and Moshe Tennenholtz. Differential pricing with inequity aversion in social networks. In *Proc. of EC*, pages 9–24, 2013.

2 Georgios Amanatidis, Evangelos Markakis, and Krzysztof Sornat. Inequity aversion pricing over social networks: Approximation algorithms and hardness results. In *Proc. of MFCS*, pages 9:1–9:13, 2016.

3 E. Anshelevich, K. Kar, and S. Sekar. Envy-free pricing in large markets: Approximating revenue and welfare. In *Proc. of ICALP*, pages 52–64. Springer, 2015.

4 A. Archer, C. H. Papadimitriou, K. Talwar, and É. Tardos. An approximate truthful mechanism for combinatorial auctions with single parameter agents. *Internet Mathematics*, 1(2):129–150, 2003.

5 M. F. Balcan, A. Blum, and Y. Mansour. Item pricing for revenue maximization. In *Proc. of EC*, pages 50–59, 2008.

6 S. Bikhchandani and J. W. Mamer. Competitive equilibrium in an exchange economy with indivisibilities. *Journal of Economic Theory*, 74(2):386–413, 1997.

7 V. Bilò, M. Flammini, and G. Monaco. Approximating the revenue maximization problem with sharp demands. *Theoretical Computer Science*, 662:9–30, 2017.

8 V. Bilò, M. Flammini, G. Monaco, and L. Moscardelli. On the impact of buyers preselection in pricing problems. In *Proc. of AAMAS*, 2018.

9 P. Briest. Uniform budgets and the envy-free pricing problem. In *Proc. of ICALP*, pages 808–819. Springer, 2008.

10 P. Briest and P. Krysta. Single-minded unlimited supply pricing on sparse instances. In *Proc. of SODA*, pages 1093–1102. ACM Press, 2006.

11 P. Chalermsook, J. Chuzhoy, S. Kannan, and S. Khanna. Improved hardness results for profit maximization pricing problems with unlimited supply. In *Proc. of APPROX*, pages 73–84. Springer, 2012.

12 P. Chalermsook, B. Laekhanukit, and D. Nanongkai. Graph products revisited: Tight approximation hardness of induced matching, poset dimension and more. In *Proc. of SODA*, pages 1557–1576. ACM Press, 2013.

13 P. Chalermsook, B. Laekhanukit, and D. Nanongkai. Independent set, induced matching, and pricing: Connections and tight (subexponential time) approximation hardnesses. In *Proc. of FOCS*, pages 370–379. IEEE Computer Society, 2013.

14 N. Chen and X. Deng. Envy-free pricing in multi-item markets. In *Proc. of ICALP*, pages 418–429. Springer, 2010.

15 N. Chen, X. Deng, P. W. Goldberg, and J. Zhang. On revenue maximization with sharp multi-unit demands. *Journal of Combinatorial Optimization*, 31(3):1174–1205, 2016.

16 N. Chen, A. Ghosh, and S. Vassilvitskii. Optimal envy-free pricing with metric substitutability. *SIAM Journal on Computing*, 40(3):623–645, 2011.

17 Ning Chen and Atri Rudra. Walrasian equilibrium: Hardness, approximations and tractable instances. *Algorithmica*, 52(1):44–64, 2008.

18 M. Cheung and C. Swamy. Approximation algorithms for single-minded envy-free profit-maximization problems with limited supply. In *Proc. of FOCS*, pages 35–44. IEEE Computer Society, 2008.

19 E. H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

20 E. D. Demaine, U. Feige, M. Hajiaghayi, and M. R. Salavatipour. Combination can be hard: Approximability of the unique coverage problem. *SIAM Journal on Computing*, 38(4):1464–1483, 2008.

21 K. Dudzinski and S. Walukiewicz. Exact methods for the knapsack problem and its generalizations. *European Journal of Operational Research*, 28(1):3–21, 1987.

22 M. Feldman, A. Fiat, S. Leonardi, and P. Sankowski. Revenue maximizing envy-free multi-unit auctions with budgets. In *Proc. of EC*, pages 532–549. ACM Press, 2012.

23 M. Feldman, N. Gravin, and B. Lucier. Combinatorial walrasian equilibrium. *SIAM Journal on Computing*, 45(1):29–48, 2016.

24 M. Flammini, M. Mauro, and M. Tonelli. On social envy-freeness in multi-unit markets. In *Proc. of AAAI*, 2018.

25 D. Foley. Resource allocation and the public sector. *Yale Economic Essays*, 7:45–98, 1967.

26 T. Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

27 F. Gul and E. Stacchetti. Walrasian equilibrium with gross substitutes. *Journal of Economic Theory*, 87:95–124, 1999.

28 V. Guruswami, J. D. Hartline, A. R. Karlin, D. Kempe, C. Kenyon, and F. McSherry. On profit-maximizing envy-free pricing. In *Proc. of SODA*, pages 1164–1173. ACM Press, 2005.

29 J. Hartline and Q. Yan. Envy, truth, and profit. In *Proc. of EC*, pages 243–252. ACM Press, 2011.

30 E. L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 44(4):339–356, 1979.

31 G. Monaco, P. Sankowski, and Q. Zhang. Revenue maximization envy-free pricing for homogeneous resources. In *Proc. of IJCAI*, pages 90–96, 2015.

32 H. R. Varian. Equity, envy, and efficiency. *Journal of Economic Theory*, 9:63–91, 1974.

33 W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

34 L. Walras. *Elements of Pure Economics*. Allen and Unwin, 1954.

On Randomized Generation of Slowly Synchronizing Automata

Costanza Catalano
Gran Sasso Science Institute
Viale Francesco Crispi 7, L'Aquila, Italy
costanza.catalano@gssi.it

Raphaël M. Jungers[1]
ICTEAM Institute, UCLouvain
Avenue Georges Lemaîtres 4-6, Louvain-la-Neuve, Belgium
raphael.jungers@uclouvain.be

Abstract

Motivated by the randomized generation of slowly synchronizing automata, we study automata made of permutation letters and a merging letter of rank $n-1$. We present a constructive randomized procedure to generate synchronizing automata of that kind with (potentially) large alphabet size based on recent results on *primitive* sets of matrices. We report numerical results showing that our algorithm finds automata with much larger reset threshold than a mere uniform random generation and we present new families of automata with reset threshold of $\Omega(n^2/4)$. We finally report theoretical results on randomized generation of primitive sets of matrices: a set of permutation matrices with a 0 entry changed into a 1 is primitive and has exponent of $O(n \log n)$ with high probability in case of uniform random distribution and the same holds for a random set of binary matrices where each entry is set, independently, equal to 1 with probability p and equal to 0 with probability $1-p$, when $np - \log n \to \infty$ as $n \to \infty$.

2012 ACM Subject Classification Mathematics of computing → Combinatorics, Mathematics of computing → Random graphs, Theory of computation → Randomness, geometry and discrete structures

Keywords and phrases Synchronizing automata, random automata, Černý conjecture, automata with simple idempotents, primitive sets of matrices

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.48

Acknowledgements The authors thank François Gonze and Vladimir Gusev for significant suggestions and fruitful discussions on the topic.

1 Introduction

A (complete deterministic finite) *automaton* $\mathcal{A}$ on n states can be defined as a set of m binary row-stochastic[2] matrices $\{A_1, \ldots, A_m\}$ that are called the *letters* of the automaton. We say that $\mathcal{A}$ is *synchronizing* if there exists a product of its letters, with repetitions allowed, that has an all-ones column[3] and the length of the shortest of these products is called the *reset*

[1] R. M. Jungers is a FNRS Research Associate. He is supported by the French Community of Belgium, the Walloon Region and the Innoviris Foundation.

[2] A *binary* matrix is a matrix with entries in $\{0, 1\}$. A *row-stochastic* matrix is a matrix with nonnegative entries where the entries of each row sum up to 1. Therefore a matrix is *binary* and *row-stochastic* if each row has exactly one 1.

[3] A column whose entries are all equal to 1.

Table 1 Table on upper bounds on the reset threshold for some classes of automata and examples of automata with large reset threshold belonging to these classes, up to date.

Classes	**Upper b. on rt**	**Families with quadratic rt**
Eulerian automata	n^2-3n+3 Kari [16]	$(n^2-3)/2$ Szykuła and Vorel [27] (4 letters)
Automata with full transition monoid	$2n^2-6n+5$ Gonze et. al. [14]	$n(n-1)/2$ Gonze et. al. [14] (n letters)
One cluster automata	n^2-7n+7 Béal et al. [4]	$(n-1)^2$ Černý [28] (2 letters)
Strongly connected weakly monotone automata	$\lfloor n(n+1)/6 \rfloor$ Volkov [29]	?
Automata with simple idempotents	$2(n-1)^2$ Rystov [25]	$(n-1)^2$ Černý [28] (2 letters) $\geq (n^2+3n-6)/4$ for $n=4k+3$ [Conjectured $(n^2-1)/2$] $\geq (n^2+3n-8)/4$ for $n=4k+1$ [Conjectured $(n^2-1)/2$] $\geq (n^2+2n-4)/4$ for $n=4k$ [Conjectured $(n^2-2)/2$] $\geq (n^2+2n-12)/4$ for $n=4k+2$ [Conjectured $(n^2-10)/2$] **Our contribution** (3 letters)

threshold ($rt(\mathcal{A})$) of the automaton. In other words, an automaton is synchronizing if there exists a word that brings the automaton into a particular state, regardless of the initial one. Synchronizing automata appear in different research fields; for example they are often used as models of error-resistant systems [10, 7] and in symbolic dynamics [18]. For a brief account on synchronizing automata and their other applications we refer the reader to [30]. The importance of synchronizing automata also arises from one of the most longstanding open problems in this field, the Černý conjecture, which affirms that any synchronizing automaton on n states has reset threshold at most $(n-1)^2$. If it is true, the bound is sharp due to the existence of a family of 2-letter automata attaining this value, family discovered by Černý in [28]. Despite great effort, the best upper bound for the reset threshold known so far is $(15617n^3+7500n^2+9375n-31250)/93750$, recently obtained by Szykuła in [26] and thereby beating the 30 years-standing upper bound of $(n^3-n)/6$ found by Pin and Frankl in [11, 22]. Better upper bounds have been obtained for certain families of automata and the search for automata attaining quadratic reset threshold within these families have been the subject of several contributions in recent years. These results are (partly) summarized in Table 1.
Exhaustive search confirmed the conjecture for small values of n (see [3, 9]). The hunt for a possible counterexample to the conjecture turned out not to be an easy task as well; the search space is wide and calculating the reset threshold is computationally hard (see [10, 21]). Automata with reset thresholds close to $(n-1)^2$, called *extremal* or *slowly synchronizing* automata, are also hard to detect and not so many families are known; Bondt et. al. [9] make a thorough analysis of automata with small number of states and we recall, among others, the families found by Ananichev et al. [3], by Gusev and Pribavkina [15], by Kisielewicz and Szykuła [17] and by Dzyga et. al. [19]. These last two examples are, in particular, some

of the few examples of slowly synchronizing automata with more than two letters that can be found in the literature. Almost all the families of slowly synchronizing automata listed above are closely related to the Černý automaton $\mathcal{C}(n) = \{a, b\}$, where a is the cycle over n vertices and b the letter that fixes all the vertices but one, which is mapped to the same vertex as done by a; indeed all these families present a letter that is a cycle over n vertices and the other letters have an action similar to the one of letter b. As these examples seem to have a quite regular structure, it is natural to wonder whether a randomized procedure to generate automata could obtain less structured automata with possibly larger reset thresholds. This probabilistic approach can be rooted back to the work of Erdős in the 60's, where he developed the so-called *Probabilistic Method*, a tool that permits to prove the existence of a structure with certain desired properties by defining a suitable probabilistic space in which to embed the problem; for an account on the probabilistic method we refer the reader to [1]. The simplest way to randomly generate an automaton of m letters is to uniformly and independently sample m binary row-stochastic matrices: unfortunately, Berlinkov first proved in [5] that two uniformly sampled random binary row-stochastic matrices synchronize with high probability (i.e. the probability that they form a synchronizing automaton tends to 1 as the matrix dimension tends to infinity), then Nicaud showed in [20] that they also have reset threshold of order $O(n \log^3 n)$ with high probability. We say that an automaton is *minimally synchronizing* if any proper subset of its letters is not synchronizing; what just presented before implies that a uniformly sampled random automaton of m letters has low reset threshold and is *not* minimally synchronizing with high probability. Summarizing:

- slowly synchronizing automata cannot be generated by a mere uniform randomized procedure;
- minimally synchronizing automata with more than 2 letters are especially of interest as they are hard to find and they do not appear often in the literature, so the behaviour of their reset threshold is still unclear.

With this motivation in place, our paper tackles the following questions:

Q1 Is there a way to randomly generate (minimally) slowly synchronizing automata (with more than two letters)?

Q2 Can we find some automata families with more than two letters, quadratic reset threshold and that do not resemble the Černý family?

Our Contribution. In this paper we give positive answers to both questions **Q1** and **Q2**. For the first one, we rely on the concept of *primitive* set of matrices, introduced by Protasov and Voynov in [24]: a finite set of matrices with nonnegative entries is said to be *primitive* if there exists a product of these matrices, with repetitions allowed, with all positive entries. A product of this kind is called *positive* and the length of the shortest positive product of a primitive set $\mathcal{M}$ is called the *exponent* ($exp(\mathcal{M})$) of the set. Although the Protasov-Voynov primitivity has gained a lot of attention in different fields as in stochastic switching systems [23] and consensus for discrete-time multi-agent systems [8], we are interested in its connection with automata theory. In the following, we say that a matrix is *NZ* if it has neither zero-rows nor zero-columns[4]; a matrix set is said to be *NZ* if all its matrices are NZ.

▶ **Definition 1.** Let $\mathcal{M} = \{M_1, \ldots, M_m\}$ be a binary NZ-matrix set. The *automaton associated to* the set $\mathcal{M}$ is the automaton $\mathcal{A}(\mathcal{M})$ whose letters are all the binary row-stochastic matrices that are entrywise not greater than at least one matrix in $\mathcal{M}$.

[4] Thus a NZ-matrix must have a positive entry in every row and in every column.

Figure 1 The automata $\mathcal{A}(\mathcal{M})$ and $\mathcal{A}(\mathcal{M}^T)$ of Example 2.

▶ **Example 2.** We here provide an example of a primitive set $\mathcal{M} = \{M_1, M_2\}$ and the associated automata $\mathcal{A}(\mathcal{M})$ and $\mathcal{A}(\mathcal{M}^T)$ in both their matrix and graph representations (Figure 1), where $\mathcal{M}^T = \{M_1^T, M_2^T\}$.

$$\mathcal{M} = \left\{ \begin{pmatrix} 0&1&0\\1&0&0\\0&0&1 \end{pmatrix}, \begin{pmatrix} 1&0&1\\0&0&1\\0&1&0 \end{pmatrix} \right\}, \mathcal{A}(\mathcal{M}) = \left\{ \begin{pmatrix} 0&1&0\\1&0&0\\0&0&1 \end{pmatrix}, \begin{pmatrix} 1&0&0\\0&0&1\\0&1&0 \end{pmatrix}, \begin{pmatrix} 0&0&1\\0&0&1\\0&1&0 \end{pmatrix} \right\} = \{a, b, c\},$$

$$\mathcal{A}(\mathcal{M}^T) = \left\{ \begin{pmatrix} 0&1&0\\1&0&0\\0&0&1 \end{pmatrix}, \begin{pmatrix} 1&0&0\\0&0&1\\0&1&0 \end{pmatrix}, \begin{pmatrix} 1&0&0\\0&0&1\\1&0&0 \end{pmatrix} \right\} = \{a, b, c'\}.$$

The following theorem summarizes two results proved by Blondel et. al. ([6], Theorems 16-17) and a result proved by Gerencsér et al. ([12], Theorem 8). Note that we state it for sets of *binary* NZ-matrices but it more generally holds for any set of NZ-matrices with nonnegative entries; this relies on the fact that in the notion of primitivity what counts is the position of the nonnegative entries within the matrices of the set and not their the actual values. In this case we should add to Definition 1 the request of setting to 1 all the positive entries of the matrices of $\mathcal{M}$ before building $\mathcal{A}(\mathcal{M})$.

▶ **Theorem 3.** *Let $\mathcal{M} = \{M_1, \ldots, M_m\}$ a set of binary NZ-matrices of size $n \times n$ and $\mathcal{M}^T = \{M_1^T, \ldots, M_m^T\}$. It holds that $\mathcal{M}$ is primitive if and only if $\mathcal{A}(\mathcal{M})$ (equiv. $\mathcal{A}(\mathcal{M}^T)$) is synchronizing. If $\mathcal{M}$ is primitive, then it also holds that:*

$$\max\Big\{rt\big(\mathcal{A}(\mathcal{M})\big), rt\big(\mathcal{A}(\mathcal{M}^T)\big)\Big\} \leq exp(\mathcal{M}) \leq rt\big(\mathcal{A}(\mathcal{M})\big) + rt\big(\mathcal{A}(\mathcal{M}^T)\big) + n - 1. \tag{1}$$

▶ **Example 4.** For the matrix set $\mathcal{M}$ defined in Example 2, it holds that $exp(\mathcal{M}) = 8$, $rt\big(\mathcal{A}(\mathcal{M})\big) = 4$ and $rt\big(\mathcal{A}(\mathcal{M}^T)\big) = 2$.

Theorem 3 will be extensively used throughout the paper. It shows that primitive sets can be used for generating synchronizing automata and Equation (1) tells us that the presence of a primitive set with large exponent implies the existence of an automaton with large reset threshold; in particular the discovery of a primitive set $\mathcal{M}$ with $exp(\mathcal{M}) \geq 2(n-1)^2 - n + 1$ would disprove the Černý conjecture. On the other hand, the upper bounds on the automata reset threshold mentioned before imply that $exp(\mathcal{M}) = O(n^3)$.
One advantage of using primitive sets is the Protasov-Voynov characterization theorem (see Theorem 6 in Section 2) that describes a combinatorial property that a NZ-matrix set must have in order *not* to be primitive: by constructing a primitive set such that each of its proper subsets has this property, we can make it *minimally primitive*[5].
We decided to focus our attention on what we call *perturbed permutation* sets, i.e. sets made of permutation matrices (binary matrices having exactly one 1 in every row and in every column) where a 0-entry of one of these matrices is changed into a 1. These sets have many interesting properties:

[5] Thus a *minimally primitive* set is a primitive set that does not contain any proper primitive subset.

- they have the least number of positive entries that a NZ-primitive set can have, which intuitively should lead to sets with large exponent;
- the associated automaton $\mathcal{A}(\mathcal{M})$ of a perturbed permutation set $\mathcal{M}$ can be easily computed. It is made of permutation letters and a letter of rank $n-1$ and its alphabet size is just one unit more than the cardinality of $\mathcal{M}$;
- if the matrix set $\mathcal{M}$ is minimally primitive, the automaton $\mathcal{A}(\mathcal{M})$ is minimally synchronizing (or it can be made minimally synchronizing by removing one known letter, as shown in Proposition 9);
- primitivity is easily checked by the Protasov-Voynov algorithm ([24], Proposition 2), and primitivity of $\mathcal{M}$ assures that $\mathcal{A}(\mathcal{M})$ is synchronizing (Theorem 3).

The characterization theorem for primitive sets and the above properties are the main ingredients of our randomized algorithm that finds minimally synchronizing automata of 3 and 4 letters (and can eventually be generalized to m letters); to the best of our knowledge, this is the first time where a constructive procedure for finding minimally synchronizing automata is presented. This is described in Section 3 where numerical results are reported. The random construction let us also find new families of 3-letters automata, presented in Section 4, with reset threshold $\Omega(n^2/4)$ and that do not resemble the Černý automaton, thus answering question **Q2**. Finally, in Section 5 we extend a result on perturbed permutation sets obtained by Gonze et al. in [13]: we show that a random perturbed permutation set is primitive with high probability for any matrix size n (and not just when n is a prime number as in [13]) and that its exponent is of order $O(n \log n)$ still with high probability. A further generalization is then presented for sets of random binary matrices: if each entry of each matrix is set to 1 with probability p and to 0 with probability $1-p$, independently from each other, then the set is primitive and has exponent of order $O(n \log n)$ with high probability for any p such that $np - \log n \to \infty$ as $n \to \infty$.
The proofs of the results presented in this paper have been omitted due to length restrictions.

2 Definitions and notation

In this section we briefly go through some definitions and results that will be needed in the rest of the paper.
We indicate with $[n]$ the set $\{1, \dots, n\}$ and with S_k the set of permutations over k elements; with a slight abuse of notation S_k will also denote the set of the $k \times k$ permutation matrices. An n-state automaton $\mathcal{A} = \{A_1, \dots, A_m\}$ can be represented by a labelled digraph on n vertices with a directed edge from vertex i to vertex j labelled by A_k if $A_k(i,j) = 1$; in this case we also use the notation $iA_k = j$. We remind that a matrix M is *irreducible* if there does not exist a permutation matrix P such that PMP^T is block-triangular; a set $\{M_1, \dots, M_m\}$ is said to be *irreducible* iff the matrix $\sum_{i=1}^m M_i$ is irreducible. The *directed graph associated to* an $n \times n$ matrix M is the digraph D_M on n vertices with a directed edge from i to j if $M(i,j) > 0$. A matrix M is irreducible if and only if D_M is strongly connected, i.e. if and only if there exists a directed path between any two given vertices. A *primitive* set $\mathcal{M}$ is a set of m matrices $\{M_1, \dots, M_m\}$ with nonnegative entries where there exists a product $M_{i_1} \cdots M_{i_l} > 0$ entrywise, for $i_1, \dots, i_l \in [m]$. The length of the shortest of these products is called the *exponent* ($exp(\mathcal{M})$) of the set. Irreducibility is a necessary (but not sufficient) condition for a matrix set to be primitive (see [24], Section 1). Primitive sets of NZ-matrices can be characterized as follows:

▶ **Definition 5.** Let $\Omega = \dot{\bigcup}_{l=1}^{k} \Omega_l$ be a partition of $[n]$ with $k \geq 2$. We say that an $n \times n$ matrix M has a *block-permutation structure on the partition* Ω if there exists a permutation $\sigma \in S_k$ such that $\forall\, l = 1, \ldots, k$ and $\forall\, i \in \Omega_l$, if $M(i,j) > 0$ then $j \in \Omega_{\sigma(l)}$. We say that a set of matrices *has a block-permutation structure* if there exists a partition on which *all* the matrices of the set have a block-permutation structure.

▶ **Theorem 6** ([24], Theorem 1)**.** *An irreducible set of NZ matrices of size $n \times n$ is* not *primitive if and only if the set has a block-permutation structure.*

We end this section with the last definition and our first observation (Proposition 8).

▶ **Definition 7.** A matrix A *dominates* a matrix B if $A(i,j) \geq B(i,j),\ \forall\, i,j$.

▶ **Proposition 8.** *Consider an irreducible set $\{M_1, \ldots, M_m\}$ in which every matrix dominates a permutation matrix. If the set has a block-permutation structure, then all the blocks of the partition must have the same size.*

3 Minimally primitive sets and minimally synchronizing automata

In this section we focus on *perturbed permutation* sets, i.e. matrix sets made of permutation matrices where a 0-entry of one matrix is changed into a 1. We represent a set of this kind as $\mathcal{M} = \{P_1, \ldots, P_{m-1}, P_m + \mathbb{I}_{i,j}\}$, where $P_1, \ldots, P_m$ are permutation matrices, $\mathbb{I}_{i,j}$ is a matrix whose (i,j)-th entry is equal to 1 and all the others entries are equal to 0 and $P_m(i,j') = 1$ for a $j' \neq j$. The first result states that we can easily generate minimally synchronizing automata starting from minimally primitive perturbed permutation sets:

▶ **Proposition 9.** *Let $\mathcal{M} = \{P_1, \ldots, P_{m-1}, P_m + \mathbb{I}_{i,j}\}$ be a minimally primitive perturbed permutation set and let $j' \neq j$ be the integer such that $P_m(i,j') = 1$. The automaton $\mathcal{A}(\mathcal{M})$ (see Definition 1) can be written as $\mathcal{A}(\mathcal{M}) = \{P_1, \ldots, P_{m-1}, P_m, M\}$ with $M = P_m + \mathbb{I}_{i,j} - \mathbb{I}_{i,j'}$. If $\mathcal{A}(\mathcal{M})$ is not minimally sychronizing, then $\bar{\mathcal{A}} = \{P_1, \ldots, P_{m-1}, M\}$ is.*

3.1 A randomized algorithm for constructing minimally primitive sets

If we want to find minimally synchronizing automata, Proposition 9 tells us that we just need to generate minimally primitive perturbed permutation sets; in this section we implement a randomized procedure to build them.

Theorem 6 says that a matrix set is *not* primitive if all the matrices share the same block-permutation structure, therefore a set of m matrices is minimally primitive iff every subset of cardinality $m-1$ has a block-permutation structure on a certain partition; this is the condition we will enforce. As we are dealing with perturbed permutation sets, Proposition 8 tells us that we just need to consider partitions with blocks of the same size; if the blocks of the partition have size n/q, we call it a *q-partition* and we say that the set has a *q-permutation structure*. Given $R, C \subset [n]$ and a matrix M, we indicate with $M(R,C)$ the submatrix of M with rows indexed by R and columns indexed by C. The algorithm first generates a set of permutation matrices satisfying the requested block-permutation structures and then a 0-entry of one of the obtained matrices is changed into a 1; while doing this last step, we will make sure that such change will preserve all the block-permutation structures of the matrix. We underline that our algorithm finds perturbed permutation sets that, *if* are primitive, are minimally primitive. Indeed, the construction itself only ensures minimality and not primitivity: this latter property has to be verified at the end.

3.1.1 The algorithm

For generating a set of m matrices $\mathcal{M} = \{M_1, \dots, M_m\}$ we choose m prime numbers $q_1 \geq \cdots \geq q_m \geq 2$ and we set $n = \prod_{i=1}^{m} q_i$. For $j = 1, \dots, m$, we require the set $\{M_1, \dots, M_{j-1}, M_{j+1}, \dots, M_m\}$ (the set obtained from $\mathcal{M}$ by erasing matrix M_j) to have a q_j-permutation structure; this construction will ensure the minimality of the set. More in detail, for all $j=1, \dots, m$ we will enforce the existence of a q_j-partition $\Omega_{q_j} = \dot{\bigcup}_{i=1}^{q_j} \Omega_i^j$ of $[n]$ on which, for all $k \neq j$, the matrix M_k has to have a block-permutation structure. As stated by Definition 5, this request means that for every $k = 1, \dots, m$ and for every $j \neq k$ there must exist a permutation $\sigma_j^k \in S_{q_j}$ such that for all $i=1, \dots, q_j$ and $l \neq \sigma_j^k(i)$, $M_k(\Omega_i^j, \Omega_l^j)$ is a zero matrix.

The main idea of the algorithm is to initialize every entry of each matrix to 1 and then, step by step, to set to 0 the entries that are not compatible with the conditions that we are requiring. As our final goal is to have a set of permutation matrices, at every step we need to make sure that each matrix dominates at least one permutation matrix, despite the increasing number of zeros among its entries.

▶ **Definition 10.** Given a matrix M and a q-partition $\Omega_q = \dot{\bigcup}_{i=1}^{q} \Omega_i^q$, we say that a permutation $\sigma \in S_q$ is *compatible* with M and Ω_q if for all $i = 1, \dots, q$, there exists a permutation matrix Q_i such that

$$M\left(\Omega_i^q, \Omega_{\sigma(i)}^q\right) \geq Q_i. \tag{2}$$

The algorithm itself is formally presented in Listing 1; we here describe in words how it operates. Each entry of each matrix is initialized to 1. The algorithm has two for-loops: the outer one on $j = 1, \dots, m$, where a q_j-partition $\Omega_{q_j} = \dot{\bigcup}_{i=1}^{q_j} \Omega_i^j$ of $[n]$ is uniformly randomly sampled and then the inner one on $k = 1, \dots, m$ with $k \neq j$ where we verify whether there exists a permutation $\sigma_j^k \in S_{q_j}$ that is compatible with M_k and Ω_{q_j}. If it does exist, then we choose one among all the compatible permutations and the algorithm moves to the next step $k + 1$. If such permutation does not exist, then the algorithm exits the inner for-loop and it randomly selects another q_j-partition Ω'_{q_j} of $[n]$; it then repeats the inner loop for $k = 1, \dots, m$ with $k \neq j$ with this new partition. If after $T1$ steps it is choosing a different q_j-partition Ω'_{q_j} the existence, for each $k \neq j$, of a permutation $\sigma_j'^k \in S_{q_j}$ that is compatible with M_k and Ω'_{q_j} is not established, we stop the algorithm and we say that *it did not converge*. If the inner for-loop is completed, then for each $k \neq j$ the algorithm modifies the matrix M_k by keeping unchanged each block $M_k\left(\Omega_i^j, \Omega_{\sigma_j^k(i)}^j\right)$ for $i = 1, \dots, q_j$ and by setting to zero all the other entries of M_k, where σ_j^k is the selected compatible permutation; the matrix M_k has now a block-permutation structure over the sampled partition Ω_{q_j}. The algorithm then moves to the next step $j + 1$. If it manages to finish the outer for-loop, we have a set of binary matrices with the desired block-permutation structures. We then just need to select a permutation matrix $P_k \leq M_k$ for every $k = 1, \dots, m$ and then to randomly change a 0-entry into a 1 in one of the matrices without modifying its block-permutation structures: this is always possible as the blocks of the partitions are non trivial and a permutation matrix has just n positive entries. We finally check whether the set is primitive.

Here below we present the procedures that the algorithm uses:

(a) $[p, P] = Extractperm(M, met)$

This is the key function of the algorithm. It returns $p=1$ if the matrix M dominates a permutation matrix, it returns $p=0$ and $P=M$ otherwise. In the former case it also returns a permutation matrix P selected among the ones dominated by M according to met; if $met = 2$ the matrix P is sampled uniformly at random, while if $met = 3$ we

make the choice of P deterministic. More in detail, the procedure works as follows: we first count the numbers of 1s in each column and in each row of the matrix M. We then consider the row or the column with the least number of 1s; if this number is zero we stop the procedure and we set $p = 0$, as in this case M does not dominate a permutation matrix. If this number is strictly greater than zero, we choose one of the 1-entries of the row or the column attaining this minimum: if $met = 2$ (**method 2**) the entry is chosen uniformly at random while if $met = 3$ (**method 3**) we take the first 1-entry in the lexicographic order. Suppose that such 1-entry is in position (i, j): we set to zero all the other entries in row i and column j and we iterate the procedure on the submatrix obtained from M by erasing row i and column j. We can prove that this procedure is well-defined and in at most n steps it produces the desired output: $p = 0$ if and only if M does not dominate a permutation matrix and, in case $p = 1$, method 2 indeed sample uniformly one of the permutations dominated by M, while method 3 is deterministic and the permutation obtained usually has its 1s distributed around the main diagonal. Method 3 will play an important role in our numerical experiments in Section 3.2 and in the discovery of new families of automata with quadratic reset threshold in Section 4.

(b) $[a, A] = DomPerm(M, \Omega, met)$
It returns $a = 1$ if there exists a permutation compatible with the matrix M and the partition $\Omega = \dot{\bigcup}_{i=1}^{q} \Omega_i^q$, it returns $a = 0$ and $A = M$ otherwise. In the former case it chooses one of the compatible permutations, say σ, according to met and returns the matrix A equal to M but the entries not in the blocks defined by (2) that are set to zero; A has then a block-permutation structure on Ω. More precisely, $DomPerm$ acts in two steps: it first defines a $q \times q$ matrix B such that, for all $i, k = 1, \ldots, q$,

$$B(i,k) = \begin{cases} 1 & \text{if } M(\Omega_i^q, \Omega_k^q) \text{ dominates a permutation matrix} \\ 0 & \text{otherwise} \end{cases};$$

this can be done by calling $ExtractPerm$ with input $M(\Omega_i^q, \Omega_k^q)$ and met for all $i, k = 1, \ldots, q$. At this point, asking if there exists a permutation compatible with M and Ω is equivalent of asking if B dominates a permutation matrix. Therefore, the second step is to call again $[p, P] = ExtractPerm(B, met)$: if $p = 0$ we set $a = 0$ and $A = M$, while if $p = 1$ we set $a = 1$ and A as described before with $\sigma = P$ (i.e. $\sigma(i) = j$ iff $P(i, j) = 1$); indeed the permutation P is one of the permutations compatible with M and Ω.

(c) $Mset = Addone(P_1, ..., P_m)$
It changes a 0-entry of one of the matrices $P_1, ..., P_m$ into a 1 preserving all its block-permutation structures. The matrix and the entry are chosen uniformly at random and the procedure iterates the choice till it finds a compatible entry (which always exists); it then returns the final perturbed permutation set $Mset$.

(d) $pr = Primitive(Mset)$
It returns $pr = 1$ if the matrix set $Mset = \{M_1, \ldots, M_m\}$ is primitive and $pr = 0$ otherwise. It first verifies if the set is irreducible by checking the strong connectivity of the digraph D_N where $N = \sum_{i=1}^{k} M_i$ (see Section 2) via breadth-first search on every node, then if the set is irreducible, primitivity is checked by the Protasov-Voynov algorithm ([24], Section 4).

All the above routines have polynomial time complexity in n, apart from routine $Primitive$ that has time complexity $O(mn^2)$.

▶ Remark.

1. In all our numerical experiments the algorithm always converged, i.e. it always ended before reaching the stopping value $T1$, for $T1$ large enough. This is probably due to the fact that the matrix dimension n grows exponentially as the number of matrices m increases, which produces enough degrees of freedom. We leave the proof of this fact for future work.

2. A recent work of Alpin and Alpina [2] generalizes Theorem 6 for the characterization of primitive sets to sets that are allowed to be reducible and the matrices to have zero columns but not zero rows. Clearly, automata fall within this category. Without going into many details (for which we refer the reader to [2], Theorem 3), Alpin and Alpina show that an n-state automaton is *not* synchronizing if and only if there exist a partition $\dot{\bigcup}_{j=1}^{s}\Omega_j$ of $[n]$ such that it has a block-permutation structure on a *subset* of that partition. This characterization is clearly less restrictive: it just suffices to find a subset $I \subset [s]$ such that for each letter A of the automaton there exists a permutation $\sigma \in S_I$ such that for all $i \in I$, if $A(i,j) = 1$ then $j \in \Omega_{\sigma(i)}$. Our algorithm could leverage this recent result in order to directly construct minimal synchronizing automata. We also leave this for future work.

Listing 1 Algorithm for finding minimally primitive sets.

```
Input: q_1,...,q_m,T1,met
Initialize M_1,...,M_m as all-ones matrices
for j:=1 to m do
        t1=0
        while t1<T1 do
                t1=t1+1
                choose a q_j-partition Omega_j
                for k=1 to m and k!=j do
                   [a,A_k]=DomPerm(M_k,Omega_j,met)
                   if a==0
                         exit inner for-loop
                   end
                end
                if a==1
                   exit while-loop
                end
        end
        if t1==T1
            display 'does not converge', exit procedure
        else
            set M_k=A_k for all k=1,...,m and k!=j
        end
end
for i:=1 to m do
    [p_i,P_i]=Extractperm(M_i,met)
end
Mset=Addone(P_1,...,P_m)
pr=Primitive(Mset)
return Mset, pr
```

3.2 Numerical results

We have seen that once we have a minimally primitive perturbed permutation set, it is easy to generate a minimally synchronizing automaton from it, as stated by Proposition 12. Our goal is to generate automata with large research threshold, but checking this property on many randomly generated instances is prohibitive. Indeed, we recall that computing the reset threshold of an automaton is in general NP-hard [10]. Instead, as a proxy for the reset threshold, we compute the *diameter of the square graph*, which we now introduce:

▶ **Definition 11.** The *square graph* $S(\mathcal{A})$ of an n-state automaton $\mathcal{A}$ is the labelled directed graph with vertex set $V=\{(i,j) : 1 \leq i \leq j \leq n\}$ and edge set E such that $e=\{(i,j),(i',j')\} \in E$ if there exists a letter $A \in \mathcal{A}$ such that $A(i,i') > 0$ and $A(j,j') > 0$, or $A(i,j') > 0$ and $A(j,i') > 0$. In this case, we label the directed edge e by A (multiple labels are allowed). A vertex of type (i,i) is called a *singleton*.

A well-known result ([30], Proposition 1) states that an automaton is synchronizing if and only if in its square graph there exists a path from any non-singleton vertex to a singleton one; the proof of this fact also implies that

$$diam\big(S(\mathcal{A})\big) \leq rt(\mathcal{A}) \leq n \cdot diam\big(S(\mathcal{A})\big), \tag{3}$$

where $diam\big(S(\mathcal{A})\big)$ denotes the *diameter* of $S(\mathcal{A})$ i.e. the maximum length of the shortest path between any two given vertices, taken over all the pairs of vertices. The diameter can be computed in polynomial time, namely $O(mn^2)$ with m the number of letters of the automaton.

We now report our numerical results based on the diameter of the square graph. We compare three methods of generating automata: we call **method 1** the uniform random generation of 2-letter automata made of one permutation matrix and a matrix of rank $n-1$, while **method 2** and **method 3**, already introduced in the previous paragraph, refer to the different ways of extracting a permutation matrix from a binary one in our randomized construction. We set $T1 = 1000$ and for each method and each choice of n we run the algorithm $it(n) = 50n^2$ times, thus producing each time $50n^2$ sets. This choice for $it(n)$ has been made by taking into account two facts: on one hand, it is desirable to keep constant the rate $it(n)/k_m(n)$ between the number of sampled sets $it(n)$ and the cardinality $k_m(n)$ of the set of the perturbed permutation sets made of m matrices. Unfortunately, $k_m(n+1)/k_m(n)$ grows approximately as n^m and so $k_m(n)$ explodes very fast. On the other hand, we have to deal with the limited computational speed of our computers. The choice of $it(n) = 50n^2$ comes as a compromise between these two issues, at least when $n \leq 70$.
Among the $it(n)$ generated sets, we select the primitive ones and we generate their associated automata (Definition 1); we then check which ones are not minimally synchronizing and we make them minimally synchronizing by using Proposition 9. Finally, we compute the square graph diameter of all the minimally synchronizing automata obtained. Figure 2 reports on the y axis the maximal square graph diameter found for each method and for each matrix dimension n when n is the product of three prime numbers (left picture) and when it is a product of four prime numbers (right picture). We can see that our randomized construction manages to reach higher values of the square graph diameter than the mere random generation; in particular, method 3 reaches quadratic diameters in case of three matrices.

We also report in Figure 3 (left) the behavior of the *average* diameter of the minimally synchronizing automata generated on $50n^2$ iterations when n is the product of three prime numbers: we can see that in this case method 2 does not perform better than method 1, while method 3 performs just slightly better. This behavior could have been expected since our primary goal was to randomly generate *at least one* slowly synchronizing automata; this is indeed what happens with method 3, that manages to reach quadratic reset thresholds most of the times.
A remark can be done on the percentage of the generated sets that are *not* primitive; this is reported in Figure 3 (right), where we divide nonprimitive sets into two categories: reducible sets and *imprimitive* sets, i.e. irreducible sets that are not primitive. We can see that the percentage of nonprimitive sets generated by method 1 goes to 0 as n increases, behavior

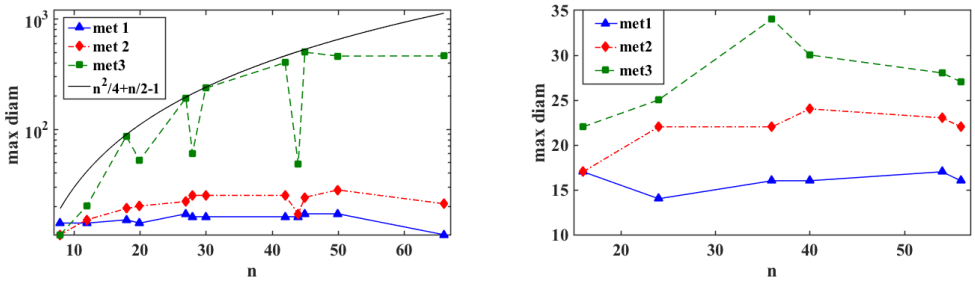


Figure 2 Comparison of our methods (met 2 and 3) with the naive method (met 1) with respect to the maximal diameter found on $50n^2$ iterations. Left: n is the product of three prime numbers; the y axis is in logarithmic scale. Right: n is the product of four prime numbers.

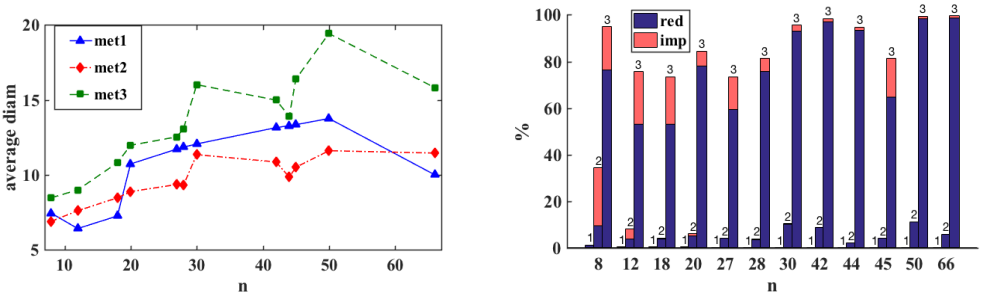


Figure 3 Left: Average diameter found by methods 1, 2 and 3 when n is the product of three prime numbers. Right: Percentage of nonprimitive sets (divided into reducible and imprimitive sets) generated by methods 1, 2 and 3 (indicated above each bar) when n is the product of three prime numbers. For instance, on sets of dimension $n = 20$, method 1 generates 0.35% of nonprimitive sets (0.35% reducible, 0% imprimitive), method 2 generates 6.15% of nonprimitive sets (5.18% reducible, 0.97% imprimitive) and method 3 generates 84.5% of nonprimitive sets (77.9% reducible, 6.6% imprimitive).

that we expected (see Section 5, Theorem 18), while method 2 seems to always produce a not negligible percentage of nonprimitive sets, although quite small. The behavior is reversed for method 3: most of the generated sets are not primitive. This can be interpreted as a good sign. Indeed, nonprimitive sets can be seen as sets with *infinite* exponent; as we are generating a lot of them with method 3, we intuitively should expect that, when a primitive set is generated, it has high chances to have large diameter.

The slowly synchronizing automata found by our randomized construction are presented in the following section. We believe that some parameters of our construction, as the way a permutation matrix is extracted from a binary one or the way the partitions of $[n]$ are selected, could be further tuned or changed in order to generate new families of slowly synchronizing automata; for example, we could think about selecting the ones in the procedure *Extractperm* according to a given distribution. We leave this for future work.

4 New families of automata with quadratic reset threshold

We present here four new families of (minimally synchronizing) 3-letter automata with square graph diameter of order $\Omega(n^2/4)$, which represents a lower bound for their reset threshold. These families are all made of two symmetric permutation matrices and a matrix of rank $n-1$ that merges two states and fixes all the others (a perturbed identity matrix): they thus

lie within the class of automata *with simple idempotents*, class introduced by Rystsov in [25] in which every letter A of the automaton is requested either to be a permutation or to satisfy $A^2 = A$. These families have been found via the randomized algorithm described in Section 3.1 using the deterministic procedure to extract a permutation matrix from a binary one (method 3). The following proposition shows that primitive sets made of a perturbed identity matrix and two symmetric permutations must have a very specific shape; we then present our families, prove closed formulas for their square graph diameter and finally state a conjecture on their reset thresholds. With a slight abuse of notation we identify a permutation matrix Q with its underlying permutation, that is we say that $Q(i) = j$ if and only if $Q(i, j) = 1$; the identity matrix is denoted by I. Note that a permutation matrix is symmetric if and only if its cycle decomposition is made of fixed points and cycles of length 2.

▶ **Proposition 12.** *Let $\mathcal{M}_{ij} = \{\bar{I}_{ij}, Q_1, Q_2\}$ be a matrix set of $n \times n$ matrices where $\bar{I}_{ij} = I + \mathbb{I}_{ij}$, $j \neq i$, is a perturbed identity and Q_1 and Q_2 are two symmetric permutations. If $\mathcal{M}$ is irreducible then, up to a relabelling of the vertices, Q_1 and Q_2 have the following form:*

- *if n is even*

$$Q_1(i) = \begin{cases} 1 & \text{if } i = 1 \\ i+1 & \text{if } i \text{ even}, 2 \leq i \leq n-2 \\ i-1 & \text{if } i \text{ odd}, 3 \leq i \leq n-1 \\ n & \text{if } i = n \end{cases}, \quad Q_2(i) = \begin{cases} i-1 & \text{if } i \text{ even} \\ i+1 & \text{if } i \text{ odd} \end{cases} \tag{4}$$

or

$$Q_1(i) = \begin{cases} n & \text{if } i = 1 \\ i+1 & \text{if } i \text{ even}, 2 \leq i \leq n-2 \\ i-1 & \text{if } i \text{ odd}, 3 \leq i \leq n-1 \\ 1 & \text{if } i = n \end{cases}, \quad Q_2(i) = \begin{cases} i-1 & \text{if } i \text{ even} \\ i+1 & \text{if } i \text{ odd} \end{cases} \tag{5}$$

- *if n is odd*

$$Q_1(i) = \begin{cases} 1 & \text{if } i = 1 \\ i+1 & \text{if } i \text{ even} \\ i-1 & \text{if } i \text{ odd}, 3 \leq i \leq n \end{cases}, \quad Q_2(i) = \begin{cases} i-1 & \text{if } i \text{ even} \\ i+1 & \text{if } i \text{ odd}, 1 \leq i \leq n-2 \\ n & \text{if } i = n \end{cases}. \tag{6}$$

▶ **Proposition 13.** *A matrix set $\mathcal{M}_{ij} = \{\bar{I}_{ij}, Q_1, Q_2\}$ of type (5) is never primitive.*

▶ **Definition 14.** We define $\mathcal{A}_{ij} = \{\underline{I}_{ij}, Q_1, Q_2\}$ to be the associated automaton $\mathcal{A}(\mathcal{M}_{ij})$ of $\mathcal{M}_{ij}$ (see Definition 1), where $\underline{I}_{ij} = I + \mathbb{I}_{ij} - \mathbb{I}_{ii}$.

It is clear that $\mathcal{A}_{ij}$ is with simple idempotents. Figure 4 represents $\mathcal{A}_{1,6}$ with $n = 8$. We set now $\mathcal{E}_n = \mathcal{A}_{1,n-2}$ for $n = 4k$ and $k \geq 2$, $\mathcal{E}'_n = \mathcal{A}_{1,n-4}$ for $n = 4k+2$ and $k \geq 2$, $\mathcal{O}_n = \mathcal{A}_{\frac{n-1}{2},\frac{n+1}{2}}$ for $n = 4k+1$ and $k \geq 1$, $\mathcal{O}'_n = \mathcal{A}_{\frac{n-1}{2},\frac{n+1}{2}}$ for $n = 4k+3$ and $k \geq 1$. The following theorem holds:

▶ **Theorem 15.** *The automaton $\mathcal{E}_n$ has square graph diameter (SGD) of $(n^2 + 2n - 4)/4$, $\mathcal{E}'_n$ has SGD of $(n^2 + 2n - 12)/4$, $\mathcal{O}_n$ has SGD of $(n^2 + 3n - 8)/4$ and $\mathcal{O}'_n$ has SGD of $(n^2 + 3n - 6)/4$. Therefore all the families $\mathcal{E}_n$, $\mathcal{E}'_n$, $\mathcal{O}_n$ and $\mathcal{O}'_n$ have reset threshold of $\Omega(n^2/4)$.*

Figure 5 represents the square graph of the automaton $\mathcal{E}_8$, where its diameter is colored in red. All the singletons but the one that belongs to the diameter have been omitted.

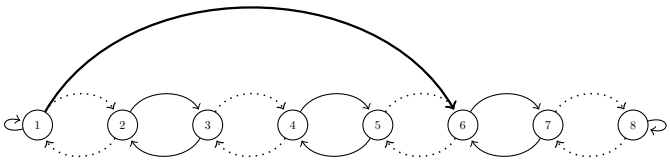


Figure 4 The automata $\mathcal{A}_{1,6}$ with $n = 8$; $rt(\mathcal{A}_{1,6})$=31. Dashed arrows refer to matrix Q_2, normal arrows to matrix Q_1 and bold arrows to matrix $\underline{I}_{1,6}$ where its selfloops have been omitted.

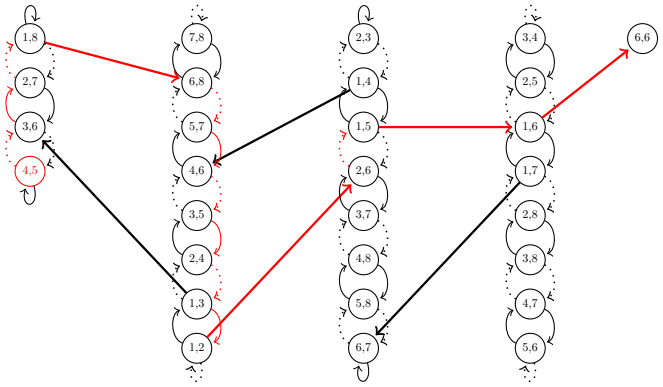


Figure 5 Square graph of automaton $\mathcal{E}_8$, $diam\big(S(\mathcal{E}_8)\big) = 19$. Normal arrows refer to matrix Q_1, dotted arrows to matrix Q_2 and bold arrows to matrix $\underline{I}_{1,6}$, where its selfloops have been omitted. The red path is the diameter.

▶ **Conjecture 16.** *The automaton $\mathcal{E}_n$ has reset threshold of $(n^2-2)/2$, $\mathcal{E}'_n$ has reset threshold of $(n^2-10)/2$ and $\mathcal{O}_n$ and $\mathcal{O}'_n$ have reset threshold of $(n^2-1)/2$. Furthermore, they represent the automata with the largest possible reset threshold among the family $\mathcal{A}_{ij}$ for respectively $n = 4k$, $n = 4k+2$, $n = 4k+1$ and $n = 4k+3$.*

We end this section by remarking that, despite the fact that the randomized construction for minimally primitive sets presented in Section 3 works just when the matrix size n is the product of at least three prime numbers, here we found an extremal automaton of quadratic reset threshold for *any* value of n.

5 Primitivity with high probability

We call *random perturbed permutation* set a perturbed permutation set of $m \geq 2$ matrices constructed with the following randomized procedure:

▶ **Procedure 17.**

1. *We sample m permutation matrices $\{P_1, \ldots, P_m\}$ independently and uniformly at random from the set S_n of all the permutations over n elements;*
2. *A matrix P_i is uniformly randomly chosen from the set $\{P_1, \ldots, P_m\}$. Then, one of its 0-entry is uniformly randomly selected among its 0-entries and changed into a 1. It becomes then a perturbed permutation matrix $\bar{P}_i$;*
3. *The final random perturbed permutation set is the set $\{P_1, \ldots, P_{i-1}, \bar{P}_i, P_{i+1}, \ldots, P_m\}$.*

This procedure is also equivalent to choosing independently and uniformly at random $m-1$ permutation matrices from S_n and one *perturbed* permutation matrix from $\bar{S}_n$ with $\bar{S}_n = \{\bar{P} : \bar{P} = P + \mathbb{I}_{i,j},\ P \in S_n,\ P(i,j') = 1,\ j \neq j'\}$. We say that a property X holds for a random matrix set with *high probability* if the probability that property X holds tends to 1 as the matrix dimension n tends to infinity.

▶ **Theorem 18.** *A random perturbed permutation set constructed via Procedure 17 is primitive and has exponent of order $O(n \log n)$ with high probability.*

Theorem 18 can be extended to random sets of binary matrices. It is clear that focusing just on binary matrices is not restrictive as in the definition of primitivity what counts is just the position of the positive entries within the matrices and not their actual values. Let $B(p)$ denote a random binary matrix where each entry is independently set to 1 with probability p and to 0 with probability $(1-p)$ and let $\mathcal{B}_m(p)$ denote a set of $m \geq 2$ matrices obtained independently in this way. Under some mild assumptions over p, we still have primitivity with high probability:

▶ **Theorem 19.** *For any fixed integer $m \geq 2$, if $np - \log n \longrightarrow \infty$ as $n \to \infty$, then $\mathcal{B}_m(p)$ is primitive and $exp\big(\mathcal{B}_m(p)\big) = O(n \log n)$ with high probability.*

It is interesting to compare this result with the one obtained by Gerencsér et al. in [12]: they prove that, if $exp(n)$ is the maximal value of the exponent among all the binary primitive sets of $n \times n$ matrices, then $\lim_{n\to\infty}(\log exp(n))/n = (\log 3)/3$. This implies that, for n big enough, there must exist some primitive sets whose exponent is close to $3^{n/3}$, but these sets must be very few as Theorem 19 states that they are almost impossible to be detected by a mere random generation. We conclude with a result on when a set of two random binary matrices is *not* primitive with high probability:

▶ **Proposition 20.** *For any fixed integer $m \geq 2$, if $2np - \log n \longrightarrow -\infty$ as $n \to \infty$, then $\mathcal{B}_m(p)$ is reducible with high probability. This implies that $\mathcal{B}_m(p)$ is not primitive with high probability.*

6 Conclusion

In this paper we have proposed a randomized construction for generating slowly minimally synchronizing automata. Our strategy relies on a recent characterization of primitive sets (Theorem 6), together with a construction (Definition 1 and Theorem 3) allowing to build (slowly minimally) synchronizing automata from (slowly minimally) primitive sets. We have obtained four new families of automata with simple idempotents with reset threshold of order $\Omega(n^2/4)$. The *primitive sets approach* to synchronizing automata seems promising and we believe that out randomized construction could be further refined and tweaked in order to produce other interesting automata with large reset threshold, for example by changing the way a permutation matrix is extracted by a binary one. As mentioned at the end of Section 3.1, it would be also of interest to apply the minimal construction directly to automata by leveraging the recent result of Alpin and Alpina ([2], Theorem 3).

References

1 N. Alon and J. H. Spencer. *The Probabilistic Method.* Wiley Publishing, 4th edition, 2016.
2 Yu. Alpin and V. Alpina. Combinatorial properties of entire semigroups of nonnegative matrices. *Journal of Mathematical Sciences*, 207(5):674–685, 2015.

3 D. S. Ananichev, M. V. Volkov, and V. V. Gusev. Primitive digraphs with large exponents and slowly synchronizing automata. *Journal of Mathematical Sciences*, 192(3):263–278, 2013.

4 M.-P. Béal, M. V. Berlinkov, and D. Perrin. A quadratic upper bound on the size of a synchronizing word in one-cluster automata. In *International Journal of Foundations of Computer Science*, pages 277–288, 2011.

5 M.V. Berlinkov. *On the Probability of Being Synchronizable*, pages 73–84. Springer International Publishing, 2016.

6 V.D. Blondel, R.M. Jungers, and A. Olshevsky. On primitivity of sets of matrices. *Automatica*, 61:80–88, 2015.

7 Y.-B. Chen and D. J. Ierardi. The complexity of oblivious plans for orienting and distinguishing polygonal parts. *Algorithmica*, 14(5):367–397, 1995.

8 P.-Y. Chevalier, J.M. Hendrickx, and R.M. Jungers. Reachability of consensus and synchronizing automata. In *4th IEEE Conference on Decision and Control*, pages 4139–4144, 2015.

9 M. de Bondt, H. Don, and H. Zantema. Dfas and pfas with long shortest synchronizing word length. In *Developments in Language Theory*, pages 122–133, 2017.

10 D. Eppstein. Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19(3):500–510, 1990.

11 P. Frankl. An extremal problem for two families of sets. *European Journal of Combinatorics*, 3:125–127, 1982.

12 B. Gerencsér, V. V. Gusev, and R. M. Jungers. Primitive sets of nonnegative matrices and synchronizing automata. *Siam Journal on Matrix Analysis and Applications*, 39(1):83–98, 2018.

13 F. Gonze, B. Gerencsér, and R.M. Jungers. Synchronization approached through the lenses of primitivity. In *35th Benelux Meeting on Systems and Control*, 2016.

14 F. Gonze, V.V. Gusev, B. Gerencsér, R.M. Jungers, and M.V. Volkov. *On the Interplay Between Babai and Černý's Conjectures*, pages 185–197. Springer International Publishing, 2017.

15 V. V. Gusev and E. V. Pribavkina. Reset thresholds of automata with two cycle lengths. In *Implementation and Application of Automata*, pages 200–210, 2014.

16 J. Kari. Synchronizing finite automata on eulerian digraphs. *Theoretical Computer Science*, 295(1):223–232, 2003.

17 A. Kisielewicz and M. Szykuła. Synchronizing automata with extremal properties. In *Mathematical Foundations of Computer Science 2015*, pages 331–343, 2015.

18 A. Mateescu and A. Salomaa. Many-valued truth functions, Černý's conjecture and road coloring. In *EATCS Bull.*, page 134–150, 1999.

19 M. Michalina Dzyga, R. Ferens, V.V. Gusev, and M. Szykuła. Attainable values of reset thresholds. In *42nd International Symposium on Mathematical Foundations of Computer Science*, volume 83, pages 40:1–40:14, 2017.

20 C. Nicaud. Fast synchronization of random automata. In *Approximation, Randomization, and Combinatorial Optimization*, volume 60 of *Leibniz International Proceedings in Informatics*, pages 43:1–43:12, 2016.

21 J. Olschewski and M. Ummels. *The Complexity of Finding Reset Words in Finite Automata*, pages 568–579. Springer Berlin Heidelberg, 2010.

22 J.-E. Pin. On two combinatorial problems arising from automata theory. In *Proceedings of the International Colloquium on Graph Theory and Combinatorics*, volume 75, pages 535–548, 1983.

23 V.Yu. Protasov and R.M. Jungers. Lower and upper bounds for the largest lyapunov exponent of matrices. *Linear Algebra and its Applications*, 438:4448–4468, 2013.

24 V.Yu. Protasov and A.S. Voynov. Sets of nonnegative matrices without positive products. *Linear Algebra and its Applications*, 437:749–765, 2012.

25 I. K. Rystsov. Estimation of the length of reset words for automata with simple idempotents. *Cybernetics and Systems Analysis*, 36(3):339–344, 2000.

26 M. Szykuła. Improving the upper bound the length of the shortest reset words. In *STACS*, 2018.

27 M. Szykuła and V. Vorel. An extremal series of eulerian synchronizing automata. In *Developments in Language Theory*, pages 380–392, 2016.

28 J. Černý. Poznámka k homogénnym eksperimentom s konečnými automatami. *Matematicko-fysikalny Casopis SAV*, 14:208–216, 1964.

29 M. V. Volkov. Synchronizing automata preserving a chain of partial orders. In *Implementation and Application of Automata*, pages 27–37, 2007.

30 M.V. Volkov. *Synchronizing automata and the Černý conjecture*, volume 5196, pages 11–27. Springer, 2008.

Counting Homomorphisms to Trees Modulo a Prime

Andreas Göbel
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany

J. A. Gregor Lagodzinski
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany

Karen Seidel
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany

Abstract

Many important graph theoretic notions can be encoded as counting graph homomorphism problems, such as partition functions in statistical physics, in particular independent sets and colourings. In this article we study the complexity of $\#_p\text{HomsTo}H$, the problem of counting graph homomorphisms from an input graph to a graph H modulo a prime number p. Dyer and Greenhill proved a dichotomy stating that the tractability of non-modular counting graph homomorphisms depends on the structure of the target graph. Many intractable cases in non-modular counting become tractable in modular counting due to the common phenomenon of cancellation. In subsequent studies on counting modulo 2, however, the influence of the structure of H on the tractability was shown to persist, which yields similar dichotomies.

Our main result states that for every tree H and every prime p the problem $\#_p\text{HomsTo}H$ is either polynomial time computable or $\#_p\mathsf{P}$-complete. This relates to the conjecture of Faben and Jerrum stating that this dichotomy holds for every graph H when counting modulo 2. In contrast to previous results on modular counting, the tractable cases of $\#_p\text{HomsTo}H$ are essentially the same for all values of the modulo when H is a tree. To prove this result, we study the structural properties of a homomorphism. As an important interim result, our study yields a dichotomy for the problem of counting weighted independent sets in a bipartite graph modulo some prime p. These results are the first suggesting that such dichotomies hold not only for the one-bit functions of the modulo 2 case but also for the modular counting functions of all primes p.

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

Keywords and phrases Algorithms, Theory, Graph Homomorphisms, Counting Modulo a Prime, Complexity Dichotomy

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.49

Related Version A full version of the paper is available at `https://arxiv.org/abs/1802.06103`.

Acknowledgements The first author would like to thank Leslie Ann Goldberg and David Richerby for fruitful discussions during the early stages of this work.

1 Introduction

Edge preserving functions between the vertices of two graphs, known as *graph homomorphisms*, generate a powerful language expressing important notions; examples include constraint satisfaction problems and partition functions in statistical physics. As such, the computational

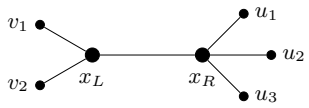


Figure 1 The graph H will be our recurring example and the labelling of the vertices is justified later in the introduction.

complexity of graph homomorphism problems has been studied extensively from a wide range of views. Early results include that of Hell and Nešetřil [14], who study the complexity of HomsToH, the problem of deciding if there exists a homomorphism from an input graph G to a fixed graph H. They show the following dichotomy: if H is bipartite or has a loop, the problem is in P and in every other case HomsToH is NP-complete. In particular, this is of interest since a result of Ladner [15] shows that if P $\neq$ NP, then there exist problems that are neither in P nor NP-hard.

Dyer and Greenhill [5] show a dichotomy for the problem #HomsToH, the problem of counting the homomorphisms from an input graph G to H. Their theorem states that #HomsToH is tractable if H is a complete bipartite graph or a complete graph with loops on all vertices; otherwise #HomsToH is # P-complete. This dichotomy was progressively extended to weighted sums of homomorphisms with integer weights, by Bulatov and Gohe [1]; with real weights, by Goldberg et al. [11]; finally, with complex weights, by Cai, Chen and Lu [2].

We study the complexity of counting homomorphisms modulo a prime p. The set of homomorphisms from the input graph G to the target graph H is denoted by $\mathrm{Hom}(G \to H)$. For each pair of fixed parameters p and H, we study the computational problem $\#_p$HomsToH, that is the problem of computing $|\mathrm{Hom}(G \to H)|$ modulo p. The value of p and the structure of the target graph H influence the complexity of $\#_p$HomsToH. Consider the graph H in Figure 1. Our results show that $\#_p$HomsToH is computable in polynomial time when $p = 2, 3$ while it is hard for any other prime p.

Our main goal is to fully characterise the complexity of $\#_p$HomsToH in a dichotomy theorem. In this manner we aim to determine for which pair of parameters (H, p) the problem is tractable and show that for every other pair of parameters the problem is hard. As the theorem of Ladner [15] extends to the modular counting problems, it is not obvious that there are no instances of $\#_p$HomsToH with an intermediate complexity.

The first study of graph homomorphisms under the setting of modular counting has been conducted by Faben and Jerrum [7]. Their work is briefly described in the following and we assume the reader to be familiar with the notion of an automorphism and its order. We provide the formal introduction in the full version. Given a graph H and an automorphism ϱ of H, H^{ϱ} denotes the subgraph of H induced by the fixpoints of ϱ. We write $H \Rightarrow_k H'$ if there is an automorphism ϱ of order k of H such that $H^{\varrho} = H'$ and we write $H \Rightarrow_k^* H'$ if either H is isomorphic to H' (written $H \cong H'$) or, for some positive integer t, there are graphs $H_1, \ldots, H_t$ such that $H \cong H_1$, $H_1 \Rightarrow_k \cdots \Rightarrow_k H_t$, and $H_t \cong H'$.

Faben and Jerrum showed [7, Lemma 3.3] that if the order of ϱ is a prime p, then $|\mathrm{Hom}(G \to H)|$ is equivalent to $|\mathrm{Hom}(G \to H^{\varrho})|$ modulo p. Furthermore, they showed [7, Theorem 3.7] that there is (up to isomorphism) exactly one graph H^{*p} without automorphisms of order p, such that $H \Rightarrow_p^* H^{*p}$. This graph H^{*p} is called the *order p reduced form* of H (see Figure 4 of the full version). If H^{*p} falls into the polynomial computable cases of the theorem of Dyer and Greenhill, then $\#_p$HomsToH is computable in polynomial time as

well. For $p = 2$, Faben and Jerrum conjectured that these are the only instances computable in polynomial time and that in every other case $\#_2\text{HOMSTO}H$ is $\#_2\,\mathsf{P}$-complete, where $\#_k\,\mathsf{P}$ is the "canonical" hardness class for modular counting problems (see Section 1.1).

▶ **Conjecture 1.1** (Faben and Jerrum [7])**.** Let H be a graph. If its order 2 reduced form H^{*2} has at most one vertex, then $\#_2\text{HOMSTO}H$ is in FP; otherwise, $\#_2\text{HOMSTO}H$ is $\#_2\,\mathsf{P}$-complete.

Faben and Jerrum [7, Theorem 3.8] underlined their conjecture by proving it for the case in which H is a tree. In subsequent works this proof was extended to cactus graphs in [9] and to square-free graphs in [10], by Göbel, Goldberg and Richerby.

The present work follows a direction orthogonal to the aforementioned. Instead of proving the conjecture for richer classes of graphs, we show a dichotomy for all primes, starting again by studying trees.

▶ **Theorem 1.2.** *Let p be a prime and let H be a graph, such that its order p reduced form H^{*p} is a tree. If H^{*p} is a star, then $\#_p\text{HOMSTO}H$ is computable in polynomial time; otherwise, $\#_p\text{HOMSTO}H$ is $\#_p\,\mathsf{P}$-complete.*

Our results are the first to suggest that the conjecture of Faben and Jerrum might apply to counting graph homomorphisms modulo every prime p instead of counting modulo 2. This suggestion, however, remains hypothetical. Borrowing the words of Dyer, Frieze and Jerrum [4]: "One might even rashly conjecture" it "(though we shall not do so)".

To justify our title we give the following corollary, stating a dichotomy for all trees H.

▶ **Corollary 1.3.** *Let p be a prime and let H be a tree. If the order p reduced form H^{*p} of H is a star, then $\#_p\text{HOMSTO}H$ is computable in polynomial time; otherwise, $\#_p\text{HOMSTO}H$ is $\#_p\,\mathsf{P}$-complete.*

We illustrate Theorem 1.2 using the following discussion on Figure 1. The order 2 and the order 3 reduced form of H both are the graph with one vertex, whereas for any other prime the graph stays as such.

The polynomial computable cases follow directly from the results of Faben and Jerrum. Thus, to prove Theorem 1.2 it suffices to show that $\#_p\text{HOMSTO}H$ is $\#_p\,\mathsf{P}$-complete for every tree H that is not a star and has no automorphism of order p. The reductions in [7, 9, 10] show hard instances of $\#_2\text{HOMSTO}H$ by starting from $\#_2$IS, the problem of computing $|\mathcal{I}(G)| \pmod 2$, where $\mathcal{I}(G)$ is the set of independent sets of G. $\#_2$IS was shown to be $\#_2\,\mathsf{P}$ complete by Valiant [18]. Later on, Faben [6] extended this result by proving $\#_k$IS to be $\#_k\,\mathsf{P}$-complete for all integers k. For reasons to be explained in Section 1.3 we do not use this problem as a starting point for our reductions.

We turn our attention to $\#_p$BIS, the problem of counting the independent sets of a bipartite graph modulo p. In the same work Faben [6] includes a construction to show hardness for $\#_p$BIS. We employ the weighted version $\#_p\text{BIS}_{\lambda_\ell,\lambda_r}$ as a starting point for our reduction extending the research on #BIS.

▶ **Problem 1.4.** $\#_p\text{BIS}_{\lambda_\ell,\lambda_r}$

Parameter.	p prime and $\lambda_\ell, \lambda_r \in \mathbb{Z}_p$.
Input.	Bipartite graph $G = (V_L, V_R, E)$.
Output.	$Z_{\lambda_\ell,\lambda_r}(G) = \sum_{I\in\mathcal{I}(G)} \lambda_\ell^{\lvert V_L\cap I\rvert}\lambda_r^{\lvert V_R\cap I\rvert} \pmod p$.

In fact, we obtain the following dichotomy.

▶ **Theorem 1.5.** *Let p be a prime and let λ_ℓ, $\lambda_r \in \mathbb{Z}_p$. If $\lambda_\ell \equiv 0 \pmod p$ or $\lambda_r \equiv 0 \pmod p$, then $\#_p\text{BIS}_{\lambda_\ell,\lambda_r}$ is computable in polynomial time. Otherwise, $\#_p\text{BIS}_{\lambda_\ell,\lambda_r}$ is $\#_p\,\mathsf{P}$-complete.*

In order to prove hardness for $\#_p\text{HomsTo}H$ we employ a reduction in three phases: (i) we reduce the "canonical" $\#_p\,\mathsf{P}$-complete problem $\#_p\text{SAT}$ to $\#_p\text{BIS}_{\lambda_\ell,\lambda_r}$; (ii) we reduce $\#_p\text{BIS}_{\lambda_\ell,\lambda_r}$ to $\#_p\text{PartLabHomsTo}H$, a restricted version of $\#_p\text{HomsTo}H$ which we define in Section 1.3; (iii) we reduce $\#_p\text{PartLabHomsTo}H$ to $\#_p\text{HomsTo}H$.

Section 1.1 provides background knowledge on modular counting. In Section 1.2 we will discuss some related work. A high level proof of our three way reduction is provided in Section 1.3. There we also explain the technical obstacles arising from values of the modulo $p > 2$ and how we overcome them by generalising the techniques used for the case $p = 2$. First, we explain step (i), the reduction from $\#_p\text{SAT}$ to $\#_p\text{BIS}_{\lambda_\ell,\lambda_r}$. Afterwards, we describe step (iii), the reduction from $\#_p\text{PartLabHomsTo}H$ to $\#_p\text{HomsTo}H$ establishing the required notation for the subsequent illustration of step (ii), the reduction from $\#_p\text{BIS}_{\lambda_\ell,\lambda_r}$ to $\#_p\text{PartLabHomsTo}H$. In Section 1.4 we discuss the limits of our techniques, which do not yield a dichotomy modulo any integer k.

1.1 Modular counting

Modular counting was originally studied from the decision problem's point of view. Here, the objective is to determine if the number of solutions is non-zero modulo k. The complexity class $\oplus\,\mathsf{P}$ was first studied by Papadimitriou and Zachos [16] and by Goldschlager and Parberry [12]. $\oplus\,\mathsf{P}$ consists of all problems of the form "is $f(x)$ odd or even?", where $f(x)$ is a function in $\#\,\mathsf{P}$. A result of Toda [17] states that every problem in the polynomial time hierarchy reduces in polynomial time to some problem in $\oplus\,\mathsf{P}$. This result suggests that $\oplus\,\mathsf{P}$-completeness represents strong evidence for intractability.

For an integer k the complexity class $\#_k\,\mathsf{P}$ consists of all problems of the form "compute $f(x)$ modulo k", where $f(x)$ is a function in $\#\,\mathsf{P}$. In the special case of $k = 2$, $\#_2\,\mathsf{P} = \oplus\,\mathsf{P}$, as the instances of $\#_2\,\mathsf{P}$ require a one bit answer. Throughout this paper though, instead of the more traditional notation $\oplus\,\mathsf{P}$, we will use $\#_2\,\mathsf{P}$ to emphasise our interest in computing functions.

If a counting problem can be solved in polynomial time, the corresponding decision and modular counting problems can also be solved in polynomial time. The converse, though, does not necessarily hold. The reason is that efficient counting algorithms rely usually on an exponential number of cancellations that occur in the problem, e.g. compute the determinant of a non-negative matrix. The modulo operator introduces a natural setting for such cancellations to occur.

For instance, consider the $\#\,\mathsf{P}$-complete problem of counting proper 3-colourings of a graph G in the modulo 3 (or even modulo 6) setting. 3-colourings of a graph assigning all three colours can be grouped in sets of size 6, since there are $3! = 6$ permutations of the colours. Thus, the answer to these instances is always a multiple of 6, and therefore "cancels out". It remains to compute the number of 3-colourings assigning less than 3 colours. For the case of using exactly 2 colours we distinguish the following two cases: G is not bipartite and there are no such colourings; G is bipartite and the number of 3-colourings of G that use exactly 2 colours is $3(2^c)$, where c is the number of components of G. Finally, computing the number of proper 3-colourings of G that use exactly one colour is an easy task. Either G has an edge and there are no such colourings, or G has no edges and for every vertex there are 3 colours to choose from.

Valiant [18] observed a surprising phenomenon in the tractability of modular counting problems. He showed that for a restricted version of 3-SAT computing the number of solutions modulo 7 is in FP, but computing this number modulo 2 is $\#_2$ P-complete. This mysterious number 7 was later explained by Cai and Lu [3], who showed that the k-SAT version of Valiant's problem is tractable modulo any prime factor of $2^k - 1$.

1.2 Related work

We have already mentioned earlier work on counting graph homomorphisms. In this section we highlight the work of Faben [6] and the work of Guo et al. [13] on the complexity of the modular counting variant of the constraint satisfaction problem.

▶ **Problem 1.6.** $\#_k\mathrm{CSP}(\mathcal{F})$

Parameter.	$k \in \mathbb{Z}_{>0}$ and a set of functions $\mathcal{F} = \{f_1, \ldots, f_m\}$, where for each $j \in [m]$, $f_j : \{0,1\}^{r_j} \to \mathbb{Z}_p$ and $r_j \in \mathbb{Z}_{>0}$.
Input.	Finite set of constraints over Boolean variables $x_1, \ldots, x_n$ of the form $f_{j_l}(x_{i_{l,1}}, x_{i_{l,2}}, \ldots, x_{i_{l,r_{j_l}}})$.
Output.	$\sum_{x_1,\ldots,x_n \in \{0,1\}} \prod_l f_{j_l}(x_{i_{l,1}}, x_{i_{l,2}}, \ldots, x_{i_{l,r_{j_l}}}) \pmod k$.

Faben showed a dichotomy theorem [6, Theorem 4.11] when the functions in $\mathcal{F}$ have Boolean domain and Boolean range, i.e. $f : \{0,1\} \to \{0,1\}$. Guo et al. extended this dichotomy [13, Theorem 4.1] to $\#_k$CSP, when the functions in $\mathcal{F}$ have Boolean domain $\{0,1\}$ but range in $\mathbb{Z}_k$.

Constraint satisfaction problems generalise graph homomorphism problems, when the domain of the constraint functions is arbitrarily large. In order to illustrate that $\#_k$CSP is a generalisation of $\#_k$HomsToH, let G be an input for $\#_k$HomsToH, for which we describe an equivalent $\#_k$CSP instance. The domain of the constraint satisfaction problem is $D = V(H)$ and $\mathcal{F}$ contains a single binary relation R_H, with $R_H(u,v) = 1$ whenever $(u,v) \in E(H)$ and $R_H(u,v) = 0$ otherwise. Thus, $\#_k$HomsToH is an instance of $\#_k\mathrm{CSP}(\{R_H\})$. The input of $\#_k\mathrm{CSP}(\{R_H\})$ contains a variable x_v for every vertex $v \in V(G)$ and a constraint $R_H(x_u, x_v)$ for every edge $(u,v) \in E(G)$. As can be observed from the construction, every valid homomorphism $\sigma : V(G) \to V(H)$ corresponds to an assignment of the variables $\{x_v\}_{v \in V(G)}$ satisfying every constraint in the CSP.

These results of Faben and of Guo et al. are incomparable to ours. We consider prime values of the modulo and a single binary relation, however the domain of our relations is arbitrarily large. Furthermore, the results of Faben [6, Theorem 4.11] show that the constraint language $\mathcal{F}$ for which $\#_2$CSP is tractable is richer than the constraint language for which $\#_k$CSP is tractable, where $k > 2$. In contrast, our results show that the dichotomy criterion of $\#_p$HomsToH remains the same for all primes p, when H is a tree.

1.3 Beyond one-bit functions

Weighted bipartite independent sets

To explain how we prove Theorem 1.5, consider a bipartite graph $G = (V_L, V_R, E)$ and let $\lambda_\ell = 0$ (the case $\lambda_r = 0$ is symmetric). We observe that every independent set I which contributes a non-zero summand to $Z_{\lambda_\ell,\lambda_r}(G)$ can only contain vertices in V_R ($Z_{\lambda_\ell,\lambda_r}(G)$ is defined in Problem 1.4). This yields the closed form $Z_{\lambda_\ell,\lambda_r}(G) = (\lambda_r + 1)^{|V_R|}$, which

is computable in polynomial time. Regarding the case $\lambda_\ell, \lambda_r \not\equiv 0 \pmod p$, we employ a generalisation of a reduction used by Faben. In [6, Theorem 3.7] Faben reduces $\#_p\text{SAT}$ to $\#_p\text{BIS}_{1,1}$, the problem of counting independent sets of a bipartite graph.

We have to generalise this reduction for the weighted setting, in particular allowing different vertex weights for the vertices of each partition. Furthermore, during the construction we have to keep track of the assignment of vertices to their corresponding part, V_L or V_R. For this purpose we need to show the existence of bipartite graphs B, where $Z_{\lambda_\ell,\lambda_r}(B)$ takes specific values. These graphs are then used as gadgets in our reduction. In the unweighted setting $\#_p\text{BIS}_{1,1}$ the graphs B are complete bipartite graphs. However, in the weighted setting $\#_p\text{BIS}_{\lambda_\ell,\lambda_r}$ complete bipartite graphs are not sufficient. Therefore, we prove the existence of the necessary bipartite gadgets B constructively. Key results appear in Section 2 and the technical proofs appear in Section 3 of the full version.

Pinning

Similar to the existing hardness proofs on modular counting graph homomorphisms we deploy a "pinning" technique. A partial function from a set X to a set Y is a function $f : X' \to Y$ for some $X' \subseteq X$. For any graph H, a *partially H-labelled graph* $J = (G, \tau)$ consists of an *underlying graph* G and a *pinning function* τ, which is a partial function from $V(G)$ to $V(H)$. A homomorphism from a partially labelled graph $J = (G, \tau)$ to H is a homomorphism $\sigma\colon G \to H$ such that, for all vertices $v \in \text{dom}(\tau)$, $\sigma(v) = \tau(v)$. The resulting problem is denoted by $\#_p\text{PartLabHomsTo}H$, that is, given a prime p and graph H, compute $|\text{Hom}(J \to H)| \pmod p$. In Section 3, we show that $\#_p\text{PartLabHomsTo}H$ reduces to $\#_p\text{HomsTo}H$. This allows us to establish hardness for $\#_p\text{HomsTo}H$, by proving hardness for $\#_p\text{PartLabHomsTo}H$. The reduction generalises the pinning reduction of Göbel, Goldberg and Richerby [10] from $\#_2\text{PartLabHomsTo}H$ to $\#_2\text{HomsTo}H$.

We explain how to prove pinning when we restrict the value of the modulo to 2 and the pinning function $\tau(J) = \{u \mapsto v\}$ to "pin" a single vertex. Given two graphs with distinguished vertices (G, u) and (H, v), let $\text{Hom}((G, u) \to (H, v))$ be the set of homomorphisms from G to H mapping u to v. Given a graph with a distinguished vertex (G, u) and a graph H, we define $\mathbf{w}_H(G)$ to be the $\{0, 1\}$-vector containing the entries $|\text{Hom}((G, u) \to (H, v))| \pmod 2$ for each vertex $v \in V(H)$. Observe that for two vertices $v_1, v_2 \in V(H)$, such that $(H, v_1) \cong (H, v_2)$, and any graph G the relevant entries in $\mathbf{w}_G(H)$ will always be equal. Therefore, we can contract all such entries to obtain the *orbit vectors* $\mathbf{v}_H(G)$. Suppose that there exists a graph with a distinguished vertex (Θ, u_Θ), such that $\mathbf{v}_H(\Theta) = 0 \ldots 010 \ldots 0$, where the 1-entry corresponds to the vertex v of H. Given our input J for $\#_2\text{PartLabHomsTo}H$ we can now define an input G for $\#_2\text{HomsTo}H$, such that $|\text{Hom}(J \to H)| \equiv |\text{Hom}((G(J), u) \to (H, v))| \equiv |\text{Hom}(G \to H)| \pmod 2$. G contains a disjoint copy of $G(J)$ and Θ, where the vertices u and u_Θ are identified (recall that u is the vertex of J mapped by $\tau(J)$). Due to the value of $\mathbf{v}_H(\Theta)$ and the structure of G there is an even number of homomorphisms mapping u to any vertex $v' \neq v$, which establishes the claim.

Such a graph Θ, however, is not guaranteed to exist. Instead, we can define a set of operations on the vectors $\mathbf{v}_H$ corresponding to graph operations and show that for any vector in $\{0, 1\}^{|V(H)|}$ there exist a sequence of graphs with distinguished vertices $(\Theta_1, u_1), \ldots, (\Theta_t, u_t)$ that "generate" this vector. Thus, there exists a set of graphs that "generate" $\mathbf{v} = 0 \ldots 010 \ldots 0$, which yields the desired reduction. This technique of [10] exploits the value of the modulo to be 2. Applying this technique to counting modulo any prime p directly, one can establish pinning for asymmetric graphs, that is graphs whose automorphism group contains only the identity. A dichotomy for $\#_p\text{HomsTo}H$, when H is an asymmetric tree appears in the first author's doctoral thesis [8].

In order to go beyond asymmetric graphs, one has to observe that information redundant only in the modulo 2 case is lost from the contraction of the vectors $\mathbf{w}_H$ to the vectors $\mathbf{v}_H$. This works on asymmetric graphs, since then these two vectors are identical. For general graphs we are able to restore pinning for counting homomorphisms modulo any prime p by utilising the non-contracted vectors $\mathbf{w}_H$.

▶ **Theorem 1.7.** *Let p be a prime and let H be a graph. Then $\#_p$PartLabHomsToH reduces to $\#_p$HomsToH via polynomial time Turing reduction.*

To obtain hardness for $\#_p$HomsToH we only need to pin two vertices when H is a tree, i.e. the domain of the pinning function τ has size two. For a study of a more general class of target graphs H (see [10]), the size of the domain has to be larger. As our pinning theorem applies to all primes p, all graphs H and pinning functions of arbitrary domain size, it can potentially be used to show hardness for $\#_p$HomsToH for all primes and any class of target graphs H. The key lemmas are presented in Section 3 and the formal proofs in Section 5 of the full version.

Gadgets

Gadgets are structures appearing in the target graph H that allow to reduce $\#_2$IS to $\#_2$PartLabHomsToH (the hardness of $\#_2$HomsToH is then immediate from Theorem 1.7). For illustrative purposes we simplify the definitions appearing in [10]. $\#_2$HomsToH–gadgets consist of two partially labelled graphs with distinguished vertices (J_1, y), (J_2, y, z) along with two "special" vertices $i, o \in V(H)$. Given the input G for $\#_2$IS, we construct an input G' for $\#_2$PartLabHomsToH as follows. We attach a copy of J_1 to every vertex u of G (identifying u with y) and replace every edge (u, v) of G with a copy of J_2 (identifying u with y and v with z). The properties of J_1 ensure that there is an odd number of homomorphisms from G' to H where the original vertices of G are mapped to i or o, while the number of the remaining homomorphisms cancels out. The properties of J_2 ensure that there is an even number of homomorphisms from G' to H when two adjacent vertices of G are both mapped to i, and an odd number of homomorphisms in every other case. We can now observe that $|\mathcal{I}(G)| \equiv |\mathrm{Hom}\,(G' \to H)| \pmod 2$, as the set of homomorphisms that does not cancel out must map every vertex of G to i or o and no pair of adjacent vertices both to i. Every vertex of G that is in an independent set must be mapped to i, and every vertex that is out of the independent set must be mapped to o.

Generalising the described approach to modulo any prime $p > 2$ one would end up reducing from a restricted $\#_p$CSP instance, containing a binary relation and a unary weight that must be applied to every variable of the instance (this is known as *external field* in statistical physics). Similar to the modulo 2 case the edge interaction is captured by the binary relation and size of the set of "special" vertices by the unary weights. Since for primes $p > 2$ there are more non-zero values than 1 (odd) a study of the external field is no longer trivial in this case. Instead we choose a different approach and reduce from $\#_p\mathrm{BIS}_{\lambda_\ell,\lambda_r}$. This seems to capture the structure that produces hardness in $\#_p$HomsToH in a more natural way.

We formally present our reduction in Section 4. In the following we sketch our proof method and focus our attention on the example graph H in Figure 1. Let $G = (V_L, V_R, E)$ be a bipartite graph. Homomorphisms from G to H must respect the partition of G, i.e. the vertices in V_L can only be mapped to the vertices in $\{x_L, u_1, u_2, u_3\}$ and the vertices in V_R can only be mapped to the vertices in $\{x_R, v_1, v_2\}$, or vice versa. Any homomorphism σ from G to H, which maps the vertex $w \in V(G)$ to any vertex in $\{u_1, u_2, u_3\}$, must map

every neighbour of w to x_R. Similarly, any homomorphism σ from G to H, which maps the vertex $w \in V(G)$ to any vertex in $\{v_1, v_2\}$, must map every neighbour of w to x_L. Thus, homomorphisms from G to H express independent sets of G: $\{u_1, u_2, u_3\}$ represent the vertices of V_L in the independent set and $\{v_1, v_2\}$ represent the vertices of V_R in the independent set, or vice versa. We construct a partially labelled graph J from G to fix the choice of V_L and V_R in the set of homomorphisms from G to H. $G(J)$ contains a copy of G, where every vertex in V_L is attached to the new vertex $\hat{u}$ and every vertex in V_R is attached to the new vertex $\hat{v}$. In addition, $\tau(J) = \{\hat{u} \mapsto x_R, \hat{v} \mapsto x_L\}$ is the pinning function. We observe that the vertices in V_L can only be mapped to vertices in $\{x_L, u_1, u_2, u_3\}$ and vertices in V_R can only be mapped to vertices in $\{x_R, v_1, v_2\}$. This observation yields that the number of homomorphisms from J to H is equivalent to $\sum_{I \in \mathcal{I}(G)} 3^{|V_L \cap I|} 2^{|V_R \cap I|} \pmod p$. Furthermore, the cardinality of the sets $\{u_1, u_2, u_3\}$ and $\{v_1, v_2\}$ introduces weights in a natural way.

For the reduction above we need the following property easily observable in H: there exist two adjacent vertices of degree $a = \lambda_\ell + 1 \not\equiv 1 \pmod p$ and $b = \lambda_r + 1 \not\equiv 1 \pmod p$. Recall that in order to obtain hardness for $\#_p\text{BIS}_{\lambda_\ell, \lambda_r}$ Theorem 1.5 requires $\lambda_\ell, \lambda_r \not\equiv 0 \pmod p$. In fact, as we will show in Section 4, these vertices need not be adjacent. During the construction of J we can replace the edges of G with paths of appropriate length. We call such a structure in H an (a, b, p)-path. In Lemma 4.4 we formally prove that if H has an (a, b, p)-path, then $\#_p\text{HomsTo}H$ is $\#_p\,\mathsf{P}$-hard. In particular, observe that stars cannot contain (a, b, p)-paths. Finally, we show that every non-star tree H contains an (a, b, p)-path, which yields our main result on $\#_p\text{HomsTo}H$ (Lemma 4.2).

1.4 Composites

We outline the obstacles occurring when extending the dichotomy for $\#_k\text{HomsTo}H$ to any integer k. Let H be a graph and let $k = \prod_{i=1}^{m} k_i$, where $k_i = p_i^{r_i}$ is an integer with its prime factorisation. Assuming $\#_k\text{HomsTo}H$ can be solved in polynomial time, then for each $i \in [m]$, $\#_{k_i}\text{HomsTo}H$ can also be solved in polynomial time. The reason is that k_i is a factor of k and we can apply the modulo k_i operator to the answer for the $\#_k\text{HomsTo}H$ instance. The Chinese remainder theorem shows that the converse is also true: if for each $i \in [m]$ we can solve $\#_{k_i}\text{HomsTo}H$ in polynomial time, then we can also solve $\#_k\text{HomsTo}H$ in polynomial time. By the previous observations we can now focus on powers of primes $k = p^r$. Assuming $\#_k\text{HomsTo}H$ is computable in polynomial time yields again that $\#_p\text{HomsTo}H$ is also computable in polynomial time. However, the converse is not always true.

Guo et al. [13] were able to obtain this reverse implication for the constraint satisfaction problem. They showed [13, Lemma 4.1 and Lemma 4.3] that when p is a prime $\#_{p^r}\text{CSP}$ is computable in polynomial time if $\#_p\text{CSP}$ is computable in polynomial time. In the full version we show that their technique cannot be transferred to the $\#_k\text{HomsTo}H$ setting. We show that there is a graph (P_4) such that $\#_2\text{HomsTo}P_4$ is computable in polynomial time, while $\#_4\text{HomsTo}P_4$ is $\#_2\,\mathsf{P}$-hard.

2 Weighted bipartite independent set

We study the complexity of Problem 1.4, the problem of computing the weighted sum over independent sets in a bipartite graph. We begin by identifying the tractable instances.

▶ **Proposition 2.1.** *If* $\lambda_\ell \equiv 0 \pmod p$ *or* $\lambda_r \equiv 0 \pmod p$ *then* $\#_p\mathrm{BIS}_{\lambda_\ell,\lambda_r}$ *is computable in polynomial time.*

To prove the hardness of the remaining cases, we reduce from $\#_p\mathrm{SAT}$. Our reduction starts with a Boolean formula φ and constructs, in two stages, a graph G_φ, such that $Z_{\lambda_\ell,\lambda_r}(G_\varphi) \equiv K|\operatorname{sat}(\varphi)| \pmod p$, where $\operatorname{sat}(\varphi)$ denotes the set of satisfying assignments of φ and K is a constant depending on the values of the weights λ_ℓ and λ_r. In the first stage we define the graph G'_φ. For every variable x_i in φ, G'_φ contains three vertices u_i, $\bar{u}_i$ and w_i to the left vertex set $V_L(G'_\varphi)$ as well as three vertices v_i, $\bar{v}_i$ and z_i to the right vertex set $V_R(G'_\varphi)$. For every clause c_j of φ, G'_φ further contains a vertex y_j in the right vertex set $V_R(G'_\varphi)$. We further introduce the edges forming the cycle $u_iv_iw_i\bar{v}_i\bar{u}_iz_iu_i$ to $E(G'_\varphi)$ for every variable x_i in φ. Additionally for all $i \in [n]$, if x_i appears as a literal in clause c_j of φ, we introduce the edge (u_i, y_j) in G'_φ and if $\bar{x}_i$ appears as a literal in clause c_j, we introduce the edge $(\bar{u}_i, y_j)$ in G'_φ. The left part of Figure 2 illustrates an example of this construction.

The second stage uses copies of a bipartite graph B, which is obtained by the following key lemma.

▶ **Lemma 2.2.** *Let* p *be a prime and* $\lambda_\ell, \lambda_r \in \mathbb{Z}_p^*$. *There exists a bipartite graph* $B = (V_L, V_R, E)$ *with distinguished vertices* $u_L \in V_L$ *and* $v_R \in V_R$, *that satisfies*

1. $Z_{\lambda_\ell,\lambda_r}(B) \equiv 0 \pmod p$,
2. $Z_{\lambda_\ell,\lambda_r}(B - u_L) \not\equiv 0 \pmod p$,
3. $Z_{\lambda_\ell,\lambda_r}(B - v_R) \not\equiv 0 \pmod p$.

In the second and final stage, we construct the graph G_φ. Let (B, u_L, v_R) be the graph obtained from Lemma 2.2. G_φ is a copy of G'_φ together with two copies of B for every variable of φ and one copy of B for every clause. The first n copies $B^1, \dots, B^n$ are connected to G'_φ by identifying the distinguished vertex u_L^i in the left component with $w_i \in V_L(G'_\varphi)$ for all $i \in [n]$. The second n copies $B^{n+1}, \dots, B^{2n}$ are connected to G'_φ by identifying the distinguished vertex v_R^{n+i} in their right components with $z_i \in V_R(G'_\varphi)$ for all $i \in [n]$. The remaining m copies $B^{2n+1}, \dots, B^{2n+m}$ of B are connected to G'_φ by identifying the distinguished vertex v_R^{2n+j} in their right components with $y_j \in V_R(G'_\varphi)$ for all $j \in [m]$. For an example see the right part of Figure 2.

From this construction we obtain the desired result.

▶ **Theorem 1.5.** *Let* p *be a prime and let* λ_ℓ, $\lambda_r \in \mathbb{Z}_p$. *If* $\lambda_\ell \equiv 0 \pmod p$ *or* $\lambda_r \equiv 0 \pmod p$ *then* $\#_p\mathrm{BIS}_{\lambda_\ell,\lambda_r}$ *is computable in polynomial time. Otherwise,* $\#_p\mathrm{BIS}_{\lambda_\ell,\lambda_r}$ *is* $\#_p\,\mathsf{P}$*-complete.*

3 Homomorphisms of partially labelled graphs

We study the following problem.

▶ **Problem 3.1.** $\#_p$PartLabHomsToH.

Parameter.	Graph H and prime p.
Input.	Partially H-labelled graph $J = (G, \tau)$.
Output.	$\lvert\operatorname{Hom}(J \to H)\rvert \pmod p$.

We observe the following theorem for all primes p.

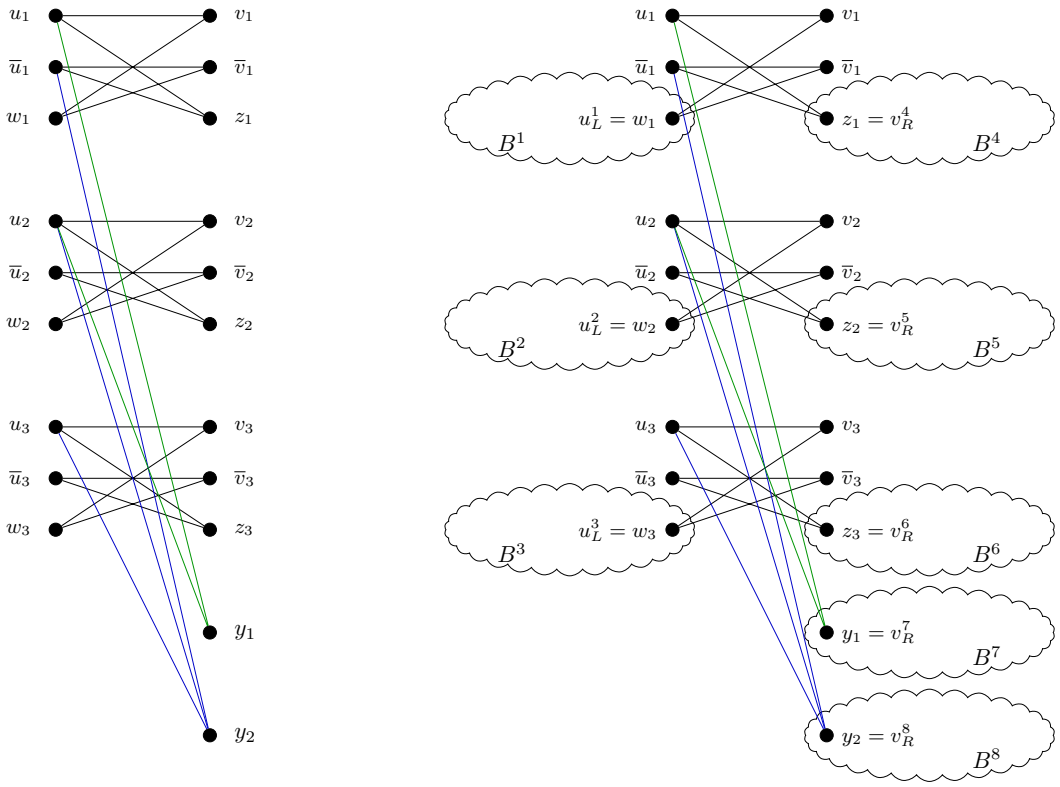


Figure 2 The graphs G'_φ and G_φ for $\varphi = (x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$.

▶ **Lemma 3.2** (Göbel, Goldberg and Richerby [10]). *Let p be a prime and let $(H, \bar{v})$ and $(H', \bar{v}')$ be graphs having no automorphism of order p, each with r distinguished vertices. Then $(H, \bar{v}) \cong (H', \bar{v}')$ if and only if, for all (not necessarily connected) graphs $(G, \bar{u})$ with r distinguished vertices,*

$$|\operatorname{Hom}((G, \bar{u}) \to (H, \bar{v}))| \equiv |\operatorname{Hom}((G, \bar{u}) \to (H', \bar{v}'))| \pmod{p}.$$

Instead of orbit vectors, which are used in the pinning proof of [10], we employ tuple vectors. The tuple vectors include the sizes of the orbits $\operatorname{Orb}_H(\bar{v})$, for all $v \in V(H)^r$, and this information is vital for the proof of our pinning theorem.

▶ **Definition 3.3.** Let H be a graph with no automorphism of order p, $r \in \mathbb{Z}_{>0}$ and let $\bar{w}_1, \ldots, \bar{w}_\nu$ be an enumeration of $(V(H))^r$, i.e., $\nu = |V(H)|^r$. Let $(G, \bar{u})$ be a graph with r distinguished vertices. We define the *tuple vector* $\mathbf{w}_H(G, \bar{u}) \in (\mathbb{Z}_p)^\nu$ where, for each $j \in [\nu]$, the j-th component of $\mathbf{w}_H(G, \bar{u})$ is given by

$$\left(\mathbf{w}_H(G, \bar{u})\right)_j \equiv |\operatorname{Hom}((G, \bar{u}) \to (H, \bar{w}_j))| \pmod{p}.$$

We say that $(G, \bar{u})$ *implements* this vector.

Not all tuple vectors in $(\mathbb{Z}_p)^\nu$ are implementable. We only require the following set to be implementable.

▶ **Definition 3.4.** Let H be a graph with no automorphism of order p, $r \in \mathbb{Z}_{>0}$ and let $\bar{w}_1, \ldots, \bar{w}_\nu$ be an enumeration of $(V(H))^r$, i.e., $\nu = |V(H)|^r$. Denote by $F(H, r) \subseteq (\mathbb{Z}_p)^\nu$ the set of vectors $\mathbf{w}$, such that, for all $i, j \in [\nu]$ with $(H, \bar{w}_i) \cong (H, \bar{w}_j)$, we have $(\mathbf{w})_i = (\mathbf{w})_j$.

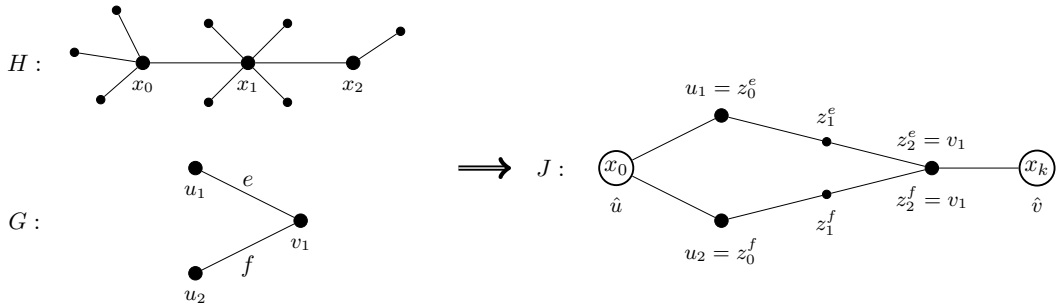


Figure 3 An instance for $p = 5$ of $\#_p\text{BIS}_{3,1}$ reducing to $\#_p\text{PartLabHomsTo}H$. H contains the $(4, 2, 5)$-path $x_0x_1x_2$. G is transformed to the partially labelled graph J, where the mappings of $\tau(J)$ are shown as vertices encircling the target of the mapping.

▶ **Lemma 3.5.** *Let H be a graph with no automorphism of order p, $r \in \mathbb{Z}_{>0}$ and $\bar{w}_1, \ldots, \bar{w}_\nu$ an enumeration of $(V(H))^r$, i.e., $\nu = |V(H)|^r$. Then every $\mathbf{w} \in F(H, r)$ is H-implementable.*

Using the above in the full version we prove the following theorem.

▶ **Theorem 1.7.** *Let p be a prime and let H be a graph. Then $\#_p\text{PartLabHomsTo}H$ reduces to $\#_p\text{HomsTo}H$ via polynomial-time Turing reduction.*

4 Hardness for trees

The structure in H that yields hardness for $\#_p\text{HomsTo}H$ is formally defined as follows.

▶ **Definition 4.1.** Let H be a graph, p be a prime and $a, b \in \mathbb{Z}_p \setminus \{1\}$. Assume H contains a path $P = x_0 \ldots x_k$ for $k > 0$, such that the following hold

1. P is the unique path between x_0 and x_k in H.
2. $\deg_H(x_0) \equiv a \pmod p$ and $\deg_H(x_k) \equiv b \pmod p$.
3. For all $0 < i < k$, $\deg_H(x_i) \equiv 1 \pmod p$.

Then, we will call P an *(a, b, p)-path in H* and denote it Q_H.

▶ **Lemma 4.2.** *Let H be a tree that has no automorphism of order p. Then, either H is a star or there are $a, b \in \mathbb{Z}_p \setminus \{1\}$ such that H contains an (a, b, p)-path.*

In order to show that $\#_p\text{HomsTo}H$ is $\#_p$ P-hard we are going to establish a reduction from $\#_p\text{BIS}_{\lambda_\ell,\lambda_r}$ to $\#_p\text{PartLabHomsTo}H$. Let p be a prime and let H be a tree, target graph in $\#_p\text{PartLabHomsTo}H$. Given a graph $G = (V_L, V_R, E)$ input for $\#_p\text{BIS}_{\lambda_\ell,\lambda_r}$, we construct a partially labelled graph J, input for $\#_p\text{PartLabHomsTo}H$, such that $Z_{\lambda_\ell,\lambda_r}(G) \equiv |\operatorname{Hom}(J \to H)| \pmod p$. Assume H contains an (a, b, p)-path $Q = x_0 \ldots x_k$ and let $P_k = z_0 \ldots z_k$ be the k-path of length k. For every edge $e \in E$, we take a copy of P_k denoted $P_k^e = z_0^e \ldots z_k^e$. Then, J is constructed starting with G by adding two vertices $\hat{u}$ and $\hat{v}$ and connecting them to every vertex in V_L and V_R, respectively. Subsequently, every edge $e \in E$ is substituted with a the path P_k^e. Finally, the pinning function of J maps $\hat{u}$ to x_0 as well as $\hat{v}$ to x_k. See Figure 3 for an example.

The following lemma gives the key properties of J, which establish the reduction.

▶ **Lemma 4.3.** *Let p be a prime, $G = (V_L, V_R, E)$ a bipartite graph and H be a tree. Assume there are $a, b \in \mathbb{Z}_p \setminus \{1\}$ such that H contains an (a, b, p)-path $Q_H = x_0 \ldots x_k$. We denote the diminished neighbourhoods of x_0 and x_k by $W_L = \Gamma_H(x_0) - x_1$ and $W_R = \Gamma_H(x_k) - x_{k-1}$,*

respectively. Additionally, let J be the partially labelled graph described above. Then, for every homomorphism σ from J to H the following hold.

1. *Let $u \in V_L$ and $v \in V_R$, then $\sigma(u) \in \Gamma_H(x_0)$ and $\sigma(v) \in \Gamma_H(x_k)$, respectively;*
2. *Let $\mathfrak{O}_\sigma = \{u \in V_L \mid \sigma(u) = x_1\} \cup \{v \in V_R \mid \sigma(v) = x_{k-1}\}$ and $\mathfrak{I}_\sigma = (V_L \cup V_R) \setminus \mathfrak{O}_\sigma$. Given another homomorphism σ' from J to H, the relation $\sigma \sim_\mathfrak{I} \sigma'$ if $\mathfrak{I}_\sigma = \mathfrak{I}_{\sigma'}$ is an equivalence relation with equivalence class denoted $[\![\cdot]\!]_\mathfrak{I}$;*
3. *Let $\sigma_1, \ldots, \sigma_\mu$ be representatives from each $\sim_\mathfrak{I}$-equivalence class. Then, the set $\mathcal{I}(G)$ of independent sets of G is exactly the set $\{\, \mathfrak{I}_{\sigma_i} \mid i \in [\mu] \,\}$.*
4. *For the diminished neighbourhoods holds $|[\![\sigma]\!]_\mathfrak{I}| \equiv |W_L|^{|\mathfrak{I}_\sigma \cap V_L|} |W_R|^{|\mathfrak{I}_\sigma \cap V_R|} \pmod p$.*

▶ **Lemma 4.4.** *Let p be a prime and let H be a graph with no automorphism of order p. If there are $a, b \in \mathbb{Z}_p \setminus \{1\}$ such that H has an (a, b, p)-path Q_H then $\#_p\textsc{HomsTo}H$ is $\#_p\,\mathsf{P}$-hard under Turing reductions.*

5 Dichotomy theorems

The results of Faben and Jerrum [7] combined with Lemma 4.4 give the following dichotomy theorem.

▶ **Theorem 1.2.** *Let p be a prime and let H be a graph, such that its order p reduced form H^{*p} is a tree. If H^{*p} is a star, then $\#_p\textsc{HomsTo}H$ is computable in polynomial time; otherwise, $\#_p\textsc{HomsTo}H$ is $\#_p\,\mathsf{P}$-complete.*

To justify our title, we use the following proposition showing that our dichotomy theorem holds for all trees. In [7, Section 5.3] this was stated as an obvious fact, however for the sake of completeness we provide a formal proof.

▶ **Proposition 5.1.** *Let H be a tree and ϱ an automorphism of H. Then the subgraph H^ϱ of H induced by the fixed points of ϱ is also a tree.*

The claim implies that if H is a tree, then its order p reduced form H^{*p} is also a tree. This yields the following corollary.

▶ **Corollary 1.3.** *Let p be a prime and let H be a tree. If the order p reduced form H^{*p} of H is a star, then $\#_p\textsc{HomsTo}H$ is computable in polynomial time; otherwise, $\#_p\textsc{HomsTo}H$ is $\#_p\,\mathsf{P}$-complete.*

To deal with disconnected graphs, Faben and Jerrum [7, Theorem 6.1] show the following theorem.

▶ **Theorem 5.2** (Faben and Jerrum)**.** *Let H be a graph that has no automorphism of order 2. If H' is a connected component of H and $\#_2\textsc{HomsTo}H'$ is $\#_2\,\mathsf{P}$-hard, then $\#_2\textsc{HomsTo}H$ is $\#_2\,\mathsf{P}$-hard.*

The only part where the value 2 of the modulo is required, is the application of their pinning theorem [7, Theorem 4.7]. Since we have already shown the more general Theorem 1.7, we conclude that the theorem holds in the following form.

▶ **Theorem 5.3.** *Let p be a prime and let H be a graph that has no automorphism of order p. If H_1 is a connected component of H and $\#_p\textsc{HomsTo}H_1$ $\#_p\,\mathsf{P}$-hard, then $\#_p\textsc{HomsTo}H$ is $\#_p\,\mathsf{P}$-hard.*

The latter strengthens Theorem 1.2 to the following version.

▶ **Theorem 5.4.** *Let H be a graph whose order p reduced form H^{*p} is a forest. If every component of H^{*p} is a star, $\#_p\text{HOMSTO}H$ is computable in polynomial time, otherwise $\#_p\text{HOMSTO}H$ is $\#_p\,\mathsf{P}$-complete.*

A discussion of our results was already conducted in the introduction. Again we refer the curious reader to the full version of the paper, available at `https://arxiv.org/abs/1802.06103`.

References

1. A. A. Bulatov and M. Grohe. The complexity of partition functions. *Theoretical Computer Science*, 348(2-3):148–186, 2005. `doi:10.1016/j.tcs.2005.09.011`.
2. J.-Y. Cai, X. Chen, and P. Lu. Graph homomorphisms with complex values: A dichotomy theorem. *SIAM Journal on Computing*, 42(3):924–1029, 2013. `doi:10.1137/110840194`.
3. J.-Y. Cai and P. Lu. Holographic algorithms: From art to science. *Journal of Computer and System Sciences*, 77(1):41–61, 2011.
4. M. E. Dyer, A. M. Frieze, and M. Jerrum. On counting independent sets in sparse graphs. *SIAM Journal on Computing*, 31(5):1527–1541, 2002. `doi:10.1137/S0097539701383844`.
5. M. E. Dyer and C. S. Greenhill. The complexity of counting graph homomorphisms. *Random Structures and Algorithms*, 17(3-4):260–289, 2000.
6. J. Faben. The complexity of counting solutions to generalised satisfiability problems modulo k. *arXiv*, abs/0809.1836, 2008.
7. J. Faben and M. Jerrum. The complexity of parity graph homomorphism: an initial investigation. *Theory of Computing*, 11:35–57, 2015.
8. A. Göbel (A. Gkompel-Magkakis). *Counting, Modular Counting and Graph Homomorphisms*. PhD thesis, University of Oxford, 2016.
9. A. Göbel, L. A. Goldberg, and D. Richerby. The complexity of counting homomorphisms to cactus graphs modulo 2. *ACM Transactions on Computation Theory*, 6(4):17:1–17:29, 2014.
10. A. Göbel, L. A. Goldberg, and D. Richerby. Counting homomorphisms to square-free graphs, modulo 2. *ACM Transactions on Computation Theory,*, 8(3):12:1–12:29, 2016.
11. L. A. Goldberg, M. Grohe, M. Jerrum, and M. Thurley. A complexity dichotomy for partition functions with mixed signs. *SIAM Journal on Computing*, 39(7):3336–3402, 2010.
12. L. M. Goldschlager and I. Parberry. On the construction of parallel computers from various bases of Boolean functions. *Theoretical Computer Science*, 43:43–58, 1986.
13. H. Guo, S. Huang, P. Lu, and M. Xia. The complexity of weighted boolean #CSP modulo k. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 249–260, 2011.
14. P. Hell and J. Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48(1):92–110, 1990.
15. R. E. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22(1):155–171, 1975.
16. C. H. Papadimitriou and S. Zachos. Two remarks on the power of counting. In *Proceedings of the GI-Conference on Theoretical Computer Science*, pages 269–276, 1982.
17. S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.
18. L. G. Valiant. Accidental algorthims. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 509–517, 2006.

Car-Sharing between Two Locations: Online Scheduling with Two Servers

Kelin Luo[1]
School of Management, Xi'an Jiaotong University
Xianning West Road, Xi'an, China
luokelin@xjtu.edu.cn
https://orcid.org/0000-0003-2006-0601

Thomas Erlebach
Department of Informatics, University of Leicester
Leicester, United Kingdom
te17@leicester.ac.uk
https://orcid.org/0000-0002-4470-5868

Yinfeng Xu
School of Management, Xi'an Jiaotong University
Xianning West Road, Xi'an, China
yfxu@xjtu.edu.cn

Abstract

In this paper, we consider an on-line scheduling problem that is motivated by applications such as car sharing, in which users submit ride requests, and the scheduler aims to accept requests of maximum total profit using two servers (cars). Each ride request specifies the pick-up time and the pick-up location (among two locations, with the other location being the destination). The length of the time interval between the submission of a request (booking time) and the pick-up time is fixed. The scheduler has to decide whether or not to accept a request immediately at the time when the request is submitted. We present lower bounds on the competitive ratio for this problem and propose a smart greedy algorithm that achieves the best possible competitive ratio.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Car-sharing system, Competitive analysis, On-line scheduling

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.50

1 Introduction

In a car-sharing system, a company offers cars to customers for a period of time. Customers can pick up a car in one location, drive it to another location, and return it there. Car booking requests arrive on-line, and the goal is to maximize the profit obtained from satisfied requests. We consider a setting where all driving routes go between two fixed locations, but can be in either direction. For example, the two locations could be a residential area and a nearby shopping mall or central business district. Other applications that provide motivation for the problems we study include car rental, taxi dispatching and boat rental for river crossings.

[1] This work was partially supported by the China Postdoctoral Science Foundation (Grant No. 2016M592811), and the China Scholarship Council (Grant No. 201706280058).

In a real setting, customer requests for car bookings arrive over time, and the decision about each request must be made immediately, without knowledge of future requests. This gives rise to an on-line problem that bears some resemblance to interval scheduling, but in which additionally the pick-up and drop-off locations play an important role: The server that serves a request must be at the pick-up location at the start time of the request and will be located at the drop-off location at the end time of the request. A server can serve two consecutive requests only if the drop-off location of the first request is the same as the pick-up location of the second request, or if there is enough time to travel between the two locations otherwise. We allow 'empty movements' that allow a server to be moved from one location to another while not serving a request. Such empty movements could be implemented by having company staff drive a car from one location to another, or in the future by self-driving cars.

We assume that every request is associated with a profit $r > 0$ that is obtained if the request is accepted. When a server moves while not serving a request, a certain cost c, $0 \leq c \leq r$, is incurred. The goal is to maximize the total profit, which is the sum of the profits of the accepted requests minus the costs incurred for moving servers while not serving a request. We refer to this problem as the *car-sharing problem*. The time interval between the submission of a request (booking time) and the pick-up time is called the *booking interval*. In this paper, we focus on the special case of two servers and assume that the booking interval for each request is a fixed value a that is the same for all requests. We assume that $a \geq t$, where t is the time to move a server from one location to the other.

In [8], the authors studied the car-sharing problem for the special case of a single server, considering both the case of fixed booking intervals and the case of flexible booking intervals, and presented tight results for the competitive ratio. The optimal competitive ratio was shown to be $2r/(r-c)$ for fixed booking intervals and $(3r-c)/(r-c)$ for flexible booking intervals if $0 \leq c < r$, and 1 for fixed booking intervals and proportional to the length of the booking horizon (the range of allowed booking intervals) for flexible booking intervals if $c = r$.

The car-sharing problem belongs to the class of dynamic pickup and delivery problems surveyed by Berbeglia et al. [2]. The problem that is closest to our setting is the on-line dial-a-ride problem (OLDARP) that has been widely studied in the literature. In OLDARP, transportation requests between locations in a metric space arrive over time, but typically it is assumed that requests want to be served 'as soon as possible' rather than at a specific time as in our problem. Known results for OLDARP include on-line algorithms for minimizing the makespan [1, 3] or the maximum flow time [7]. Work on versions of OLDARP where not all requests can be served includes competitive algorithms for requests with deadlines where each request must be served before its deadline or rejected [9], and for settings with a given time limit where the goal is to maximize the revenue from requests served before the time limit [6]. In contrast to existing work on OLDARP, in this paper we consider requests that need to be served at a specific time that is specified by the request when it is released. Another related problem is the k-server problem [5, Ch. 10], but in that problem all requests must be served and requests are served at a specific location.

Off-line versions of car-sharing problems are studied by Böhmová et al. [4]. They show that if all customer requests for car bookings are known in advance, the problem of maximizing the number of accepted requests can be solved in polynomial time using a minimum-cost network flow algorithm. Furthermore, they consider the problem variant with two locations where each customer requests two rides (in opposite directions) and the scheduler must accept either both or neither of the two. They prove that this variant is NP-hard and APX-hard. In contrast to their work, we consider the on-line version of the problem with two servers.

In Section 2, we define the problem, introduce terminology, and present lower bounds on the competitive ratio. If $0 \leq c < r$, the lower bound is 2, and if $c \geq r$, the lower bound is 1. In Section 3, we propose a smart greedy algorithm that achieves the best possible competitive ratio. Section 4 concludes the paper.

2 Problem Formulation and Preliminary Results

2.1 Definitions and Problem Formulation

We consider a setting with only two locations (denoted by 0 and 1) and two servers (denoted by s_1 and s_2). The travel time from 0 to 1 is the same as the travel time from 1 to 0 and is denoted by t. Let R denote a sequence of requests that are released over time. The i-th request is denoted by $r_i = (\tilde{t}_{r_i}, t_{r_i}, p_{r_i})$ and is specified by the *booking time* or *release time* $\tilde{t}_{r_i}$, the *start time* (or *pick-up time*) t_{r_i}, and the pick-up location $p_{r_i} \in \{0, 1\}$. We assume that the booking interval $t_{r_i} - \tilde{t}_{r_i}$ is equal to a fixed value a for all requests $r_i \in R$, and we assume that $a \geq t$ so that an available server always has enough time to travel to the pick-location of a request. If r_i is accepted, the server must pick up the customer at p_{r_i} at time t_{r_i} and drop off the customer at location $\dot{p}_{r_i} = 1 - p_{r_i}$, the *drop-off location* of the request, at time $\dot{t}_{r_i} = t_{r_i} + t$, the *end time* (or *drop-off time*) of the request. We say that the request r_i *starts* at time t_{r_i}. For an interval $[b, d)$, we say that r_i starts in the interval if $t_{r_i} \in [b, d)$.

Each server can only serve one request at a time. Serving a request yields profit $r > 0$. The two servers are initially located at location 0. If the pick-up location p_{r_i} of a request r_i is different from the current location of a server and if at least t time units remain before the start time of r_i, the server can move from its current location to p_{r_i}. We refer to such moves (which do not serve a request) as *empty* moves. An empty move takes time t and incurs a cost of c, $0 \leq c \leq r$, and we say that r_i is *accepted with cost* in this case. If the server is already located at p_{r_i}, we say that r_i is *accepted without cost*. If two requests are such that they cannot both be served by one server, we say that the requests are *in conflict*. We do not require that the algorithm assigns an accepted request to a server immediately, provided that it ensures that one of the two servers will serve the request. In our setting with fixed booking intervals, however, it is not necessary for an algorithm to use this flexibility.

We denote the requests accepted by an algorithm by R', and the i-th request in R', in order of request start times, is denoted by r'_i. The l-th request which is assigned to s_j ($j \in \{1, 2\}$) in R', in order of request start times, is denoted by $r'_{l,j}$. Suppose $r'_{l,j}$ ($j \in \{1, 2\}$) is r'_i. We say that request r'_i is accepted *without cost* if $l = 1$ and $p_{r'_{l,j}} = 0$ or if $l > 1$ and $p_{r'_{l,j}} = \dot{p}_{r'_{l-1,j}}$. Otherwise, r'_i is accepted *with cost*. We denote the profit of serving the requests in R' by $P_{R'}$. If R'_c denotes the subset of R' consisting of the requests that are accepted with cost, we have $P_{R'} = r \cdot |R'| - c \cdot |R'_c|$. The goal of the car-sharing problem is to accept a set of requests R' that maximizes the profit $P_{R'}$. The problem for two servers and two locations is called the *2S2L problem*.

2.2 Online Optimization and Competitive Analysis

From an online perspective, the requests in R and the number of requests in R are unknown, and request r_i only becomes known at time $\tilde{t}_{r_i}$. For any request sequence R, let P_{R^A} denote the objective value produced by an on-line algorithm A, and P_{R^*} that obtained by an optimal scheduler OPT that has full information about the request sequence in advance.

The performance of an online algorithm for 2S2L is measured using competitive analysis (see [5]). The competitive ratio of A is defined as $\rho_A = \sup_R \frac{P_{R^*}}{P_{R_A}}$. We say that A is ρ-competitive if $P_{R^*} \leq \rho \cdot P_{R_A}$ for all request sequences R. Let ON be the set of all on-line algorithms for a problem. A value β is a *lower bound* on the best possible competitive ratio if $\rho_A \geq \beta$ for all A in ON. We say that an algorithm A is optimal if there is a lower bound β with $\rho_A = \beta$.

2.3 Lower Bounds

In this subsection, we present the lower bounds for the 2S2L problem. We use ALG to denote any on-line algorithm and OPT to denote an optimal scheduler. We refer to the servers of ALG as s'_1 and s'_2, and the servers of OPT as s^*_1 and s^*_2, respectively. The set of requests accepted by ALG is referred to as R', and the set of requests accepted by OPT as R^*. For the case $c \geq r$, a lower bound of 1 on the competitive ratio of any algorithm holds trivially.

▶ **Theorem 1.** *For $0 \leq c < r$, no deterministic on-line algorithm for 2S2L can achieve competitive ratio smaller than 2.*

Proof. Initially, the adversary releases r_1 and r_2 with $r_1 = r_2 = (t, t + a, 1)$. We distinguish three cases.

Case 1: ALG accepts r_1 and r_2 (with cost). Note that r_1 and r_2 are assigned to different servers as they are in conflict. The adversary releases requests r_3 and r_4 with $r_3 = r_4 = (\varepsilon + t, a + \varepsilon + t, 0)$ and r_5 and r_6 with $r_5 = r_6 = (\varepsilon + 2t, a + \varepsilon + 2t, 1)$, where $0 < \varepsilon < t$. OPT accepts r_3, r_4, r_5 and r_6 without cost, but ALG cannot accept any of these requests as they are in conflict with r_1 and r_2. We have $P_{R^*} = 4r$ and $P_{R'} \leq 2(r - c)$, and hence $P_{R^*}/P_{R'} \geq 2$.

Case 2: ALG accepts either r_1 or r_2. The adversary accepts r_1 and r_2. We have $P_{R^*} = 2(r-c)$ and $P_{R'} \leq r - c$, and hence $P_{R^*}/P_{R'} \geq 2$.

Case 3: ALG does not accept request r_1 and r_2. In this case, OPT accepts r_1 and r_2 and we have $P_{R^*} = 2(r - c)$ and $P_{R'} = 0$, and hence $P_{R^*}/P_{R'} = \infty$. ◀

3 Upper Bound

In this section, we propose a Smart Greedy Algorithm (SG) for the 2S2L problem, shown in Algorithm 1. Intuitively, if a request is acceptable, the algorithm always accepts it if this increases the profit by r, and it accepts the request only if it starts at least t time units later than the end time of the latest previously accepted request if the profit increase is positive but less than r. The algorithm uses the following notation:

- R'_i is the set of requests accepted by SG before r_i is released, together with the server to which each request is assigned. $R'_i \cup \{r_{i,s'_j}\}$ denotes the union of R'_i and $\{r_{i,s'_j}\}$, where r_{i,s'_j} represents the request r_i assigned to server s'_j, $j \in \{1, 2\}$, without conflict.
- $r^n_{i,j}$ denotes the latest request which was assigned to s'_j, $j \in \{1, 2\}$, before r_i is released. (If there is no such request, take $r^n_{i,j}$ to be a dummy request with drop-off location 0 and drop-off time 0.)
- r_i is *acceptable* if and only if $\exists j \in \{1, 2\} : t_{r_i} - \dot{t}_{r^n_{i,j}} \geq t$ if $p_{r_i} = p_{r^n_{i,j}}$, and $t_{r_i} - \dot{t}_{r^n_{i,j}} \geq 0$ if $p_{r_i} \neq p_{r^n_{i,j}}$.
- r^n_i is the latest request that was accepted before r_i is released. Note that $r^n_i = r^n_{i,j}$ with $j = \arg\max\{t_{r^n_{i,j}} \mid j = 1, 2\}$. Note that $\dot{t}_{r^n_1} = 0$.

Algorithm 1 Smart Greedy Algorithm (SG).

Input: two servers, requests arrive over time with fixed booking interval a.
Step: When request r_i arrives, accept r_i and assign it to the most economical server s'_j if r_i is acceptable and $P_{R'_i \cup \{r_{i,s'_j}\}} - P_{R'_i} = r$ $(j \in \{1,2\})$, or if r_i is acceptable, $P_{R'_i \cup \{r_{i,s'_j}\}} - P_{R'_i} > 0$ $(j \in \{1,2\})$ and $t_{r_i} - \dot{t}_{r_i^n} \geq t$;

- If an accepted request is acceptable by both servers, it is assigned to the *most economical* server, which is the server s'_j with $j = \arg\max\{P_{R'_i \cup \{r_{i,s'_j}\}} \mid j = 1,2\}$. If $P_{R'_i \cup \{r_{i,s'_1}\}} = P_{R'_i \cup \{r_{i,s'_2}\}}$, s'_j is chosen as the server which has accepted r_i^n (or arbitrarily in case r_i^n does not exist).

We use OPT to denote an optimal scheduler. We refer to the servers of SG as s'_1 and s'_2, and the servers of OPT as s_1^* and s_2^*, respectively. For an arbitrary request sequence $R = (r_1, r_2, r_3, \ldots, r_n)$, note that we have $t_{r_i} \leq t_{r_{i+1}}$ for $1 \leq i < n$ because $t_{r_i} - \tilde{t}_{r_i} = a$ is fixed. Denote the requests accepted by OPT by $R^* = \{r_1^*, r_2^*, \ldots, r_{k^*}^*\}$ and the requests accepted by SG by $R' = \{r'_1, r'_2, ...r'_k\}$, indexed in order of non-decreasing start times. Denote the requests accepted by SG which start at location 0 by $R'^0 = \{r'^0_1, r'^0_2, ...r'^0_{k_0}\}$ and the requests accepted by SG which start at location 1 by $R'^1 = \{r'^1_1, r'^1_2, ...r'^1_{k_1}\}$. Denote the requests accepted by OPT which start at location 0 by $R^{*0} = \{r_1^{*0}, r_2^{*0}, ...r_{k_0^*}^{*0}\}$ and the requests accepted by OPT which start at location 1 by $R^{*1} = \{r_1^{*1}, r_2^{*1}, ...r_{k_1^*}^{*1}\}$. Note that $R'^0 \bigcup R'^1 = R'$ and $R^{*0} \bigcup R^{*1} = R^*$.

▶ **Theorem 2.** *Algorithm SG is* 1*-competitive for 2S2L if* $c = r$.

Proof. If $c = r$, accepting a request with cost yields profit $r - c = 0$. Without loss of generality, we can therefore assume that both SG and OPT only accept requests without cost. Observe that this means that both the SG servers (s'_1 and s'_2) and the OPT servers (s_1^* and s_2^*) accept requests with alternating pick-up location, starting with a request with pick-up location 0. Therefore each server can accept at most one more request which starts at location 0 over the requests which start at location 1. That means when OPT accepts w requests which start at location 1, OPT at least accepts w requests which start at location 0, and accepts at most $w + 2$ requests which start at location 0 ($k_1^* \leq k_0^* \leq k_1^* + 2$).

Considering the condition that requests r_j^{*0} and r_j^{*1} are both assigned to the same server for $j < i$ and r_i^{*0} and r_i^{*1} are assigned to different servers (without loss of generality, suppose r_i^{*0} is assigned to s_1^* and r_i^{*1} is assigned to s_2^*), we reassign r_i^{*1} to server s_1^*, reassign all requests in $R^* \backslash (\{r_1^{*0}, r_2^{*0}, ..., r_{i+1}^{*0}\} \bigcup \{r_1^{*1}, r_2^{*1}, ..., r_i^{*1}\})$ that are assigned to s_1^* (denote the set of these requests by $\Re_1$) to server s_2^*, and reassign all requests in $R^* \backslash (\{r_1^{*0}, r_2^{*0}, ..., r_{i+1}^{*0}\} \bigcup \{r_1^{*1}, r_2^{*1}, ..., r_i^{*1}\})$ that are assigned to s_2^* (denote them by $\Re_2$) to server s_1^*. As each server accepts requests with alternating pick-up location, starting with a request with pick-up location 0, we have $\dot{t}_{r'^0_i} \leq t_{r'^1_i}$ (for all $i \leq k'_1$) and $\dot{t}_{r_i^{*0}} \leq t_{r_i^{*1}}$ (for all $i \leq k_1^*$). That means for $i \leq k_1^*$, r_i^{*0} and r_i^{*1} are not in conflict, and hence reassigning r_i^{*1} to server s_1^* is valid. Observe that s_2^* must serve a request which has pick-up location 0 and starts during interval $[t_{r_i^{*0}}, t_{r_i^{*1}} - t]$ and that request is r_{i+1}^{*0}. Because $t_{r_{i+1}^{*0}} \leq t_{r_i^{*1}} - t$ and the first request in $\Re_1$, denoted by r_o, has pick-up location 1 and starts after $t_{r_i^{*1}}$, r_o and r_{i+1}^{*0} are not in conflict. As any two consecutive requests in $\Re_1$ are not in conflict, reassigning all requests of $\Re_1$ to server s_2^* is valid. Note that $t_{r_{i+2}^{*0}} \geq \dot{t}_{r_i^{*1}}$ as OPT accepts at most two requests which start during interval $[t_{r_i^{*0}}, t_{r_i^{*1}}]$ (during interval $[0, t_{r_i^{*1}}]$ if $i = 1$) and have pick-up location 0. Because the first request (r_l) in $\Re_2$ starts at 0 and starts after $\dot{t}_{r_i^{*1}}$, r_l and

r_i^{*1} are not in conflict. As any two consecutive requests in $\Re_2$ are not in conflict, reassigning all requests of $\Re_2$ to server s_1^* is valid. From this it follows that R^* is still a valid solution with the same profit after the reassignment. For simplification of the analysis, we reassign the requests in R^* and R' based on the above process until both request r_i^{*0} and r_i^{*1} are assigned to the same server for $i \leq k_1^*$, and $r_i'^0$ and $r_i'^1$ are assigned to the same server for $i \leq k_1'$. Note that this reassignment does not affect the validity of R^* and R', and P_{R^*} and $P_{R'}$ do not change.

We claim that R^* can be transformed into R' without reducing its profit, thus showing that $P_{R^*} = P_{R'}$. As SG accepts the request r_γ which is the first acceptable request that starts at location 0 and the request r_δ which is the first acceptable request that starts at location 1 (r_δ is the first request in R that starts at location 1 and starts after $\dot{t}_{r_\gamma}$), it is clear that $t_{r_1'^0} \leq t_{r_1^{*0}}$ and $t_{r_1'^1} \leq t_{r_1^{*1}}$. If $r_1'^0 \neq r_1^{*0}$, we can replace r_1^{*0} by $r_1'^0$ in R^{*0}, and if $r_1'^1 \neq r_1^{*1}$, we can replace r_1^{*1} by $r_1'^1$ in R^{*1}.

Now assume, that R' and R^* are identical with respect to $2i$ requests (i requests in R^{*0} and i requests R'^0, and i requests in R^{*1} and i requests in R'^1, where $1 \leq i \leq k_1^*$), and both requests r_j^{*0} and r_j^{*1} are assigned to the same server for $1 \leq j \leq i$.

Without loss of generality, suppose $r_i'^1$ is assigned to s_1^* by OPT and $r_i'^1$ is assigned to s_1' by SG. Observe that s_1^* and s_1' are at location 0 at time $\dot{t}_{r_i'^1}$. We claim that s_2^* (resp. s_2') is at location 0 at time $\dot{t}_{r_{i-1}'^1}$ and $\dot{t}_{r_{i-1}'^1} \leq t_{r_{i+1}^{*0}}$. If $r_{i-1}'^1$ is assigned to s_2^* (resp. s_2'), s_2^* (resp. s_2') is at location 0 at time $\dot{t}_{r_{i-1}'^1}$ and $\dot{t}_{r_{i-1}'^1} = \min\{\dot{t}_{r_{i-1}'^1}, \dot{t}_{r_i'^1}\} \leq t_{r_{i+1}^{*0}}$. If $r_{i-1}'^1$ is assigned to s_1^* (resp. s_1'), we have $\dot{t}_{r_{i-1}'^1} \leq t_{r_i'^0} \leq t_{r_{i+1}^{*0}}$. Observe that OPT does not accept any request which starts in period $(t_{r_{i-1}'^1}, \dot{t}_{r_{i-1}'^1})$. As both SG servers, s_1' and s_2', and OPT servers, s_1^* and s_2^*, accept requests with alternating pick-up location and starting with a request with pick-up location 0, either the pick-up location of the request r_o (where r_o is the last request which starts at or before $t_{r_{i-1}'^1}$ and is assigned to s_2^* (resp. s_2')) is 1, or s_2^* (resp. s_2') does not accept any request which starts before $t_{r_{i-1}'^1}$. Hence s_2^* (resp. s_2') is at location 0 at time $\dot{t}_{r_o}$ ($\leq \dot{t}_{r_{i-1}'^1}$), or at time 0 if r_o does not exist, and stays at that location until time $\dot{t}_{r_{i-1}'^1}$.

If there are two requests r_{i+1}^{*0} and r_{i+1}^{*1}, as s_2' is at location 0 at $\dot{t}_{r_{i-1}'^1}$ and $\dot{t}_{r_{i-1}'^1} \leq t_{r_{i+1}^{*0}}$, there must also be two requests $r_{i+1}'^0$ and $r_{i+1}'^1$ with $t_{r_{i+1}'^0} \leq t_{r_{i+1}^{*0}}$ and $t_{r_{i+1}'^1} \leq t_{r_{i+1}^{*1}}$, as SG could accept r_{i+1}^{*0} and r_{i+1}^{*1} by s_2'. We can replace r_{i+1}^{*0} and r_{i+1}^{*1} by $r_{i+1}'^0$ and $r_{i+1}'^1$ in R^{*0} and R^{*1}. If $k_0^* = k_1^*$, the claim thus follows by induction.

If $k_0^* \neq k_1^*$ ($k_0^* - k_1^* = 1$ or $k_0^* - k_1^* = 2$), then R^{*1} is already identical to R'^1, and the first k_1^* requests of R^{*0} are already identical to the first k_1^* requests of R'^0 by the argument above. If $k_0^* - k_1^* = 1$, there is a request $r_{k_1^*+1}^{*0}$. As s_2' is at location 0 at $\dot{t}_{r_{k_1^*-1}'^1}$ and $\dot{t}_{r_{k_1^*-1}'^1} \leq t_{r_{k_1^*+1}^{*0}}$, there must also be one request $r_{k_1^*+1}'^0$ with $t_{r_{k_1^*+1}'^0} \leq t_{r_{k_1^*+1}^{*0}}$, as SG could accept $r_{k_1^*+1}^{*0}$ by s_2'. We can replace $r_{k_1^*+1}^{*0}$ by $r_{k_1^*+1}'^0$ in R^{*0}, making R^{*0} identical to R'^0. If $k_0^* - k_1^* = 2$, there are two requests $r_{k_1^*+1}^{*0}$ and $r_{k_1^*+2}^{*0}$. Note that $r_{k_1^*+1}^{*0}$ and $r_{k_1^*+2}^{*0}$ must be assigned to different servers by OPT as $k_0^* - k_1^* = 2$. Recall that s_1^* is at location 0 at $\dot{t}_{r_{k_1^*}'^1}$, and s_2^* is at location 0 at $\dot{t}_{r_{k_1^*-1}'^1}$. Hence $t_{r_{k_1^*+1}^{*0}} \geq \dot{t}_{r_{k_1^*-1}'^1}$ and $t_{r_{k_1^*+2}^{*0}} \geq \dot{t}_{r_{k_1^*}'^1}$. As s_1' is at location 0 at $\dot{t}_{r_{k_1^*}'^1}$ and s_2' is at location 0 at $\dot{t}_{r_{k_1^*-1}'^1}$, there must also be two requests $r_{k_1^*+1}'^0$ and $r_{k_1^*+2}'^0$ with $t_{r_{k_1^*+1}'^0} \leq t_{r_{k_1^*+1}^{*0}}$ and $t_{r_{k_1^*+2}'^0} \leq t_{r_{k_1^*+2}^{*0}}$, as SG could accept $r_{k_1^*+1}^{*0}$ by s_2', and accept $r_{k_1^*+2}^{*0}$ by s_1'. We can replace $r_{k_1^*+1}^{*0}$ and $r_{k_1^*+2}^{*0}$ by $r_{k_1^*+1}'^0$ and $r_{k_1^*+2}'^0$ in R^{*0}, making R^{*0} identical to R'^0. As R^{*1} is already identical to R'^1, R^* is identical to R' because $R^* = R^{*0} \bigcup R^{*1}$ and $R' = R'^0 \bigcup R'^1$. ◀

▶ **Theorem 3.** *Algorithm SG is 2-competitive for 2S2L if $c = 0$.*

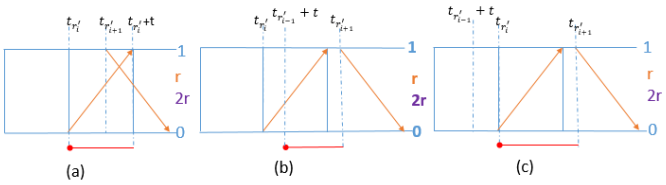


Figure 1 $c = 0$, $|R'| = k > 1$, $1 \le i \le k$.

Proof. We partition the time horizon $[0, \infty)$ into intervals (periods) that can be analyzed independently. Period i, for $1 < i < k$, is the interval $[\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}, \max\{\dot{t}_{r'_i}, t_{r'_{i+1}}\})$. Period 1 is $[0, \max\{\dot{t}_{r'_1}, t_{r'_2}\})$, and period k is $[\max\{\dot{t}_{r'_{k-1}}, t_{r'_k}\}, \infty)$. (If $k = 1$, there is only a single period $[0, \infty)$.) Set $t_{r'_{k+1}} = \infty$ and $\dot{t}_{r'_0} = 0$. Let R^*_i denote the set of requests accepted by OPT that start in period i, for $1 \le i \le k$. For all $1 \le i \le k$, if $\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\} \ge \max\{\dot{t}_{r'_i}, t_{r'_{i+1}}\}$, $R^*_i = \emptyset$, and hence $P_{R^*_i} = 0$. Denote $R'_i = \{r'_i\}$ for $1 \le i \le k$.

For $1 < i \le k$, r'_i starts at time $t_{r'_i}$ and the first request of R^*_i starts during the interval $[\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}, \max\{\dot{t}_{r'_i}, t_{r'_{i+1}}\})$ (or the interval $[\max\{\dot{t}_{r'_{k-1}}, t_{r'_k}\}, \infty)$ if $i = k$). Furthermore, r'_1 is the first acceptable request in R, and so the first request of R^*_1 cannot start before $t_{r'_1}$. Hence, for all $1 \le i \le k$, the first request in R^*_i cannot start before $t_{r'_i}$.

We bound the competitive ratio of SG by analyzing each period independently. As $R' = \bigcup_i R'_i$ and $R^* = \bigcup_i R^*_i$, it is clear that $P_{R^*}/P_{R'} \le \alpha$ follows if we can show that $P_{R^*_i}/P_{R'_i} \le \alpha$ for all i, $1 \le i \le k$. For $1 \le i \le k$ we distinguish the following cases in order to bound $P_{R^*_i}/P_{R'_i}$. As $R'_i = \{r_i\}$, $P_{R'_i} = r$ (because $c = 0$). We need to show $P_{R^*_i} \le 2r$.

Case 1: $k = 1$. Without loss of generality, suppose r'_1 is assigned to s'_1. We claim R^* contains at most one request (r'_1). Assume that R^* contains at least two requests and the second request is r_o. As s'_2 is at location 0 at time 0, r_o would be acceptable to SG by s'_2. Hence, there cannot be such a request r_o that starts in period $[0, \infty)$. As we have shown that OPT can accept at most one request (r'_1), we get that $\frac{P_{R^*}}{P_{R'}} \le \frac{r}{r} < 2$.

Case 2: $k > 1$. For all $1 \le i \le k$, we claim that R^*_i contains at most two requests (each server accepts at most one request). Assume that s^*_q ($q \in \{1, 2\}$) accepts at least two requests. Let r_o be the second request (in order of start time) which is assigned to s^*_q in R^*_i. We distinguish three sub-cases. Without loss of generality, suppose r'_i is assigned to s'_1.

Case 2.1: $\dot{t}_{r'_i} > t_{r'_{i+1}}$ (Fig. 1.a shows an example). If $i > 1$, the period i, which is the period $[\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}, \max\{\dot{t}_{r'_i}, t_{r'_{i+1}}\}) = [\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}, \dot{t}_{r'_i})$, has length less than t. If $i = 1$, note that the period $[t_{r'_1}, \max\{\dot{t}_{r'_1}, t_{r'_2}\}) = [t_{r'_1}, \dot{t}_{r'_1})$ has length less than t and no request of R^*_1 can start before $t_{r'_1}$ during period 1, $[0, \max\{\dot{t}_{r'_1}, t_{r'_2}\})$. Therefore, each server can accept at most one request that starts during period i, and hence R^*_i contains at most two requests.

Case 2.2: $\dot{t}_{r'_i} \le t_{r'_{i+1}}$ and $\dot{t}_{r'_{i-1}} > t_{r'_i}$ (Fig. 1.b shows an example). Observe that s'_1 is at $p_{r'_i}$ at $t_{r'_i}$. As the drop-off time of r'_{i-1} is later than the pick-up time of r'_i, r'_{i-1} must be assigned to s'_2 and we have that s'_2 is at $\dot{p}_{r'_{i-1}}$ at $\dot{t}_{r'_{i-1}}$. As the first request in R^*_i does not start before $\dot{t}_{r'_{i-1}}$, we have $t_{r_o} \ge \dot{t}_{r'_{i-1}} + t$. This means that r_o would be acceptable to s'_2. Therefore, SG accepts either r_o or another request starting before t_{r_o}, and that request becomes r'_{i+1}. Hence, there cannot be such a request r_o that starts in period i.

Case 2.3: $\dot{t}_{r'_i} \le t_{r'_{i+1}}$ and $\dot{t}_{r'_{i-1}} \le t_{r'_i}$ (Fig. 1.c shows an example). As the drop-off time of r'_{i-1} is earlier than the pick-up time of r'_i, s'_2 is at the drop-off location of the request r_l (where r_l denotes the latest request that starts at or before $t_{r'_i}$ and is assigned to s'_2;

if there is no such request, let r_l be a dummy request with $\dot{t}_{r_l} = 0$ and $\dot{p}_{r_l} = 0$) at $\dot{t}_{r_l}$ and $\dot{t}_{r_l} \leq \dot{t}_{r'_{i-1}} \leq t_{r'_i}$. Observe that s'_2 does not accept any request which starts during period $[t_{r'_i}, \dot{t}_{r'_i})$, s'_2 does not start to move before $t_{r'_i}$ for serving the next request, and hence s'_2 is at $\dot{p}_{r_l}$ (0 or 1) at $t_{r'_i}$. As the first request in R^*_i does not start before $t_{r'_i}$, we have $t_{r_o} \geq t_{r'_i} + t$. This means that r_o would be acceptable to s'_2. Therefore, SG accepts either r_o or another request starting before t_{r_o}, and that request becomes r'_{i+1}. Hence, there cannot be such a request r_o that starts in period i.

As we have shown that R^*_i contains at most two requests, we get that $P_{R^*_i} \leq 2r$. Since $P_{R'_i} = r$, we have $P_{R^*_i}/P_{R'_i} \leq 2r/r = 2$. The theorem follows. ◀

▶ **Lemma 4.** *When $0 < c < r$, for all $1 < i \leq k$, one server of SG is at $p_{r'_i}$ at $t_{r'_i}$ and the other server of SG is at 0 or 1 at $\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}$.*

Proof. For $1 < i \leq k$, without loss of generality, suppose r'_i is assigned to s'_1. Observe that s'_1 is at $p_{r'_i}$ at $t_{r'_i}$.

If $\dot{t}_{r'_{i-1}} > t_{r'_i}$, then r'_{i-1} must be assigned to s'_2, and hence s'_2 is at $\dot{p}_{r'_{i-1}}$ (0 or 1) at $\dot{t}_{r'_{i-1}}$ $(= \max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}$).

If $\dot{t}_{r'_{i-1}} \leq t_{r'_i}$, then s'_2 is at 0 or 1 at the drop-off time t' ($t' \leq \dot{t}_{r'_{i-1}}$) of the latest request which is assigned to s'_2 and starts at or before $t_{r'_i}$. (If no such request exists, s'_2 is at 0 at $t' = 0$.) Suppose r_f is the first request that starts at or after $t_{r'_i}$ and is served by s'_2. If r_f does not exist, then s'_2 does not move after $\dot{t}_{r'_{i-1}}$, and s'_2 is at 0 or 1 at $\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}$ ($\dot{t}_{r'_{i-1}} \leq t_{r'_i}$). If r_f exists and r_f is accepted with cost, then $t_{r_f} - \dot{t}_{r'_i} \geq t$ ($\dot{t}_{r'_i} \leq \dot{t}_{r^n_f}$) because SG accepts a request r_j with cost only if the condition $t_{r_j} - \dot{t}_{r^n_j} \geq t$ is satisfied. That means s'_2 starts an empty move at or after $\dot{t}_{r'_i}$. If r_f exists and r_f is accepted without cost, then s'_2 starts to move at or after $t_{r'_i}$ ($t_{r_f} \geq t_{r'_i}$). Therefore s'_2 is at 0 or 1 at $t_{r'_i}$ $(= \max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\})$. ◀

▶ **Lemma 5.** *When $0 < c < r$, for all $1 \leq i \leq k$, if r'_i is accepted with cost, then one server of SG is at $p_{r'_i}$ at $t_{r'_i}$ and the other server of SG is at $\dot{p}_{r'_i}$ at $t_{r'_i}$.*

Proof. For $1 \leq i \leq k$, without loss of generality, suppose r'_i is assigned to s'_1. Observe that s'_1 is at $p_{r'_i}$ at $t_{r'_i}$.

If $i = 1$, then $p_{r'_1} = 1$ and s'_2 is at $\dot{p}_{r'_1}$ (location 0) at time 0. Suppose r_o is the first request which is assigned to s'_2. If $p_{r_o} = 0$, then s'_2 starts to move at t_{r_o} ($\geq t_{r'_1}$), and hence s'_2 is at 0 at $t_{r'_i}$. If $p_{r_o} = 1$, then $t_{r_o} \geq \dot{t}_{r'_1} + t$ because by definition SG accepts a request r_j with cost only if the condition $t_{r_j} - \dot{t}_{r^n_j} \geq t$ is satisfied. Observe that s'_2 starts to move at $t_{r_o} - t$ ($\geq \dot{t}_{r'_1}$), and hence s'_2 is at 0 at $t_{r'_i}$.

If $1 < i \leq k$, s'_2 is at 0 or 1 at $\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}$ according to Lemma 4. As r'_i is accepted with cost, $t_{r'_i} - \dot{t}_{r'_{i-1}} \geq t$ because SG accepts a request r_j with cost only if the condition $t_{r_j} - \dot{t}_{r^n_j} \geq t$ is satisfied, and hence $\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\} = t_{r'_i}$. We prove this lemma by contradiction. Assume that s'_2 is at $p_{r'_i}$ at $t_{r'_i}$. Note that r'_i is acceptable to SG by s'_2 without cost, and hence SG assigns r'_i to s'_2 because SG always assigns a request to the most economical server (Recall Algorithm 1). This contradicts our initial assumption that r'_i is assigned to s'_1. ◀

▶ **Lemma 6.** *When $0 < c < r$, for all $1 < i < k$, if r'_i is accepted without cost and r'_{i+1} is accepted with cost, then one server of SG is at $p_{r'_i}$ at $t_{r'_i}$, and the other server of SG is at $\dot{p}_{r'_i}$ at $\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}$.*

Proof. For $1 < i < k$, without loss of generality, suppose r'_i is assigned to s'_1. Observe that s'_1 is at $p_{r'_i}$ at $t_{r'_i}$. According to Lemma 4, s'_2 is at 0 or 1 at $\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}$. As r'_{i+1} is accepted with cost, $t_{r'_{i+1}} - \dot{t}_{r'_i} \geq t$ because SG accepts a request r_j with cost only if the

condition $t_{r_j} - \dot{t}_{r_j^n} \geq t$ is satisfied. Note that $p_{r'_{i+1}} = p_{r'_i}$, otherwise r'_{i+1} is acceptable to SG by s'_1 without cost.

We prove this lemma by contradiction. Assume that s'_2 is at $p_{r'_i}$ at $\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}$. Suppose $r_f = r'_{i+1}$. Observe that $p_{r'_f} = p_{r'_i}$ and $t_{r'_f} \geq \dot{t}_{r'_i} + t \geq \max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}$. From this it follows that r'_f is acceptable to SG by s'_2 without cost, and hence SG assigns r'_f to s'_2 because SG always assigns a request to the most economical server (Recall Algorithm 1). This contradicts the statement that r'_{i+1} is accepted with cost. ◀

▶ **Lemma 7.** *When $0 < c < r$, for all $1 < i < k$, if r'_i is accepted without cost and r'_{i+1} is accepted with cost, then r'_{i-1} must be accepted without cost.*

Proof. For $1 < i < k$, without loss of generality, suppose r'_{i-1} is assigned to s'_1. Observe that s'_1 is at $p_{r'_{i-1}}$ at $t_{r'_{i-1}}$ (and is at $\dot{p}_{r'_{i-1}}$ at $\dot{t}_{r'_{i-1}}$). We prove this lemma by contradiction. Assume r'_{i-1} is accepted with cost. According to Lemma 5, s'_2 is at $\dot{p}_{r'_{i-1}}$ at $t_{r'_{i-1}}$. As r'_i is accepted without cost, the pick-up location of r'_i is $\dot{p}_{r'_{i-1}}$. Suppose $r_f = r'_{i+1}$. Observe that $t_{r_f} \geq \dot{t}_{r'_i} + t$ (because r_f is accepted with cost) and $p_{r_f} = p_{r'_i} = \dot{p}_{r'_{i-1}}$ (otherwise, the server that has served r'_i could accept r_f without cost).

If r'_i is assigned to s'_1, then s'_2 does not accept any request which starts in period $[\max\{\dot{t}_{r'_{i-2}}, t_{r'_{i-1}}\}, t_{r_f})$, and hence s'_2 is at $\dot{p}_{r'_{i-1}}$ in period $[\max\{\dot{t}_{r'_{i-2}}, t_{r'_{i-1}}\}, t_{r_f} - t)$. If r'_i is assigned to s'_2, then s'_1 does not accept any request which starts in period $[\dot{t}_{r'_{i-1}}, t_{r_f})$, and hence s'_1 is at $\dot{p}_{r'_{i-1}}$ in period $[\dot{t}_{r'_{i-1}}, t_{r_f} - t)$. As r_f is released and $\tilde{t}_{r_f} = t_{r_f} - a \leq t_{r_f} - t$, server s'_q (for a $q \in \{1, 2\}$) is at $\dot{p}_{r'_{i-1}}$ and does not plan to move, hence r_f is acceptable to SG by s'_q without cost. From this it follows that r_f will be accepted by SG without cost because SG always assigns a request to the most economical server. This contradicts the statement that r'_{i+1} is accepted with cost. ◀

For simplification of the analysis, we suppose that the OPT servers make an empty movement only if they do so in order to serve a request r_i such that the pick-up location of r_i is the pick-up location of the previous request which is assigned to the same server, or the pick-up location is 1 if r_i is the first request which is assigned to a server s^*_q ($q \in \{1, 2\}$), and we suppose that for all such requests r_i ($r_i \in R^*$), the OPT server serving r_i makes an empty movement between $t_{r_i} - t$ and t_{r_i}. This simplification does not affect the validity of R^*, and does not decrease P_{R^*}.

▶ **Theorem 8.** *Algorithm SG is 2-competitive for 2S2L if $0 < c < r$.*

Proof. Assume that SG accepts k ($k = |R'|$) requests. We partition the time horizon $[0, \infty)$ into k' ($1 \leq k' \leq k$) intervals (periods) that can be analyzed independently. We partition the time horizon based on Algorithm 2, in such a way that all requests in the first period are accepted with cost (if r'_1 is accepted with cost), and exactly one request of each period (except the first period if r'_1 is accepted with cost), the first request of each period, is accepted without cost. Denote the request number in R' (in order of starting time) of the first request of period j ($1 \leq j \leq k'$) by l_j. For $1 < j < k'$, SG j period is $[t_{r'_{l_j}}, t_{r'_{l_{j+1}}})$. SG 1 period is $[0, t_{r'_{l_2}})$ and SG k' period is $[t_{r'_{l_{k'}}}, \infty)$ (If $k' = 1$, there is only a single period $[0, \infty)$). We set $l_{k'+1} = k + 1$, $t_{r'_0} = 0$ and $t_{r'_{k+1}} = \infty$. Let R'_j ($1 \leq j \leq k'$) denote the set of requests accepted by SG that start in SG j period. For $1 < j \leq k'$, if $t_{r'_{l_{j-1}}} = t_{r'_{l_j}}$, let $R'_{j-1} = \{r'_{l_{j-1}}\}$ and $R'_j = \{r'_{l_j}, r'_{l_j+1}, ..., r'_{l_{j+1}-1}\}$. Note that there are exactly $l_{j+1} - l_j$ ($l_{j+1} - l_j \geq 1$) requests in R'_j ($1 \leq j \leq k'$), and $R'_j = \{r'_{l_j}, r'_{l_j+1}, ..., r'_{l_{j+1}-1}\}$.

For all $1 < j \leq k'$, we have the following property: if $|R'_j| = 1$, then r'_{l_j} is accepted without cost; if $|R'_j| > 1$, then r'_{l_j} is accepted without cost, the remaining requests in R'_j

Algorithm 2 Partition Rule (PR).

Initialization: $k = |R'|$, $k' = 1$, $j = 1$, $l_j = j$ for all $1 \le j \le k$.
For $i = 2$ to k
if r'_i is accepted without cost then
$j = j + 1$, $l_j = i$;
$k' = j$, $l_{k'+1} = k + 1$.

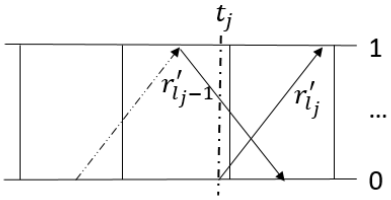


Figure 2 An example of t_j.

are accepted with cost. For $j = 1$, if $r'_1 (= r'_{l_1})$ is accepted with cost, all requests in R'_1 are accepted with cost; if r'_1 is accepted without cost, then except r'_1 all requests in R'_1 are accepted with cost.

▶ **Definition 9.** For $1 < j \le k'$, t_j is defined as follows: $t_j = t_{r'_{l_j}}$ if r'_{l_j-1} is accepted with cost, r'_{l_j} is accepted without cost, $\dot{t}_{r'_{l_j-1}} > t_{r'_{l_j}}$ and $\dot{p}_{r'_{l_j-1}} = p_{r'_{l_j}}$ (Fig. 2 shows an example). Otherwise, $t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$. $t_{k'+1} = t_{r'_{k+1}} = \infty$.

For $1 < j \le k'$, $t_{j+1} = t_{r'_{l_{j+1}}}$ or $t_{j+1} = \max\{\dot{t}_{r'_{l_{j+1}-1}}, t_{r'_{l_{j+1}}}\}$, and $t_j = t_{r'_{l_j}}$ or $t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$. Because $t_{r'_{l_j}} \le t_{r'_{l_{j+1}}}$ and $\dot{t}_{r'_{l_j-1}} \le t_{r'_{l_{j+1}}}$ (if r'_{l_j-1} and r'_{l_j} are assigned to the same server, then $\dot{t}_{r'_{l_j-1}} \le t_{r'_{l_j}}$; and if r'_{l_j-1} and r'_{l_j} are assigned to different servers, then $\dot{t}_{r'_{l_j-1}} \le t_{r'_{l_{j+1}}}$), $t_j \le t_{j+1}$ if $t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ and $t_{j+1} = t_{r'_{l_{j+1}}}$. As $t_j \le \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ and $t_{j+1} \ge t_{r'_{l_{j+1}}}$, we have that $t_j \le t_{j+1}$ always holds. For $1 < j \le k'$, OPT period j is defined as $[t_j, t_{j+1})$. OPT period 1 is defined as $[0, t_2)$ (If $k' = 1$, there is only a single period $[0, \infty)$). Let R^*_j denote the set of requests accepted by OPT that start in OPT period j, and $R^*_i = \emptyset$ if $t_j = t_{j+1}$.

For all $1 < j \le k'$, r'_{l_j} starts at time $t_{r'_{l_j}}$ and the first request of R^*_j starts during the interval $[t_j, t_{j+1})$ where $t_j = t_{r'_{l_j}}$ or $t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ (recall the definition of t_j). Furthermore, r'_1 is the first acceptable request in R, and so the first request of R^*_1 cannot start before r'_1. Hence, for all $1 \le j \le k'$, the first request in R^*_j cannot start before $t_{r'_{l_j}}$.

We bound the competitive ratio of SG by analyzing each period independently. As $R' = \bigcup_{j=1}^{k'} R'_j$ and $R^* = \bigcup_{j=1}^{k'} R^*_j$, it is clear that $P_{R^*}/P_{R'} \le \alpha$ follows if we can show that $P_{R^*_j}/P_{R'_j} \le \alpha$ for all $1 \le j \le k'$. For $1 \le j \le k'$, if $t_j = t_{j+1}$, then $R^*_i = \emptyset$ and hence $P_{R^*_i} = 0$. Otherwise, for $1 \le j \le k'$ we distinguish the following cases in order to bound $P_{R^*_j}/P_{R'_j}$.

Case 1: $j = 1$. The first request of SG period 1 is r'_1. Without loss of generality, suppose r'_1 is assigned to s'_1.

Case 1.1: r'_1 is accepted with cost. Note that all requests in R'_j are accepted with cost and $P_{R'_1} = (l_2 - l_1)(r - c)$ (if $k' = 1$, then $P_{R'} = k(r - c)$). Observe that $p_{r'_i} = 1$ $(1 \le i < l_2)$ and all requests in R'_1 are assigned to s'_1 by the definition of Algorithm 1. As r'_{l_2-1} is accepted with cost, one server is at $p_{r'_{l_2-1}}$ at $t_{r'_{l_2-1}}$ (and this server is at $\dot{p}_{r'_{l_2-1}}$ at $\dot{t}_{r'_{l_2}-1}$),

and the other server is at $\dot{p}_{r'_{l_2-1}}$ at $t_{r'_{l_2-1}}$ (by Lemma 5). As r'_{l_2} is accepted without cost, we have $\dot{p}_{r'_{l_2-1}} = p_{r'_{l_2}}$. If $k' = 1$, $t_2 = \infty$. If $k' > 1$, then $t_2 = t_{r'_{l_2}}$: if $\dot{t}_{r'_{l_2-1}} > t_{r'_{l_2}}$, $t_2 = t_{r'_{l_2}}$ because $p_{r'_{l_2}} = \dot{p}_{r'_{l_2-1}}$, r'_{l_2-1} is accepted with cost and r'_{l_2} is accepted without cost; if $\dot{t}_{r'_{l_2-1}} \leq t_{r'_{l_2}}$, $t_2 = \max\{\dot{t}_{r'_{l_2-1}}, t_{r'_{l_2}}\} = t_{r'_{l_2}}$. As s'_2 does not accept any request which starts before $t_{r'_{l_2}}$ and s'_2 would not accept any request with cost which starts in period $[t_{r'_{l_2}}, \dot{t}_{r'_{l_2}})$ (Recall that Algorithm 1 accepts a request r_j with cost only if $t_{r_j} - \dot{t}_{r^n_j} \geq t$ is satisfied.), s'_2 is at 0 in period $[0, t_{r'_{l_2}}]$. We claim that R^*_j only contains requests which start at 1. Otherwise, the request is acceptable to SG by s'_2 without cost. Assume that R^*_j contains a request r_o which start at location 0. As $t_{r_o} \leq t_2 = t_{r'_{l_2}}$, r_o is acceptable to SG by s'_2 without cost. Therefore, SG accepts either r_o or another request starting before t_{r_o}, and that request becomes r'_{l_2}. Hence, there cannot be such a request r_o in R^*_j.

Note that each server of OPT does not accept any request which starts in period $[0, t_{r'_1})$. For all $l_1 \leq i \leq l_2 - 2$, we claim that each server of OPT can accept at most one request which starts during period $[t_{r'_i}, t_{r'_{i+1}})$ ($l_1 \leq i \leq l_2 - 2$), or period $[t_{r'_{l_2-1}}, t^*)$ (if $k' > 1$, $t^* = t_{r'_{l_2}}$; if $k' = 1$, $t^* = t_{r'_k} + 2t$). Assume that s^*_q ($q \in \{1, 2\}$) accepts at least two requests in one of those periods. Let r_o be the second request (in order of start time) which is assigned to s^*_q and starts during period $[t_{r'_i}, t_{r'_{i+1}})$ ($l_1 \leq i \leq l_2 - 2$) or period $[t_{r'_{l_2-1}}, t^*)$. As the request does not start before $t_{r'_i}$ ($l_1 \leq i \leq l_2 - 1$), we have $t_{r_o} \geq t_{r'_i} + 2t$. r_o is acceptable to SG with cost. Therefore, SG accepts either r_o or another request starting before t_{r_o}, and that request becomes r'_{i+1} ($l_1 \leq i < l_2$). Hence, there cannot be such a request r_o that starts during period $[t_{r'_i}, t_{r'_{i+1}})$ ($l_1 \leq i \leq l_2 - 2$) or period $[t_{r'_{l_2-1}}, t^*)$. Therefore OPT can accept at most $2(l_2 - l_1)$ ($= 2(l_2 - 2 - l_1 + 1 + 1)$) requests that start during period $[t_{r'_1}, t^*)$.

When $k' = 1$, we claim that OPT does not accept any request which starts in period $[t^*, \infty)$. Without loss of generality we assume that OPT accepts at least one request. Let r_o be the request in R^*_1 that starts during period $[t^*, \infty)$. As $t_{r_o} \geq t_{r'_k} + 2t$. r_o is acceptable to SG. Therefore, SG accepts either r_o or another request starting before t_{r_o}, and that request becomes r'_{k+1}. Hence, there cannot be such a request r_o that starts in period $[t^*, \infty)$.

As we have shown that R^*_j contains at most $2(l_2 - l_1)$ requests and the pick-up locations of them are the same (location 1), we get that $P_{R^*_j} \leq 2(l_2 - l_1)(r - c)$. Since $P_{R'_j} = (l_2 - l_1)(r - c)$, we have $P_{R^*_j}/P_{R'_j} \leq 2(l_2 - l_1)(r - c)/((l_2 - l_1)(r - c)) = 2$.

Case 1.2: r'_1 is accepted without cost. If $k = 1$, then $k' = 1$, s'_2 is at 0 in period $[0, \infty)$. If $k > 1$, we claim that r'_2 is also accepted without cost. Assume that r'_2 is accepted with cost, we have $t_{r'_2} - \dot{t}_{r'_1} > t$ because Algorithm 1 accepts a request r_j with cost only if $t_{r_j} - \dot{t}_{r^n_j} \geq t$ is satisfied. If $p_{r'_2} = 0$, r'_2 is acceptable to SG by s'_2 without cost; if $p_{r'_2} = 1$, r'_2 is acceptable to SG by s'_1 without cost. Therefore s'_2 must be accepted by SG without cost because by definition (see Algorithm 1) SG always assigns a request to the most economical server. This contradicts the assumption that r'_2 is accepted with cost. Observe that $t_2 = \max\{\dot{t}_{r'_1}, t_{r'_2}\}$ (Recall from the definition of t_2 that $t_2 = t_{r'_2}$ only if r'_1 is accepted with cost), $|R'_1| = 1$ and hence $P_{R'_1} = r$. As s'_2 does not accept any request which starts before $t_{r'_2}$ and s'_2 would not accept any request with cost which starts in period $[t_{r'_2}, \dot{t}_{r'_2})$ (Recall that Algorithm 1 accepts a request r_j with cost only if $t_{r_j} - \dot{t}_{r^n_j} \geq t$ is satisfied.), s'_2 is at 0 in period $[0, t_{r'_2}]$.

We claim that R^*_1 contains at most two requests (each server serves at most one request). Assume that s^*_q ($q \in \{1, 2\}$) accepts at least two requests. Let r_o be the second request (in order of start time) which is assigned to s^*_q in R^*_1. As the first request in R^*_1 does not

start before $t_{r'_1}$, we have $t_{r_o} \geq t_{r'_1} + t$. If $p_{r_o} = \dot{p}_{r_1}$, r_o is acceptable to SG by s'_1 without cost; if $p_{r_o} = p_{r_1}$, r_o is acceptable to SG by s'_2 without cost. Hence, there cannot be such a request in R^*_1. Since $P_{R'_1} = r$, we have $P_{R^*_1} \leq 2r$, and hence $P_{R^*_1}/P_{R'_1} \leq 2r/r = 2$.

Case 2: $j > 1$ $(1 < j \leq k')$. The first request of SG period j is r'_{l_j}.Without loss of generality, suppose r'_{l_j} is assigned to s'_1. We distinguish the following cases based on $|R'_j|$.

Case 2.1: $|R'_j| = 1$. Note that r'_{l_j} is accepted without cost. We distinguish two sub-cases.

(1) $\dot{t}_{r'_{l_j}} > t_{r'_{l_{j+1}}}$. Because r'_{l_j} $(= r'_{l_{j+1}-1})$ is accepted without cost, $t_{j+1} = \max\{\dot{t}_{r'_{l_j}}, t_{r'_{l_{j+1}}}\} = \dot{t}_{r'_{l_j}}$ (Recall that $t_{j+1} = t_{r'_{l_{j+1}}}$ only if $r'_{l_{j+1}-1}$ is accepted with cost by the definition of t_{j+1}). As OPT period j $[t_j, t_{j+1})$ has length less than t ($t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ or $t_j = t_{r'_{l_j}}$), each server of OPT can accept at most one request in R^*_j, and hence R^*_j contains at most two requests.

(2) $\dot{t}_{r'_{l_j}} \leq t_{r'_{l_{j+1}}}$ $(t_{r'_{l_{j+1}}} = \infty$ if $j = k')$. Note that $t_{j+1} = t_{r'_{l_{j+1}}}$. There are two sub-cases based on the position of s'_2 at $\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ (recall that by Lemma 4, s'_2 is at $p_{r'_{l_j}}$ or $\dot{p}_{r'_{l_j}}$ at time $\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$).

The first sub-case is that s'_2 is at $p_{r'_{l_j}}$ at $\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$. We claim that R^*_j contains at most two requests (each server serves at most one request). Assume that s^*_q $(q \in \{1, 2\})$ accepts at least two requests. Let r_o be the second request (in order of start time) which is assigned to s^*_q in R^*_j. As the requests in R^*_j do not start before $t_{r'_{l_j}}$, we have $t_{r_o} \geq t_{r'_{l_j}} + t$. If $p_{r_o} = \dot{p}_{r'_{l_j}}$, r_o is acceptable to SG by s'_1 without cost; if $p_{r_o} = p_{r'_{l_j}}$, r_o is acceptable to SG by s'_2 without cost. Therefore, SG accepts either r_o or another request starting before t_{r_o}, and that request becomes r'_{l_j+1}. Hence, there cannot be such a request r_o that starts in OPT period j.

The second sub-case is that s'_2 is at $\dot{p}_{r'_{l_j}}$ at $\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$. Note that $t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ (Recall from the definition of t_j that $t_j = t_{r'_{l_j}}$ only if $\dot{t}_{r'_{l_j-1}} > t_{r'_{l_j}}$ and $\dot{p}_{r'_{l_j-1}} = p_{r'_{l_j}}$ are satisfied. From this it follows that r'_{l_j-1} must be assigned to s'_2, that means s'_2 is at $\dot{p}_{r'_{l_j-1}}$ $(= p_{r'_{l_j}})$ at $\dot{t}_{r'_{l_j-1}}$ $(= \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\})$. This contradicts the initial assumption that s'_2 is at $\dot{p}_{r'_{l_j}}$ at $\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$.). We claim that R^*_j contains at most two requests (each server serves at most one request) and the pick-up locations of these two requests are $p_{r'_{l_j}}$. Assume that R^*_j contains a request r_i which starts at $\dot{p}_{r'_{l_j}}$. As the requests in R^*_j cannot start before t_j $(t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\})$, r_i is acceptable to s'_2 (without cost) as s'_2 is at $\dot{p}_{r'_{l_j}}$ at $\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$. Hence, there cannot be such a request r_i that starts in OPT period j. Next assume that s^*_q $(q \in \{1, 2\})$ accepts at least two requests. Let r_i and r_o be the first and second request (in order of start time) which is assigned to s^*_q in R^*_j. As the requests in R^*_j do not start before $t_{r'_{l_j}}$ and the pick-up location of r_i and r_o both are $p_{r'_{l_j}}$, we have $t_{r_o} \geq t_{r'_{l_j}} + 2t$. If $p_{r_o} = \dot{p}_{r'_{l_j}}$, r_o is acceptable to SG by s'_1 without cost; if $p_{r_o} = p_{r'_{l_j}}$, r_o is acceptable to SG by s'_2 with cost. Therefore, SG accepts either r_o or another request starting before t_{r_o}, and that request becomes r'_{l_j+1} (if it is accepted without cost) or gets added to R'_j (if it is accepted with cost). Hence, there cannot be such a request r_o that starts in OPT period j.

As we have shown that R^*_j contains at most two requests, we get that $P_{R^*_j} \leq 2r$. Since $P_{R'_j} = r$, we have $P_{R^*_j}/P_{R'_j} \leq 2r/r = 2$.

Case 2.2: $|R'_j| > 1$. Note that r'_{l_j} is accepted without cost and r'_{l_j+1} is accepted with cost. We have that s'_2 is at $\dot{p}_{r'_{l_j}}$ at $\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ by Lemma 6, and that r'_{l_j-1} is accepted without

cost by Lemma 7. Hence, $t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ (recall from the definition of t_j that $t_j = t_{r'_{l_j}}$ only if r'_{l_j-1} is accepted with cost). As $r'_{l_{j+1}-1}$ is accepted with cost, one server is at $p_{r'_{l_{j+1}-1}}$ at $t_{l_{j+1}-1}$ (and this server is at $\dot{p}_{r'_{l_{j+1}-1}}$ at $\dot{t}_{r'_{l_{j+1}-1}}$), and the other server is at $\dot{p}_{r'_{l_{j+1}-1}}$ at $t_{r'_{l_{j+1}-1}}$ (Recall Lemma 5). As $r'_{l_{j+1}}$ is accepted without cost, we have $\dot{p}_{r'_{l_{j+1}-1}} = p_{r'_{l_{j+1}}}$. If $\dot{t}_{r'_{l_{j+1}-1}} > t_{r'_{l_{j+1}}}$ $(1 \le j < k')$, $t_{j+1} = t_{r'_{l_{j+1}}}$ according to the definition of t_s $(1 \le s \le k')$. If $\dot{t}_{r'_{l_{j+1}-1}} \le t_{r'_{l_{j+1}}}$ $(1 \le j < k')$, $t_{j+1} = \max\{\dot{t}_{r'_{l_{j+1}-1}}, t_{r'_{l_{j+1}}}\} = t_{r'_{l_{j+1}}}$. Hence, $t_{j+1} = t_{r'_{l_{j+1}}}$ $(1 \le j < k')$. Observe that if $j = k'$, $t_{j+1} = t_{r'_{l_{j+1}}} = \infty$.

We claim that R^*_j only contains requests which start at $p_{r'_{l_j}}$. Assume that R^*_j contains a request r_i which starts at $\dot{p}_{r'_{l_j}}$. As the first request in R^*_j cannot start before t_j, we have $t_{r_i} \ge t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$. As s'_2 is at $\dot{p}_{r'_{l_j}}$ at $\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ and s'_2 does not accept any request which starts in period $[\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}, t_{r_i})$, and hence r_i is acceptable to SG by s'_2 without cost. This contradicts the property of R'_j that except r'_{l_j} all requests in R'_j are accepted with cost. Hence, there cannot be such a request r_i that starts in OPT period j.

We claim that each server of OPT can accept at most one request which starts in period $[t_j, t_{r'_{l_j+1}})$, or period $[t_{r'_i}, t_{r'_{i+1}})$ $(l_j+1 \le i \le l_{j+1}-2)$, or period $[t_{r'_{l_{j+1}-1}}, t^*)$ (if $1 \le j < k'$, $t^* = t_{r'_{l_{j+1}}}$; if $j = k'$, $t^* = t_{r'_k} + 2t$). Assume that s^*_q $(q \in \{1, 2\})$ accepts at least two requests in one of these periods. Let r_o be the second request (in order of start time) which is assigned to s^*_q and starts in one of these periods. As the requests in R^*_j that start in one of these periods do not start before the corresponding $t_{r'_i}$ $(l_j \le i \le l_{j+1} - 1)$ and have the same pick-up location $p_{r'_{l_j}}$, we have $t_{r_o} \ge t_{r'_{l_j}} + 2t$. r_o is acceptable to SG with cost. Therefore, SG accepts either r_o or another request starting before t_{r_o}, that request becomes r'_{i+1} $(l_j \le i \le l_{j+1} - 2)$, or we get a contradiction to $r'_{l_{j+1}-1}$ being the last request that is accepted with cost and starts in period $[t_{r'_{l_{j+1}-1}}, t^*)$ $(i = l_{j+1} - 1)$. Hence, there cannot be such a request r_o that starts in period $[t_j, t_{r'_{l_j+1}})$, or period $[t_{r'_i}, t_{r'_{i+1}})$ $(l_j + 1 \le i \le l_{j+1} - 1)$. Therefore OPT can accept at most $2(l_{j+1} - l_j)$ $(= 2(l_{j+1} - 2 - (l_j + 1) + 1 + 2))$ requests that start in period $[t_{r'_{l_{j+1}}}, t^*)$.

When $j = k'$, we claim that OPT does not accept any request which starts in period $[t^*, \infty)$. Without loss of generality we assume that OPT accepts at least one request. Let r_o be the request in R^*_j which starts during period $[t^*, \infty)$. As $t_{r_o} \ge t_{r'_k} + 2t$, r_o is acceptable to SG with cost. Therefore, SG accepts either r_o or another request starting before t_{r_o}, and that request becomes r'_{k+1}. Hence, there cannot be such a request r_o that starts in period $[t^*, \infty)$.

As we have shown that R^*_j contains at most $2(l_{j+1} - l_j)$ requests and the pick-up locations of them are the same $(p_{r'_{l_j}})$, we get that $P_{R^*_j} \le 2r + 2(l_{j+1} - l_j - 1)(r - c)$. Since $P_{R'_j} = r + (l_{j+1} - l_j - 1)(r - c)$, we have $P_{R^*_j}/P_{R'_j} \le (2r + 2(l_{j+1} - l_j - 1)(r - c))/(r + (l_{j+1} - l_j - 1)(r - c)) = 2$.

Because $P_{R^*_j}/P_{R'_j} \le 2$ holds for all $1 \le j \le k'$, we have $P_{R^*}/P_{R'} \le 2$. This proves the theorem. ◀

4 Conclusion

We have studied an on-line problem with two servers and two locations that is motivated by applications such as car sharing and taxi dispatching. The upper bounds for the 2S2L problem are all achieved by the smart greedy algorithm. A number of directions for future

work arise from this work. If there are k servers, does a kind of greedy algorithm still work well? Furthermore, it would be interesting to extend our results to the case of more than two locations. It would be interesting to determine how the constraints on the servers affect the competitive ratio for the general car-sharing problem with k servers and m locations.

References

1 Norbert Ascheuer, Sven Oliver Krumke, and Jörg Rambau. Online dial-a-ride problems: Minimizing the completion time. In Horst Reichel and Sophie Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000, Proceedings*, volume 1770 of *LNCS*, pages 639–650. Springer, 2000. doi:10.1007/3-540-46541-3_53.

2 Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8–15, 2010. doi:10.1016/j.ejor.2009.04.024.

3 Antje Bjelde, Yann Disser, Jan Hackfeld, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Kevin Schewior, Miriam Schlöter, and Leen Stougie. Tight bounds for online TSP on the line. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 994–1005. SIAM, 2017. doi:10.1137/1.9781611974782.63.

4 Katerina Böhmová, Yann Disser, Matús Mihalák, and Rastislav Srámek. Scheduling transfers of resources over time: Towards car-sharing with flexible drop-offs. In Evangelos Kranakis, Gonzalo Navarro, and Edgar Chávez, editors, *12th Latin American Symposium on Theoretical Informatics (LATIN 2016)*, volume 9644 of *LNCS*, pages 220–234. Springer, 2016. doi:10.1007/978-3-662-49529-2_17.

5 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

6 Ananya Christman, William Forcier, and Aayam Poudel. From theory to practice: maximizing revenues for on-line dial-a-ride. *J. Comb. Optim.*, 35(2):512–529, 2018. doi:10.1007/s10878-017-0188-z.

7 Sven Oliver Krumke, Willem de Paepe, Diana Poensgen, Maarten Lipmann, Alberto Marchetti-Spaccamela, and Leen Stougie. On minimizing the maximum flow time in the online dial-a-ride problem. In Thomas Erlebach and Giuseppe Persiano, editors, *Approximation and Online Algorithms, Third International Workshop, WAOA 2005, Palma de Mallorca, Spain, October 6-7, 2005, Revised Papers*, volume 3879 of *LNCS*, pages 258–269. Springer, 2006. doi:10.1007/11671411_20.

8 Kelin Luo, Thomas Erlebach, and Yinfeng Xu. Car-sharing between two locations: Online scheduling with flexible advance bookings. In *Proceedings of the 24th International Computing and Combinatorics Conference, COCOON 2018*, LNCS. Springer, 2018. To appear.

9 Fanglei Yi and Lei Tian. On the online dial-a-ride problem with time-windows. In Nimrod Megiddo, Yinfeng Xu, and Binhai Zhu, editors, *Algorithmic Applications in Management, First International Conference, AAIM 2005, Xian, China, June 22-25, 2005, Proceedings*, volume 3521 of *LNCS*, pages 85–94. Springer, 2005. doi:10.1007/11496199_11.

The Robustness of LWPP and WPP, with an Application to Graph Reconstruction

Edith Hemaspaandra
Rochester Institute of Technology, Rochester, NY, USA

Lane A. Hemaspaandra
University of Rochester, Rochester, NY, USA

Holger Spakowski
University of Cape Town, Rondebosch, South Africa

Osamu Watanabe
Tokyo Institute of Technology, Tokyo, Japan

Abstract

We show that the counting class LWPP [8] remains unchanged even if one allows a polynomial number of gap values rather than one. On the other hand, we show that it is impossible to improve this from polynomially many gap values to a superpolynomial number of gap values by relativizable proof techniques.

The first of these results implies that the Legitimate Deck Problem (from the study of graph reconstruction) is in LWPP (and thus low for PP, i.e., $\mathrm{PP}^{\text{Legitimate Deck}} = \mathrm{PP}$) if the weakened version of the Reconstruction Conjecture holds in which the number of nonisomorphic preimages is assumed merely to be polynomially bounded. This strengthens the 1992 result of Köbler, Schöning, and Torán [15] that the Legitimate Deck Problem is in LWPP if the Reconstruction Conjecture holds, and provides strengthened evidence that the Legitimate Deck Problem is not NP-hard.

We additionally show on the one hand that our main LWPP robustness result also holds for WPP, and also holds even when one allows both the rejection- and acceptance- gap-value targets to simultaneously be polynomial-sized lists; yet on the other hand, we show that for the #P-based analog of LWPP the behavior much differs in that, in some relativized worlds, even two target values already yield a richer class than one value does.

2012 ACM Subject Classification Theory of computation → Complexity classes

Keywords and phrases structural complexity theory, robustness of counting classes, the legitimate deck problem, PP-lowness, the Reconstruction Conjecture

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.51

Related Version A full version of the paper can be found at [12], `https://arxiv.org/abs/1711.01250v2`.

1 Introduction

Nothing is more natural than wanting to better understand an object by knowing what it can and cannot do. Whether wondering how fast a (rental?) car can go in reverse or wondering if $\mathrm{NP}^{\mathrm{NP}}$ can without loss of generality be assumed to ask at most one question per nondeterministic path (as it indeed can, as is implicit in the quantifier characterization [27, 30] of $\mathrm{NP}^{\mathrm{NP}}$), we both in life and as theoreticians want to find how robust things are.

We are often particularly happy when a class proves to be quite robust under definitional perturbations. Such robustness on one hand suggests that perhaps there is something broadly natural about the class, and on the other hand such robustness often makes it easier to put the class to use.

This paper shows that the counting classes LWPP and WPP, defined in 1994 in the seminal work of Fenner, Fortnow, and Kurtz [8] on gap-based counting classes, are quite robust. Even though their definitions are in terms of having the gap function (the difference between the number of accepting and rejecting paths of a machine) hit a single target value, we prove (in Section 3) that one can allow a list of up to polynomially many target values without altering the descriptive richness of the class, i.e., without changing the class.

We then apply this to the question of whether the Legitimate Deck Problem is in LWPP.

The Legitimate Deck Problem (a formal definition will be given in Section 4) is the decision problem of determining whether, given a multiset of (unlabeled) graphs, there exists a graph G such that that multiset is precisely (give or take isomorphisms) the multiset of one-node-deleted subgraphs of G (a.k.a. the *deck* of G) [17]. *The Reconstruction Conjecture* [13, 28] – which in the wake of the resolution of the Four-Color Conjecture was declared by the editorial board of the Journal of Graph Theory to be the foremost open problem in graph theory [7] – states that every graph with three or more nodes is uniquely determined (give or take isomorphisms) by its multiset of one-node-deleted subgraphs. The Legitimate Deck Problem was defined in 1978 by Nash-Williams [22], in his paper that framed the algorithmic/complexity issues related to reconstructing graphs – such as telling whether a given deck is legitimate (i.e., is the deck of some graph).

Our application of our LWPP robustness result to the question of whether the Legitimate Deck Problem is in LWPP is the following. The strongest previous evidence of the simplicity of the Legitimate Deck Problem is the 1992 result of Köbler, Schöning, and Torán [15] that the Legitimate Deck Problem is in LWPP (and thus is PP-low, i.e., $\mathrm{PP}^{\text{Legitimate Deck}} = \mathrm{PP}$) if the Reconstruction Conjecture (i.e., that each deck whose elements all have at least two nodes has at most *one* preimage, give or take isomorphisms) holds. Using this paper's main robustness result as a tool, Section 4 proves that the Legitimate Deck Problem is in LWPP (and thus is PP-low) if a weakened version of the Reconstruction Conjecture holds, namely, that each deck has at most *a polynomial number* of nonisomorphic preimages.

This weakened version is not known to be equivalent to the Reconstruction Conjecture itself. And so our result for the first time gives a path to proving that the Legitimate Deck Problem is PP-low that does not require one to, on the way, resolve the foremost open problem in graph theory [7].

We started this section by noting that it is natural to want to know both flexibilities and limitations of classes. Our main result is about flexibility: going from one target gap to instead a polynomial number. But are we leaving money on the table? Could we extend our result to slightly superpolynomial numbers of target gaps, or even to exponential numbers of target gaps? In Section 5 we note that if the robustness of LWPP were to hold up to exponentially many target gaps, then NP would be in LWPP and so would be PP-low (i.e., $\mathrm{PP}^{\mathrm{NP}} = \mathrm{NP}$); yet NP is widely suspected not to be PP-low. We also, by encoding nondeterministic oracle Turing machines by low-degree multivariate polynomials so as to capture the gap functions of those machines, show an oracle relative to which robustness fails for all superpolynomial numbers of target gaps; thus, no extension beyond this paper's polynomial-number-of-target-gaps robustness result for LWPP can be proven by a relativizable proof. And for the #P-based analogue of LWPP, in Section 6 we show that even allowing two target values yields, in some relativized worlds, a richer class than one

target value. We also (in the final part of Section 3) extend our main result to show that the simultaneous expansion to polynomial-sized lists of both the acceptance *and rejection* target-gap lists still, for LWPP and WPP, yields the classes LWPP and WPP.

To summarize: In this paper, we prove that LWPP and WPP are robust enough that they remain unchanged when their single target gap is allowed to be expanded to a polynomial-sized list; we apply this new robustness of LWPP to show that the PP-lowness of the Legitimate Deck Problem follows from a weaker hypothesis than was previously known; we show that our polynomial robustness of LWPP is optimal with respect to relativizable proofs; and we prove a number of related results on limitations and extensions.

2 Preliminaries

We first present the definitions of many of the counting classes that we will be speaking of, taking the definitions from the seminal paper of Fenner, Fortnow, and Kurtz [8].

▶ **Definition 2.1** ([8]).

1. For each nondeterministic polynomial-time Turing machine N, the function $\mathrm{acc}_N : \Sigma^* \to \mathbb{N}$ is defined such that for every $x \in \Sigma^*$, $\mathrm{acc}_N(x)$ equals the number of accepting computation paths of N on input x.
2. For each nondeterministic polynomial-time Turing machine N, the function $\mathrm{rej}_N : \Sigma^* \to \mathbb{N}$ is defined such that for every $x \in \Sigma^*$, $\mathrm{rej}_N(x)$ equals the number of rejecting computation paths of N on input x.
3. For each nondeterministic polynomial-time Turing machine N, the function $\mathrm{gap}_N : \Sigma^* \to \mathbb{Z}$ is defined such that for every $x \in \Sigma^*$,

$$\mathrm{gap}_N(x) = \mathrm{acc}_N(x) - \mathrm{rej}_N(x).$$

▶ **Definition 2.2** ([8]).

$$\mathrm{GapP} = \{\mathrm{gap}_N \mid N \text{ is a polynomial-time nondeterministic Turing machine}\}.$$

▶ **Definition 2.3** ([23, 8]). SPP is the class of all sets A such that there exists a GapP function g such that for all $x \in \Sigma^*$,

$$x \in A \Longrightarrow g(x) = 1$$
$$x \notin A \Longrightarrow g(x) = 0.$$

The following class, WPP, is potentially larger than SPP. Instead of the "target value" 1 for the case $x \in A$, we allow a target value $f(x)$, where f may be any polynomial-time computable function whose image does not contain 0. FP denotes the class of polynomial-time computable functions.

▶ **Definition 2.4** ([8]). WPP is the class of all sets A such that there exists a GapP function g and a function $f \in \mathrm{FP}$ that maps from Σ^* to $\mathbb{Z} - \{0\}$ such that for all $x \in \Sigma^*$,

$$x \in A \Longrightarrow g(x) = f(x)$$
$$x \notin A \Longrightarrow g(x) = 0.$$

The class LWPP is the same as WPP except that the "target function" f may depend on only the *length* of the input.

▶ **Definition 2.5** ([8]). LWPP is the class of all sets A such that there exists a GapP function g and a function $f \in \mathrm{FP}$ that maps from 0^* to $\mathbb{Z} - \{0\}$ such that for all $x \in \Sigma^*$,

$$x \in A \Longrightarrow g(x) = f(0^{|x|})$$
$$x \notin A \Longrightarrow g(x) = 0.$$

Here, as usual, for any $x \in \Sigma^*$, $|x|$ denotes the length of x, and for any $n \in \mathbb{N}$, 0^n is the string consisting of exactly n zeroes.

We now generalize the definition of LWPP to the case of having the target of the GapP function be, for members of the set, not a single value but a collection of values.

One might expect us to formalize this by simply having the polynomial-time computable "what is the target" function output a *list* of the nonzero-integer targets. That would work fine and be equivalent to what we are about to do, as long as we are dealing with lists having at most a polynomial number of elements. However, to be able to speak of even longer lists – as will be important in our negative results offsetting our main result – we use an indexing approach, as follows.

▶ **Definition 2.6.** Let r be any function mapping from $\mathbb{N}$ to $\mathbb{N}$. Then the class r-LWPP is the class of all sets A such that there exists a GapP function g and a function $f \in \mathrm{FP}$ that maps to $\mathbb{Z} - \{0\}$ such that for each $x \in \Sigma^*$,

$$x \in A \Longrightarrow \text{ there exists } i \in \{1, 2, \ldots, r(|x|)\} \text{ such that } g(x) = f(\langle 0^{|x|}, i\rangle)$$
$$x \notin A \Longrightarrow g(x) = 0.$$

The following class, *Poly*-LWPP, will be central to this paper: Our main result is that this class in fact equals LWPP. The "$+\,c$" in Definition 2.7 may seem strange at first. But without it we would have a boundary-case pathology at $n = 0$, namely, the class could not contain any set that contains the empty string.[1]

▶ **Definition 2.7.**

$$\textit{Poly}\text{-LWPP} = \bigcup_{c \in \mathbb{N}^+} (n^c + c)\text{-LWPP}.$$

It is easy to see that 1-LWPP = LWPP, and that, of course, more flexibility as to targets never removes sets from the class, i.e., speaking loosely for the moment as to notation (and the log case will not be defined or used again in this paper, but it is clear from context here what we mean by it; the exponential case's definition can be found in the full version of the paper [12]), 1-LWPP $\subseteq$ 2-LWPP $\subseteq$ 3-LWPP $\subseteq \cdots \subseteq$ *Log*-LWPP $\subseteq$ *Poly*-LWPP $\subseteq$ *Exp*-LWPP. As mentioned above, in this paper we will prove that the first five of these "$\subseteq$"s are in fact all equalities. We will also prove that the sixth "$\subseteq$" cannot be an equality unless NP is PP-low.

We now show that for every function r, r-LWPP is contained in the co-class of the well-known counting class $\mathrm{C}_{=}\mathrm{P}$.

---

[1] The "$+\,c$" in Definition 2.7 also, on its surface, would seem to make a difference at length 1, by allowing lists of size greater than one; however, one could work around that issue. In contrast, the exclusion of the empty string would not be avoidable if our class of polynomials were to be a class – such as n^c – such that all of its members evaluate to 0 at $n = 0$. In any case, our use of $n^c + c$ avoids any special worries at lengths 0 and 1. And since for every polynomial p there is a c such that $(\forall n \in \mathbb{N})[p(n) \leq n^c + c]$, using polynomials just of the form $n^c + c$ is in fact not a restriction.

▶ **Definition 2.8** ([25, 29]). $\mathrm{C_{=}P}$ is the class of all sets A such that there is a nondeterministic polynomial-time Turing machine N and a function $f \in \mathrm{FP}$ such that for each $x \in \Sigma^*$,

$$x \in A \iff \mathrm{acc}_N(x) = f(x).$$

More convenient for us is the following characterization of $\mathrm{C_{=}P}$ using GapP functions.

▶ **Theorem 2.9** ([8]). *For each $A \subseteq \Sigma^*$, $A \in \mathrm{C_{=}P}$ if and only if there exists a function $g \in \mathrm{GapP}$ such that for all $x \in \Sigma^*$,*

$$x \in A \iff g(x) = 0.$$

We thus certainly have the following, which holds simply by taking the GapP function g required in Theorem 2.9 to be the same as the function g in Definition 2.6.

▶ **Theorem 2.10.** *For each function $r : \mathbb{N} \to \mathbb{N}$, r-LWPP $\subseteq \mathrm{coC_{=}P}$.*

3 Main Result: LWPP Stays the Same If for Accepted Inputs We Allow Polynomially Many Gap Values Instead of One

We now state our main result: LWPP altered to allow even a polynomial number of target gap values is still LWPP (i.e., with just one target gap value).

▶ **Theorem 3.1.** *Poly*-LWPP = LWPP.

For the proof, we need the following closure properties shown by Fenner, Fortnow, and Kurtz [8].[2] For function classes $\mathcal{F}_1$ and $\mathcal{F}_2$, $\mathcal{F}_1 \circ \mathcal{F}_2 = \{f_1 \circ f_2 \mid f_1 \in \mathcal{F}_1 \wedge f_2 \in \mathcal{F}_2\}$, where $\circ$ denotes composition.

▶ **Closure Property 3.2** ([8]). $\mathrm{GapP} \circ \mathrm{FP} = \mathrm{GapP}$ *and* $\mathrm{FP} \subseteq \mathrm{GapP}$.

▶ **Closure Property 3.3** ([8]). *If $g \in \mathrm{GapP}$ then $-g \in \mathrm{GapP}$.*

▶ **Closure Property 3.4** ([8]). *If $g \in \mathrm{GapP}$ and q is a polynomial, then the function*

$$h(x) = \prod_{0 \le i \le q(|x|)} g(\langle x, i \rangle)$$

is in GapP.

▶ **Closure Property 3.5** ([8]). GapP *is closed under addition, subtraction, and multiplication.*

Proof of Theorem 3.1. As mentioned previously, it is easy to see that LWPP $\subseteq$ *Poly*-LWPP.

To show *Poly*-LWPP $\subseteq$ LWPP, let A be a set in *Poly*-LWPP defined by $g \in \mathrm{GapP}$, $f \in \mathrm{FP}$, and polynomial $r(n) = n^c + c$ according to Definitions 2.6 and 2.7.

Let h_1 be a function such that for all $x \in \Sigma^*$ and $i \in \mathbb{N}^+$,

$$h_1(\langle x, i \rangle) = f(\langle 0^{|x|}, i \rangle) - g(x).$$

[2] We mention in passing that, regarding Closure Property 3.4, if the polynomial q were allowed to have coefficients that are uncomputable – or that are extremely expensive to compute prefixes of the values of – real numbers, that claimed closure property might fail; we here are, as is typical in such settings, tacitly assuming that the polynomials have rational coefficients.

We have $h_1 \in \text{GapP}$ since $f \in \text{FP} \subseteq \text{GapP}$, $g \in \text{GapP}$, and GapP is closed under subtraction [8]. We define h_2 such that for all $x \in \Sigma^*$,

$$h_2(x) = \prod_{1 \leq i \leq r(|x|)} h_1(\langle x, i \rangle).$$

By Closure Property 3.4, $h_2 \in \text{GapP}$. Note that for all $x \in \Sigma^*$,

$$h_2(x) = \prod_{1 \leq i \leq r(|x|)} \left(f(\langle 0^{|x|}, i \rangle) - g(x) \right).$$

It follows that for every $x \in \Sigma^*$,

$$h_2(x) = \begin{cases} 0 & \text{if there exists } i \in \{1, 2, \ldots, r(|x|)\} \\ & \quad \text{such that } g(x) = f(\langle 0^{|x|}, i \rangle) \\ \prod_{1 \leq i \leq r(|x|)} f(\langle 0^{|x|}, i \rangle) & \text{if } g(x) = 0. \end{cases} \tag{1}$$

Now we define the function $\widehat{g}$ such that for all $x \in \Sigma^*$,

$$\widehat{g}(x) = h_2(x) - \prod_{1 \leq i \leq r(|x|)} f(\langle 0^{|x|}, i \rangle).$$

Using the closure properties, it is easy to see that $\widehat{g} \in \text{GapP}$.

Note that by Eqn. (1), we have that for all $x \in \Sigma^*$,

$$\widehat{g}(x) = \begin{cases} -\prod_{1 \leq i \leq r(|x|)} f(\langle 0^{|x|}, i \rangle) & \text{if there exists } i \in \{1, 2, \ldots, r(|x|)\} \\ & \quad \text{such that } g(x) = f(\langle 0^{|x|}, i \rangle) \\ 0 & \text{if } g(x) = 0. \end{cases} \tag{2}$$

Let $\widehat{f}$ be a function such that for all $\ell \in \mathbb{N}$,

$$\widehat{f}(0^\ell) = -\prod_{1 \leq i \leq r(\ell)} f(\langle 0^\ell, i \rangle).$$

It is easy to see that $\widehat{f} \in \text{FP}$. Keeping in mind Eqn. (2), it follows from the above that for every $x \in \Sigma^*$,

$$\widehat{g}(x) = \begin{cases} \widehat{f}(0^{|x|}) & \text{if } x \in A \\ 0 & \text{otherwise.} \end{cases}$$

Since $\widehat{g} \in \text{GapP}$ and $\widehat{f} \in \text{FP}$, this implies that $A \in \text{LWPP}$. ◀

A theorem analogous to Theorem 3.1 also holds for the corresponding class that allows the target values to depend on the actual input instead on only the length of the input.

▶ **Definition 3.6.**

$$\textit{Poly}\text{-WPP} = \bigcup_{c \in \mathbb{N}^+} (n^c + c)\text{-WPP}.$$

▶ **Theorem 3.7.** $\textit{Poly}\text{-WPP} = \text{WPP}$.

The proof is almost exactly the same as the proof of Theorem 3.1, simply taking into account the fact that for WPP the "gap" function can vary even among inputs of the same length.

We remark that the robustness results stated in Theorems 3.1 and 3.7 do not seem to in any obvious way follow as corollaries to the closure of LWPP under polynomial-time Turing reductions [8] or of WPP under polynomial-time truth-table reductions [26].

Let us now see whether we can extend Theorems 3.1 and 3.7. Suppose we not only allow the target-gap set for acceptance to have polynomially many values, but in addition allow the target-gap set for rejection to have polynomially many values. (In contrast, both LWPP and r-LWPP allow rejection only when the gap's value is 0). Is LWPP so robust that even *that* class is no larger than LWPP? We will now build on our main result to give the answer "yes" to that question, thus extending our main result to this more symmetric case for LWPP and for WPP.

To this end, let us define (r_A, r_R)-LWPP as follows. (One of course can in the clear, analogous way define a similarly loosened version of WPP, (r_A, r_R)-WPP.)

▶ **Definition 3.8.** Let r_A and r_R be any functions mapping from $\mathbb{N}$ to $\mathbb{N}$. Then (r_A, r_R)-LWPP is the class of all sets B such that there exists a GapP function g and functions – each mapping to $\mathbb{Z}$ – $f_A \in \mathrm{FP}$ and $f_R \in \mathrm{FP}$ such that both of the following hold:

1. For each $j \in \mathbb{N}$, $A_j \cap R_j = \emptyset$, where

$$A_j = \{n \mid (\exists i \in \{1, 2, \ldots, r_A(j)\})[f_A(\langle 0^j, i\rangle) = n]\}$$

and

$$R_j = \{n \mid (\exists i \in \{1, 2, \ldots, r_R(j)\})[f_R(\langle 0^j, i\rangle) = n]\}.$$

2. For each $x \in \Sigma^*$,

$$x \in B \Longrightarrow g(x) \in A_{|x|}$$
$$x \notin B \Longrightarrow g(x) \in R_{|x|}.$$

It is not hard to see, via shifting gaps with dummy paths, that the class $(r, 1)$-LWPP equals r-LWPP. However, what can be shown more generally about the classes (r_A, r_R)-LWPP? For example, for which functions $r_{1,A}$, $r_{1,R}$, $r_{2,A}$, and $r_{2,A}$ does it hold that $(r_{1,A}, r_{1,R})$-LWPP $\subseteq$ $(r_{2,A}, r_{2,R})$-LWPP? (There are some literature notions that, in different ways, have at least a somewhat similar flavor to our notion, namely, defining classes by being more flexible regarding acceptance types. In particular, the counting classes CP_S of Cai et al. [4] and the "C-class" framework of Bovet, Crescenzi, and Silvestri [3] have such a flavor. But in contrast with those, our classes here are ones whose definitions are centered on the notion of gaps.) We do not here undertake that general study, but instead resolve what seems the most compelling question, namely, we prove that $(Poly, Poly)$-LWPP = LWPP.

▶ **Definition 3.9.** $(Poly, Poly)$-LWPP $= \bigcup_{c \in \mathbb{N}^+} (n^c + c, n^c + c)$-LWPP.

▶ **Theorem 3.10.** $(Poly, Poly)$-LWPP $= Poly$-LWPP.

The proof of Theorem 3.10 can be found in the full version of the paper [12]. Together with Theorem 3.1, we get the following corollary.

▶ **Corollary 3.11.** $(Poly, Poly)$-LWPP = LWPP.

An analogous statement can also be shown for the analogously defined class $(Poly, Poly)$-WPP:

▶ **Theorem 3.12.** $(Poly, Poly)$-WPP = WPP.

4 Applying the Main Result to Graph Reconstruction

▶ **Definition 4.1.** Let $\langle G_1, G_2, \ldots, G_n\rangle$ be a sequence of graphs and $G = (V, E)$ a graph with $V = \{1, 2, \ldots, n\}$. Suppose that there is a permutation $\pi \in S_n$ such that for each $k \in \{1, 2, \ldots, n\}$, the graph $G_{\pi(k)}$ is isomorphic to the graph $(V - \{k\}, E - \{\{k, \ell\} : \ell \in V\})$ obtained by deleting vertex k from G. Then $\langle G_1, G_2, \ldots, G_n\rangle$ is called a *deck* of G and G is called a *preimage* of the sequence $\langle G_1, G_2, \ldots, G_n\rangle$.

A sequence of graphs $\langle G_1, G_2, \ldots, G_n\rangle$ is called a *legitimate deck* if there exists a graph G that is a preimage of $\langle G_1, G_2, \ldots, G_n\rangle$.

The *Reconstruction Conjecture* (see, e.g., the surveys [2, 21, 1] and the book [18]) says that each legitimate deck consisting of graphs with at least two vertices has exactly one preimage up to isomorphism. This conjecture is a very prominent conjecture in graph theory – as mentioned in Section 1 it is perhaps the most important conjecture in that area – and has been studied for many decades.

Nash-Williams [22], Mansfield [20], Kratsch and Hemachandra [17], and Hemaspaandra et al. [11] introduced various decision problems related to the Reconstruction Conjecture, as part of a stream of work studying the algorithmic and complexity issues of reconstruction. We here are interested mainly in the Legitimate Deck Problem, which is defined as the following decision problem.

Legitimate Deck (a.k.a. the Legitimate Deck Problem) [20]: Given a sequence of graphs $\langle G_1, G_2, \ldots, G_n\rangle$, is $\langle G_1, G_2, \ldots, G_n\rangle$ a legitimate deck?

Mansfield [20] showed that the Graph Isomorphism Problem, GI (given two graphs G_1 and G_2, are they isomorphic?), is polynomial-time many-one reducible to the Legitimate Deck Problem. However, to this day it remains open whether there is a polynomial-time many-one reduction from the Legitimate Deck Problem to GI.

So how hard is the Legitimate Deck Problem? It is easy to see that the Legitimate Deck Problem is in NP. In the following, we will see that there is some evidence that the Legitimate Deck Problem is not NP-hard, and we will improve that evidence.

Let us define the following function problem.

Preimage Counting [16, 17]: Given a sequence of graphs $\langle G_1, G_2, \ldots, G_n\rangle$, compute the number $\mathrm{PCount}(\langle G_1, G_2, \ldots, G_n\rangle)$ of all nonisomorphic preimages for the sequence $\langle G_1, G_2, \ldots, G_n\rangle$.

Köbler, Schöning, and Torán [15] showed the following theorem.

▶ **Theorem 4.2** ([15]). *There is a function $h : 0^* \to \mathbb{N} - \{0\}$ in* FP *such that the function that maps every sequence $\langle G_1, G_2, \ldots, G_n\rangle$ to $h(0^n)$ times the number of nonisomorphic preimages, i.e.,*

$$\langle G_1, G_2, \ldots, G_n\rangle \mapsto h(0^n) \cdot \mathrm{PCount}(\langle G_1, G_2, \ldots, G_n\rangle)$$

is in GapP.

Theorem 4.2 has the following corollary.

▶ **Corollary 4.3** ([15]). *If the Reconstruction Conjecture holds, then the Legitimate Deck Problem is in* LWPP.

Since all LWPP sets are PP-low, that immediately gives some evidence that the Legitimate Deck Problem is not NP-hard.

▶ **Corollary 4.4.** *If the Reconstruction Conjecture holds, then the Legitimate Deck Problem is not* NP*-hard (or even* NP*-Turing-hard) unless* NP *is* PP*-low.*

Unfortunately, we do not know whether the Reconstruction Conjecture holds. However, perhaps we can prove the membership of the Legitimate Deck Problem in LWPP under a *weaker assumption* than the Reconstruction Conjecture, for instance, what if the number of nonisomorphic preimages is not always 0 or 1 (as holds under the Reconstruction Conjecture), but rather is merely relatively small, e.g., some constant or some polynomial in the number of vertices (as must hold for each graph class having bounded minimum degree)? We will now use the results of the previous section to prove that this indeed is the case.

▶ **Conjecture 4.5** (q-Reconstruction Conjecture)**.** *For each legitimate deck there exist at most $q(n)$ nonisomorphic preimages, where n is the number of graphs in the deck.*

For any function r, we now define a complexity class r-$\widehat{\text{LWPP}}$. This class may not seem very natural, but we will see that it is very-well suited to helping us classify the problem Legitimate Deck. In some sense, it is a tool that we will use in our proof, and then will discard by noting that it in fact turns out to be a disguised version of r-LWPP.

▶ **Definition 4.6.** Let r be any function mapping from $\mathbb{N}$ to $\mathbb{N}$. Then the class r-$\widehat{\text{LWPP}}$ is the class of all sets A such that there exists a GapP function g, and a function $f \in \text{FP}$ that maps from 0^* to $\mathbb{Z} - \{0\}$, such that for each $x \in \Sigma^*$,

$$x \in A \implies \text{there exists } i \in \{1, 2, \ldots, r(|x|)\} \text{ such that } g(x) = i \cdot f(0^{|x|})$$
$$x \notin A \implies g(x) = 0.$$

▶ **Theorem 4.7.** *Let q be any nondecreasing function from $\mathbb{N}$ to $\mathbb{N}$. Then the following holds. If the q-Reconstruction Conjecture holds, then the Legitimate Deck Problem is in q-$\widehat{\text{LWPP}}$.*

Proof. Suppose that the q-Reconstruction Conjecture holds. Let $\langle G_1, G_2, \ldots, G_n\rangle$ be an input to the Legitimate Deck Problem. By our assumption, we have that

$$\langle G_1, G_2, \ldots, G_n\rangle \in \text{Legitimate Deck} \implies \text{PCount}(\langle G_1, G_2, \ldots, G_n\rangle) \in \{1, 2, \ldots, q(n)\},$$
and
$$\langle G_1, G_2, \ldots, G_n\rangle \notin \text{Legitimate Deck} \implies \text{PCount}(\langle G_1, G_2, \ldots, G_n\rangle) = 0.$$

Let $h \in \text{FP}$ be the function discussed in Theorem 4.2. Then the function g defined such that for every sequence $\langle G_1, G_2, \ldots, G_n\rangle$,

$$g(\langle G_1, G_2, \ldots, G_n\rangle) = h(0^n) \cdot \text{PCount}(\langle G_1, G_2, \ldots, G_n\rangle)$$

is in GapP. It follows that

$$\langle G_1, G_2, \ldots, G_n\rangle \in \text{Legitimate Deck} \implies g(\langle G_1, G_2, \ldots, G_n\rangle) = h(0^n) \cdot i \text{ for some } i \in \{1, 2, \ldots, q(n)\}, \text{ and}$$
$$\langle G_1, G_2, \ldots, G_n\rangle \notin \text{Legitimate Deck} \implies g(\langle G_1, G_2, \ldots, G_n\rangle) = 0.$$

This almost directly implies that Legitimate Deck $\in q$-$\widehat{\text{LWPP}}$. However, note that to satisfy Definition 4.6, function h must be a function *that depends only on the length of the input*

$\langle G_1, G_2, \ldots, G_n\rangle$. The problem here is that (depending on how exactly we decide to encode the graphs $G_1, G_2, \ldots, G_n$ in the input $\langle G_1, G_2, \ldots, G_n\rangle$) even the value n (the number of graphs in the deck) may depend on the actual input $\langle G_1, G_2, \ldots, G_n\rangle$ and not only on the length of the input $\langle G_1, G_2, \ldots, G_n\rangle$.

Fortunately, there is a way to get around this problem. From the length of the input, we get at least an upper bound for n since we can certainly assume that $n \leq |\langle G_1, G_2, \ldots, G_n\rangle|$. Define a function $\widehat{h}$ such that for each $m \in \mathbb{N}$, $\widehat{h}(0^m)$ is the product of all "h-values" *up to length m*. That is, define function $\widehat{h}$ such that for all $m \in \mathbb{N}$,

$$\widehat{h}(0^m) = \prod_{0 \leq i \leq m} h(0^i).$$

Define function h' such that for all inputs $\langle G_1, G_2, \ldots, G_n\rangle$,

$$h'(\langle G_1, G_2, \ldots, G_n\rangle) = \prod_{0 \leq i \leq |\langle G_1, G_2, \ldots, G_n\rangle| \wedge i \neq n} h(0^i).$$

Now we can see that for all inputs $\langle G_1, G_2, \ldots, G_n\rangle$,

$$g(\langle G_1, G_2, \ldots, G_n\rangle) \cdot h'(\langle G_1, G_2, \ldots, G_n\rangle) = \begin{cases} 0 & \text{if } \langle G_1, G_2, \ldots, G_n\rangle \notin \text{Legitimate Deck} \\ i \cdot \widehat{h}(0^{|\langle G_1, G_2, \ldots, G_n\rangle|}) \text{ for some } i \in \{1, 2, \ldots, q(n)\} & \text{if } \langle G_1, G_2, \ldots, G_n\rangle \in \text{Legitimate Deck.} \end{cases} \tag{3}$$

Note that $\widehat{h}, h' \in$ FP and $g \in$ GapP. By Closure Properties 3.2 and 3.5, the function $x \mapsto g(x) \cdot h'(x)$ is in GapP. Finally, note that since q is nondecreasing and $n \leq |\langle G_1, G_2, \ldots, G_n\rangle|$, we have that $q(n) \leq q(|\langle G_1, G_2, \ldots, G_n\rangle|)$ in Eqn. (3). Thus by Definition 4.6 – with the GapP function g there being our $g(x) \cdot h'(x)$ and the function f there being our $\widehat{h}$ – we have that Legitimate Deck $\in q$-$\widehat{\text{LWPP}}$. ◀

We want to show that if there exists a polynomial q such that the q-Reconstruction Conjecture holds, then Legitimate Deck is in LWPP. To this end, we need the following inclusion.

▶ **Theorem 4.8.** *For each function $r : \mathbb{N} \to \mathbb{N}$, r-$\widehat{\text{LWPP}} \subseteq r$-LWPP.*

Proof. Let A be a set in r-$\widehat{\text{LWPP}}$ via $f_1 \in$ FP and $g \in$ GapP. Let $f_2 \in$ FP be the function defined such that for every $n, i \in \mathbb{N}^+$, $f_2(\langle 0^n, i\rangle) = i \cdot f_1(0^n)$. Then A is in r-LWPP via $f_2 \in$ FP and $g \in$ GapP. ◀

▶ **Corollary 4.9.** *If the q-Reconstruction Conjecture holds for some polynomial q, then the Legitimate Deck Problem is in* LWPP.

Proof. Suppose the q-Reconstruction Conjecture holds for nondecreasing polynomial q. (If q is not nondecreasing but the q-Reconstruction Conjecture holds, then obviously we can replace q with a nondecreasing polynomial q' that on each input n is greater than or equal to $q(n)$ and the q'-Reconstruction Conjecture will hold. So we may w.l.o.g. take it that q is nondecreasing.) By Theorems 4.7 and 4.8, Legitimate Deck $\in q$-$\widehat{\text{LWPP}} \subseteq q$-LWPP and hence Legitimate Deck $\in$ *Poly*-LWPP. With Theorem 3.1, it follows that Legitimate Deck $\in$ LWPP. ◀

This gives us our new, more flexible – though still conditional – evidence that the Legitimate Deck Problem is not NP-hard.

▶ **Corollary 4.10.** *If the q-Reconstruction Conjecture holds for some polynomial q, then the Legitimate Deck Problem is not* NP*-hard (or even* NP*-Turing-hard) unless* NP *is* PP*-low.*

▶ **Definition 4.11** ([17])**.** Let $\mathcal{H}$ be any class of graphs. Then the *Legitimate Deck Problem restricted to* $\mathcal{H}$ consists of all sequences of graphs $\langle G_1, G_2, \ldots, G_n\rangle$ such that $\langle G_1, G_2, \ldots, G_n\rangle$ is a legitimate deck and for each $i \in \{1, 2, \ldots, n\}$, G_i is in $\mathcal{H}$.

Note that the above definition is so flexible that it allows even the case where the preimage(s) may not be in $\mathcal{H}$.

▶ **Theorem 4.12.** *Let $\mathcal{H}$ be any P-recognizable class of graphs such that decks consisting only of graphs in $\mathcal{H}$ have a number of nonisomorphic preimages that is bounded polynomially in the number of graphs in the deck. Then the Legitimate Deck Problem restricted to $\mathcal{H}$ is in* LWPP*.*

Proof. Let $\mathcal{H}$ be any P-recognizable class of graphs and q a nondecreasing polynomial such that decks consisting only of graphs in $\mathcal{H}$ have a number of nonisomorphic preimages that is bounded by $q(n)$, where n is the number of graphs in the deck.

First, we show that the Legitimate Deck Problem restricted to $\mathcal{H}$ is in q-$\widehat{\text{LWPP}}$. Let $\langle G_1, G_2, \ldots, G_n\rangle$ be an input to the Legitimate Deck Problem. Check if for every $i \in \{1, 2, \ldots, n\}$, G_i is in $\mathcal{H}$. If this is not the case then reject in the sense of q-$\widehat{\text{LWPP}}$, i.e., produce a gap of zero. Otherwise, the deck $\langle G_1, G_2, \ldots, G_n\rangle$ has at most $q(n)$ preimages, i.e., $\text{PCount}(\langle G_1, G_2, \ldots, G_n\rangle \leq q(n)$. Proceed as in the proof of Theorem 4.7.

Since by Theorems 4.8 and 3.1, q-$\widehat{\text{LWPP}} \subseteq q$-LWPP $\subseteq$ LWPP, it follows that the Legitimate Deck Problem restricted to $\mathcal{H}$ is in LWPP. ◀

As usual, for each graph G, $\delta(G)$ denotes the degree of a minimum-degree vertex of G.

▶ **Theorem 4.13.** *For each $k \in \mathbb{N}^+$, let*

$$\mathcal{H}_k = \{G \mid G \text{ is a graph such that } \delta(G) \leq k\}.$$

Then the Legitimate Deck Problem restricted to $\mathcal{H}_k$ is in LWPP*.*

Proof. In the proof of their Theorem 6.1, Kratsch and Hemaspaandra [17] showed that for each class of graphs with bounded minimum degree, the number of nonisomorphic preimages is polynomially bounded. Now the theorem follows from Theorem 4.12. ◀

It is interesting to note that for each of the $\mathcal{H}_k$ classes $\text{GI}_{\mathcal{H}_k} \equiv_m^p$ GI trivially holds (for example, via adding to each of the two graphs being tested for isomorphism an isolated node), notwithstanding the fact that intersection with $\mathcal{H}_k$ pulls the Legitimate Deck Problem's complexity into LWPP.

In two of this section's corollaries we used the fact that all LWPP sets are PP-low. We mention that since all LWPP set are also $\text{C}_=$P-low [15, 8], the altered versions of Corollaries 4.4 and 4.10 in which the conclusion is changed from "unless NP is PP-low" to "unless NP is $\text{C}_=$P-low" both hold, respectively due to Köbler, Schöning, and Torán [15] and the present paper.

5 Optimality of the Main Result (Brief Version)

It is easy to see that the proof of Theorem 3.1 breaks down if *Poly*-LWPP is replaced by the analogous class where the size of the set of allowed gap values can be larger than polynomial in the input length. This does not necessarily imply that the corresponding theorem does not hold. However, in this section, we establish that relativizable proof techniques are not sufficient to improve Theorem 3.1 from *Poly*-LWPP to r-LWPP for any function r that is not polynomially bounded. That is, we show that our main result is optimal with respect to what can be proven by relativizable proof techniques.

▶ **Theorem 5.1.** *Let r be any function from $\mathbb{N}$ to $\mathbb{N}$ such that for every $c \in \mathbb{N}$, $r \notin \mathcal{O}(n^c)$. Then there exists an oracle $\mathcal{O}$ such that r-LWPP$^{\mathcal{O}} \not\subseteq$ LWPP$^{\mathcal{O}}$.*

To prove Theorem 5.1, we will encode nondeterministic oracle Turing machines by low-degree multivariate polynomials. This technique is apparently folklore and has been used, for example, by de Graaf and Valiant [6] to construct a relativized world where the quantum complexity class EQP is not contained in the modularity-based complexity class MOD$_{p^k}$P. The general technique of replacing oracle machines by simpler combinatorial objects such as circuits, decision trees, or polynomials and then using properties of such combinatorial objects to show the existence of a desired oracle dates to the seminal work of Furst, Saxe, and Sipser [9], who made the connection between circuit lower bounds and the relativization of the polynomial hierarchy – a connection that has led to the resolution of many previously long-open relativized questions, such as the achievement of an oracle making the polynomial hierarchy infinite [31, 10] and of oracles making the polynomial hierarchy extend exactly k levels [14].

Note: Please see [12] for a complete version of this section, which includes the proof of Theorem 5.1 and additional results showing that LWPP *with an exponential number of target gap values cannot equal* LWPP *without causing highly unlikely structural consequences such as* PP$^{\text{NP}}$ = PP*.*

6 LWPP$^+$ (Brief Version)

Theorem 3.1 of Section 3 established a robustness property of LWPP, namely, that *Poly*-LWPP= LWPP. That is, having one target value for acceptance and having a list of target values for acceptance yield the same class of languages, in the content of LWPP, which, recall, is defined in terms of the values of GapP functions. That robustness result is itself robust in the sense that it holds both in the real world and, it is easy to see, in every relativized world.

On the other hand, this equivalence for LWPP, in terms of descriptive richness, between one target value and a polynomial number of values, may not hold even for quite similar counting-class situations. In particular, in this section we prove that in some relativized worlds, for the analog (which we will call LWPP$^+$) of LWPP defined in terms of #P rather than GapP functions (following Cox and Pay [5], who recently introduced the #P-based analogue of WPP), having even two target values for acceptance (the class Two-LWPP$^+$) yields a richer class of languages than having one value. So it is not the case that single targets and lists of targets inherently function identically as to descriptive richness for counting classes.

▶ **Theorem 6.1.** *There exists an oracle $\mathcal{O}$ such that* (LWPP$^+$)$^{\mathcal{O}} \subsetneq$ (Two-LWPP$^+$)$^{\mathcal{O}}$*.*

Note: Please see [12] for a complete version of this section, which includes definitions and the proof of our relativized result (Theorem 6.1) showing that, for the #P*-based analogue*

of LWPP*, even two target values may yield a larger class of languages than does one target value, in contrast to what holds for the* LWPP *case.*

7 Conclusions and Open Questions

In this paper, we proved that LWPP and WPP are robust enough that they remain unchanged when their single target gap is allowed to be expanded to polynomial-sized lists. We then applied this new robustness of LWPP to show that the PP-lowness of the Legitimate Deck Problem follows from a weaker hypothesis than was previously known. In doing so, we provided enhanced evidence that the Legitimate Deck Problem is not NP-hard or even NP-Turing-hard. We also showed: that the polynomial-target robustness of LWPP that we have established is optimal (i.e., cannot be extended to any superpolynomial number of targets) with respect to relativizable proofs; that for the #P-based analogue of the (GapP-based) class LWPP, in some relativized worlds even two targets give more languages than one target; that our robustness of LWPP holds even when one simultaneously expands both the acceptance target-gap set and the rejection target-gap set to be polynomial-sized lists; and that our main results also hold for WPP.

Regarding the Reconstruction Conjecture, we proved as a consequence of our results that if there exists a polynomial q such that the q-Reconstruction Conjecture holds, then the Legitimate Deck Problem is both PP-low and $C_=P$-low. Since NP is widely believed not to be PP-low or $C_=P$-low, this provides strengthened evidence that the Legitimate Deck Problem is not NP-hard or even NP-Turing hard (since otherwise even the "Poly"-Reconstruction Conjecture must fail, yet even the far stronger Reconstruction Conjecture is generally believed to hold).

A natural open problem is whether the Legitimate Deck Problem is Σ_k^p-low [24] for some k, i.e., whether for some k it holds that $(\Sigma_k^p)^{\text{Legitimate Deck}} = \Sigma_k^p$. Proving that that holds – though we mention that this has been a known open issue for more than two decades (see [17]) – would imply that the Legitimate Deck Problem cannot be NP-complete (even with respect to more flexible reductions such as Turing reductions and strong nondeterministic reductions [19]) unless the polynomial hierarchy collapses.

References

1 J. Bondy. A graph reconstructor's manual. In *Surveys in Combinatorics*, London Mathematical Society Lecture Notes Series 66, pages 221–252. Cambridge University Press, 1991.

2 J. Bondy and R. Hemminger. Graph reconstruction—a survey. *Journal of Graph Theory*, 1:227–268, 1977.

3 D. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoretical Computer Science*, 104(2):263–283, 1992.

4 J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy II: Applications. *SIAM Journal on Computing*, 18(1):95–111, 1989.

5 J. Cox and T. Pay. An overview of some semantic and syntactic complexity classes. Technical Report arXiv:1806.03501 [cs.CC], ArXiv.org, June 2018.

6 M. de Graaf and P. Valiant. Comparing EQP and MOD_{p^k}P using polynomial degree lower bounds. Technical Report quant-ph/0211179, Quantum Physics, 2002.

7 The Editors (of the Journal of Graph Theory). Editorial note. *Journal of Graph Theory*, 1(3), 1977.

8 S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994.

9 M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984.

10 J. Håstad. *Computational Limitations of Small-Depth Circuits*. MIT Press, 1987.

11 E. Hemaspaandra, L. Hemaspaandra, S. Radziszowski, and R. Tripathi. Complexity results in graph reconstruction. *Discrete Applied Mathematics*, 155(2):103–118, 2007.

12 E. Hemaspaandra, L. Hemaspaandra, H. Spakowski, and O. Watanabe. The robustness of LWPP and WPP, with an application to graph reconstruction. Technical Report arXiv:1711.01250v2 [cs.CC], ArXiv.org, November 2017. Revised, April 2018.

13 P. Kelly. *On Isometric Transformations*. PhD thesis, University of Wisconsin, USA, 1942.

14 K. Ko. Relativized polynomial-time hierarchies having exactly k levels. *SIAM Journal on Computing*, 18(2):392–408, 1989.

15 J. Köbler, U. Schöning, and J. Torán. Graph isomorphism is low for PP. *Computational Complexity*, 2:301–330, 1992.

16 D. Kratsch and L. Hemachandra. On the complexity of graph reconstruction. In *Proceedings of the 8th Conference on Fundamentals of Computation Theory*, pages 318–328. Springer-Verlag *Lecture Notes in Computer Science #529*, 1991.

17 D. Kratsch and L. Hemaspaandra. On the complexity of graph reconstruction. *Mathematical Systems Theory*, 27(3):257–273, 1994.

18 J. Lauri and R. Scapellato. *Topics in Graph Automorphisms and Reconstruction*. Cambridge University Press, 2003.

19 T. Long. Strong nondeterministic polynomial-time reducibilities. *Theoretical Computer Science*, 21:1–25, 1982.

20 A. Mansfield. The relationship between the computational complexities of the legitimate deck and isomorphism problems. *Quart. J. Math. Ser.*, 33(2):345–347, 1982.

21 B. Manvel. Reconstruction of graphs: Progress and prospects. *Congressus Numerantium*, 63:177–187, 1988.

22 C. St. J. A. Nash-Williams. The reconstruction problem. In L. Beineke and R. Wilson, editors, *Selected Topics in Graph Theory*, pages 205–236. Academic Press, 1978.

23 M. Ogiwara and L. Hemachandra. A complexity theory for feasible closure properties. *Journal of Computer and System Sciences*, 46(3):295–325, 1993.

24 U. Schöning. A low and a high hierarchy within NP. *Journal of Computer and System Sciences*, 27:14–28, 1983.

25 J. Simon. *On Some Central Problems in Computational Complexity*. PhD thesis, Cornell University, Ithaca, N.Y., 1975. Available as Cornell Department of Computer Science Technical Report TR75-224.

26 H. Spakowski, M. Thakur, and R. Tripathi. Quantum and classical complexity classes: Separations, collapses, and closure properties. *Information and Computation*, 200(1):1–34, 2005.

27 L. Stockmeyer and A. Meyer. Word problems requiring exponential time. In *Proceedings of the 5th ACM Symposium on Theory of Computing*, pages 1–9. ACM Press, 1973.

28 S. Ulam. *A Collection of Mathematical Problems*. Interscience Publishers, New York, 1960.

29 K. Wagner. The complexity of combinatorial problems with succinct input representations. *Acta Informatica*, 23(3):325–356, 1986.

30 C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3:23–33, 1977.

31 A. Yao. Separating the polynomial-time hierarchy by oracles. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, pages 1–10, 1985.

Shape Recognition by a Finite Automaton Robot

Robert Gmyr
Paderborn University, Germany
gmyr@mail.upb.de

Kristian Hinnenthal
Paderborn University, Germany
krijan@mail.upb.de

Irina Kostitsyna
TU Eindhoven, the Netherlands
i.kostitsyna@tue.nl

Fabian Kuhn
University of Freiburg, Germany
kuhn@cs.uni-freiburg.de

Dorian Rudolph
Paderborn University, Germany
dorian@mail.upb.de

Christian Scheideler
Paderborn University, Germany
scheideler@upb.de

Abstract

Motivated by the problem of shape recognition by nanoscale computing agents, we investigate the problem of detecting the geometric shape of a structure composed of hexagonal tiles by a finite-state automaton robot. In particular, in this paper we consider the question of recognizing whether the tiles are assembled into a parallelogram whose longer side has length $\ell = f(h)$, for a given function $f(\cdot)$, where h is the length of the shorter side. To determine the computational power of the finite-state automaton robot, we identify functions that can or cannot be decided when the robot is given a certain number of pebbles. We show that the robot can decide whether $\ell = ah + b$ for constant integers a and b without any pebbles, but cannot detect whether $\ell = f(h)$ for any function $f(x) = \omega(x)$. For a robot with a single pebble, we present an algorithm to decide whether $\ell = p(h)$ for a given polynomial $p(\cdot)$ of constant degree. We contrast this result by showing that, for any constant k, any function $f(x) = \omega(x^{6k+2})$ cannot be decided by a robot with k states and a single pebble. We further present exponential functions that can be decided using two pebbles. Finally, we present a family of functions $f_n(\cdot)$ such that the robot needs more than n pebbles to decide whether $\ell = f_n(h)$.

2012 ACM Subject Classification Theory of computation → Models of computation, Theory of computation → Design and analysis of algorithms

Keywords and phrases finite automata, shape recognition, computational geometry

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.52

Funding This work is partly supported by DFG grant SCHE 1592/3-1. Fabian Kuhn is supported by ERC Grant 336495 (ACDC).

Acknowledgements This work was begun at the Dagstuhl Seminar on Algorithmic Foundations of Programmable Matter, July 3–8, 2016. Preliminary results were presented at EuroCG 2018.

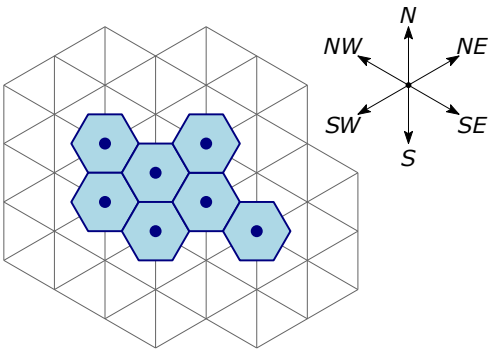


Figure 1 An exemplary tile configuration. The top right part of the figure shows the compass directions we use to describe the movement of a robot.

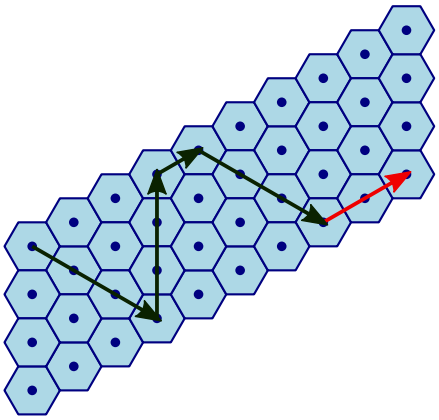


Figure 2 A N-NE-parallelogram with height 4 and length $10 = 2 \cdot 4 + 2$. The black arrows indicate the zig-zag movement of a robot as described in the proof of Theorem 2. The red arrow shows the final NE movement.

1 Introduction

The *DNA-based self-assembly* approach [20], in which large structures have to be assembled from DNA tiles in a self-organized fashion, is still quite error prone. Numerous techniques have been studied to reduce the error rates (e.g., [6]), but the research typically focuses on designing DNA tiles or assembly processes that reduce the error of incorrect attachments. Given the progress on designing so-called *DNA walkers* (see, e.g., [22, 24, 25]), it is foreseeable that also simple molecular robots could be used in order to make the assemblies more reliable. For example, such robots may be used to repair structures that did not self-assemble correctly. In order to be able to repair a given structure, it first has to be possible to detect whether a structure is not in a correct shape, which motivates the problem of *shape recognition*. Besides verifying self-assembled DNA structures, shape recognition could also be useful for minimal invasive surgery applications, such as looking for harmful substances or cells in a human body.

Naturally, molecular robots may have very limited capabilities, which is why we are focusing on robots with the computational power of a finite-state automaton. More precisely, we build upon the model introduced in [10] where finite-state automaton robots move on a structure composed of hexagonal tiles, represented as a subgraph of the infinite triangular lattice, and rearrange the tiles into a certain shape. Although shape *formation* with computationally restricted agents, as considered in [10], has been extensively studied in many other models (see, e.g., [5, 13, 17, 26]), to the best of our knowledge, the closely related problem of shape *detection* has never been explicitly studied in our setting.

1.1 Model

We assume that a single *robot* is placed on a finite set of *hexagonal tiles*. Each tile occupies exactly one node of the infinite triangular lattice $G = (V, E)$ (refer to Figure 1). We assume that the subgraph of G induced by all nodes occupied by tiles is connected. Every node $u \in V$ is adjacent to six neighbors, and, as indicated in the figure, we describe the relative positions of adjacent nodes by six compass directions.

Table 1 This table summarizes the results for recognizing whether a given parallelogram has height h and length $\ell = f(h)$ given a certain number of pebbles.

Pebbles	Possible	Impossible	Remarks	Refer to
0	$f(x) = ax + b$	$f(x) = \omega(x)$	a, b constant	Section 3.1
1	$f(x) = a_n x^n + \ldots + a_0$	$f(x) = \omega(x^{6k+2})$	n, a_i constant for all i, k is the number of the robot's states	Section 3.2
2	$f(x) = \underbrace{2^{2^{\cdot^{\cdot^{2^x}}}}}_{s+1}$	—	s constant	Section 3.3
n	$f_n(x)$	$f_{n+1}(x)$	—	Section 3.4

The robot is initially placed on a tile. It can move on the tile structure and carry a (possibly empty) set of *pebbles*, which can be placed on tiles in order to *mark* them. A tile can be marked by at most one pebble.

More specifically, the robot acts as a *deterministic finite automaton* and operates in *look-compute-move* cycles. In the *look* phase the robot can observe the node it occupies and the six neighbors of that node. For each of these nodes it can determine whether it is occupied by a tile and whether a pebble is placed on that tile. In the *compute* phase the robot potentially changes its state and determines its next move according to the observed information and the number of carried pebbles. In the *move* phase the robot can either take a pebble from its current node (if its tile is marked by a pebble), place a pebble it is carrying at that node (if its tile is not already marked and the robot carries at least one pebble), or move to an adjacent occupied node.

Note that even though we describe the algorithms as if the robot knew a global orientation, we do not actually require the robot to have a compass. For the algorithms presented in this paper, it is enough for the robot to be able to maintain its orientation with respect to its initial orientation.

1.2 Related Work

To the best of our knowledge, shape recognition has never been investigated in our model. However, solving problems by traversing a tile structure with simple agents has been studied in many different areas. For instance, [23] considers the problem of deciding whether a structure is simply-connected. Other problems include Gathering and Rendezvous (e.g., [21]), Intruder Caption and Graph Searching (e.g., [2,8]), or Black Hole Search (e.g., [16]).

For many of the above problems it has also been investigated whether pebbles can be helpful. This question is particularly well-studied for the classical Network Exploration Problem (see, e.g., [4]). For example, it is known that a finite automaton robot can neither explore all planar graphs [9] nor find its way out of a planar labyrinth [3] without any pebbles. For the Labyrinth Exploration Problem (see [14,15] for a comprehensive survey), it is known that having a single pebble does not help the robot [11]. However, a robot with two pebbles can solve the problem [1].

1.3 Our Contributions

In this paper we investigate the problem of *shape recognition* by a single robot. Specifically, we begin with testing whether a given tile formation is of a certain simple shape, in particular, a parallelogram. Then, we consider the problem of deciding for a given function $f(\cdot)$ whether the longer side of a parallelogram has length $f(h)$, where h is the shorter side's length. We particularly investigate this problem under the assumption that the robot is given a set of pebbles that can be used to mark certain positions on the tile structure. An overview of our results is given in Table 1, in which we give concrete functions $f(\cdot)$ the robot is able or not able to decide given a certain set of pebbles. Our ultimate goal is to investigate the computational capabilities of a simple robot concerning shape recognition, and to what extent the robot can benefit from employing pebbles.

As we do not make any assumptions on the length of the shorter side h of the parallelogram, we cannot assume that a robot with only a constant number of possible states is able to count up to h, even less so evaluate the function $f(h)$. Thus, if the robot does not have pebbles at its disposal, its only option is to make use of the environment's geometry. For example, the robot can 'measure' h tiles along the longer side of the parallelogram by starting in a corner and moving diagonally until reaching the opposite boundary. Furthermore, the robot can measure $2h$, $3h$, or even ah tiles, for any constant a, along the longer side.

In Section 3.1 we develop this intuition further, and show that the robot can decide whether the longest side of the parallelogram is $\ell = ah + b$, where a and b are constants. On the other hand, we show that the robot without pebbles is not able to recognize a superlinear function. In Sections 3.2 and 3.3 we show that we can tremendously increase the robot's computational power by giving it a single pebble or two pebbles, respectively. Having the ability to mark any tile with a single pebble allows the robot to recognize any polynomial function of constant degree; and being equipped with two pebbles gives the robot the ability to recognize power tower functions, where the height of the power tower is constant or even linear in h. Finally, in Section 3.4 we show that for any number of pebbles there exists a function that requires that many pebbles to be decided by the robot.

2 Recognizing Simple Shapes

First, observe that a single robot can easily detect whether the initial structure is a line, a triangle, a hexagon, or a parallelogram.

Line. For example, to test if a given tile shape is a line, the robot first chooses a direction in which there is a tile (say, w.l.o.g., N), walks in that direction as far as possible (i.e., until there is no tile in that direction anymore), and then traverses the structure into the opposite direction until no longer possible. If it ever encounters a tile to the left or right of any traversed tile, the structure is not a line.

Parallelogram. To test if a given tile shape is a (filled) parallelogram axis-aligned along the directions N and NE (see Figure 2), first, the robot moves to a locally southernmost tile of the structure by moving S and SW as long as there is a tile in any of these directions. It then traverses the shape column by column in a snake-like fashion by repeating the following movements: First, the robot moves N as far as possible; it then moves one step NE; then it moves S as far as possible, and it finally moves one step NE. The above procedure is repeated until a NE movement is impossible. By performing local checks alongside the movements described above the robot can verify whether the tile shape is a parallelogram.

The other simple shapes can be easily tested in a similar fashion.

▶ **Observation 1.** *A robot without any pebble can detect whether the initial tile configuration is a line, a triangle, a hexagon, or a parallelogram.*

3 Recognizing Parallelograms with Specific Side Ratio

As noted in Observation 1, a single robot without pebbles can verify whether a given shape is a parallelogram. To investigate the computational power of a finite automaton, in this section we consider the problem of deciding whether a parallelogram has a given side ratio. Additionally, we examine how pebbles can be helpful to decide more complex side ratios.

We assume w.l.o.g. that the robot needs to detect whether the given tile configuration is a parallelogram that is axis-aligned along the north and north-east direction, as indicated in Figure 2. We denote a maximal sequence of consecutive tiles from N to S as a *column* and a maximal sequence of consecutive tiles from SW to NE as a *row*. Let h be the size of each column, i.e., the parallelogram's *height*, ℓ be the size of each row, i.e., the parallelogram's *length*, and let $h \leq \ell$. We number the columns of the parallelogram from 0 to $\ell - 1$ growing in the north-eastern direction.

3.1 A Robot without any Pebble

First, we point out that a single robot can detect whether the structure is a parallelogram in which its length ℓ is a linear function of its height h.

▶ **Theorem 2.** *A single robot can detect whether the tile configuration is a parallelogram with $\ell = ah + b$ for any constants $a, b \in \mathbb{N}$.*

Proof. First, the robot verifies whether the structure is a parallelogram. If so, the robot moves to the northernmost tile of column 0. It then traverses the tile structure in two stages to verify the ratio of the sides. In the first stage, the robot "measures" the distance ah along the length of the parallelogram moving in a zig-zag fashion as depicted in Figure 2. In the second stage the robot measures the second term b. More specifically, in the first stage, the robot repeats the following movements in a loop: (1) move SE as far as possible, (2) move N as far as possible, and (3) make one step NE. After having performed the complete sequence of SE movements a times, the robot moves on to the second stage, in which it makes an additional b NE steps.

If the robot reaches the easternmost column before completing the above procedure, or finally halts on a tile with a neighboring tile at NE, it terminates with a negative result. Otherwise, it terminates with a positive result. It is easy to see that $\ell = ah + b$ if and only if the robot terminates with a positive result. ◀

▶ Remark. The algorithm in the previous theorem can be adapted for $b \in \mathbb{Z}$, i.e., b can also be a negative integer. To achieve that, we halt the execution of the first stage once the robot has performed $|b|$ SE steps, then move SW as far as possible, and continue the first stage from there. Then, $\ell = ah + b$ if and only if the robot eventually reaches the southernmost tile of column $\ell - 1$.

▶ Remark. The algorithm can be further extended to apply in the case of a rational a. Let $a = p/q$ be an irreducible fraction. Instead of moving in a zig-zag fashion in the first stage, the robot alternates between moving p steps NE and q steps S. To exactly end up at the southernmost tile of column $\ell - 1$, the robot needs to skip the very first NE and S step.

We have shown that a single robot can determine whether the length of a parallelogram is given by a certain linear function of its height. However, that is as much as one robot can hope for. Indeed, a single robot is not able to decide whether the length of the parallelogram is given by a superlinear function of its height, as the following theorem states.

▶ **Theorem 3.** *A single robot without any pebbles cannot decide whether the tile configuration is a parallelogram with $\ell = f(h)$, where $f(x) = \omega(x)$.*

Proof. Suppose there is an algorithm that lets the robot decide whether the tiles are arranged into a parallelogram with $\ell = f(h)$ for some superlinear function $f(x) = \omega(x)$. Let k be the total number of states used by the robot in the algorithm. Choose h large enough such that $\lfloor \frac{f(h)-2}{h} \rfloor > k$, which can be done since $f(h) = \omega(h)$.

Consider the execution of the algorithm on a parallelogram P with height h and length $\ell = f(h)$. We will show that there exists another parallelogram with height h and length greater than $f(h)$ on which the robot will eventually terminate in exactly the same state as on P, which contradicts the assumption that the algorithm is correct.

We first place the robot on the northernmost tile of column 0. First, observe that if the robot does not visit column $\ell - 1$ during the execution of the algorithm, it cannot correctly detect the length of the parallelogram. Thus, the robot visits this column at least once.

Consider the execution of the algorithm as a sequence $(p_1, p_2, \dots)$ of tuples of nodes and states $p_i = (u_i, q_i)$. Denote as $\pi_1, \dots, \pi_m$ the subsequences of the execution of the algorithm, where each subsequence starts whenever the robot leaves a node of column 0 or $\ell - 1$, and ends when it enters a node of one of those columns. More specifically, the first and last node of each π_i is a node adjacent to column 0 or $\ell - 1$ and the node immediately before and after each π_i is a node of 0 or $\ell - 1$. Note that in each π_i the robot exclusively moves in the columns between column 0 and $\ell - 1$.

First, consider a subsequence π_i at the beginning of which the robot leaves, and at the end of which, enters the same column 0 (or $\ell - 1$). Let $p_i = (s_i, q_i)$ be the node-state tuple right before the robot enters the subsequence π_i when executing the algorithm on the parallelogram P. Then, the robot would execute exactly the same subsequence π_i on a parallelogram P' with a larger length than the one of P, if it were placed on a node of P' corresponding to s_i with the same state q_i.

Next, consider a subsequence π_j at the beginning of which the robot leaves, w.l.o.g., column 0, and at the end of which it enters $\ell - 1$. Since the robot completely traverses the tile structure from column 0 to $\ell - 1$, there must be a row R_j in which the robot steps on no less than $\lfloor \frac{f(h)-2}{h} \rfloor > k$ tiles. Therefore, there will be two nodes u_j and v_j in the row R_j in which the robot appears in the same state. Let c_u, c_v be the column indices of u_j and v_j, respectively, and define $d_j = |c_v - c_u|$. Let (s_j, q_j) be the node-state tuple of the robot immediately before it enters the subsequence π_j and (s'_j, q'_j) be the node-state tuple of the robot immediately after the subsequence π_j is finished. Then, consider a parallelogram Q with height h and length $f(h) + cd_j$, where $c \in \mathbb{N}_0$. If the robot starts in the same state q_j on the node corresponding to s_j (i.e., the node in column 0 and the row of s_j) in the parallelogram Q, it will move entirely between the westernmost and easternmost column of Q until it reaches the node corresponding to s'_j (i.e., the node of column $f(h) + cd_j$ and the row of s'_j) in Q in state q'_j.

Now consider the execution of the algorithm on a parallelogram P' with height h and length $f(h) + \prod_j d_j$ for each subsequence π_j where the robot completely traverses the parallelogram between column 0 and $\ell - 1$. By the above argument, the robot will enter and leave those columns on the same tile and in the same state as in the execution of the algorithm on P. Thus, executing the same algorithm on the parallelogram P' the robot will ultimately terminate in the same state as if it were on the parallelogram P. Therefore, the robot cannot decide whether the initial tile configuration is a parallelogram with $\ell = f(h)$ for any superlinear function $f(x) = \omega(x)$. ◀

3.2 A Robot with a Single Pebble

In the following, we demonstrate that, in contrast to the negative result of Theorem 3, a single robot can decide any polynomial of constant degree.

▶ **Theorem 4.** *A single robot with a pebble can decide whether the tile configuration is a parallelogram of height h and length $\ell = p(h)$ for any given polynomial $p(\cdot)$ of constant degree n.*

Proof. Define the *falling factorial* of x as $(x)_i := x(x-1)\cdots(x-i+1)$, and transform the input polynomial into the form $p(x) = a_n \cdot (x)_n + a_{n-1} \cdot (x)_{n-1} + \ldots + a_0$. We will show that the robot can move the pebble in phases, by $|a_i \cdot (h)_i|$ steps in each phase i. Let $\mathrm{lcm}_i(x) := \mathrm{lcm}(x, \ldots, x-i+1)$, where lcm is the *least common multiple*, and $g_i(x) := (x)_i / \mathrm{lcm}_i(x)$. From [12] it follows that $\mathrm{lcm}_i(x) \mid (x)_i$, and that $g_i(x)$ is periodic with period $\mathrm{lcm}(1, \ldots, i-1)$, i.e., $g_i(x) = g_i(x + \mathrm{lcm}(1, \ldots, i-1))$. Let $p_i(x)$ be the sum of the first $n-i$ summands of $p(x)$, i.e., $p_i(x) = a_n \cdot (x)_n + a_{n-1} \cdot (x)_{n-1} + \ldots + a_{n-i+1} \cdot (x)_{n-i+1}$.

Initially, the pebble is located on the northernmost tile of column 0. To test whether $\ell = p(h)$, the robot will move the pebble along the northernmost row in phases, until it is eventually shifted $p(h) - 1$ steps to the *NE* from its original position. If upon termination the pebble is located at the northernmost tile of column $\ell - 1$, then $p(h) = \ell$.

The algorithm proceeds in phases $n, \ldots, 0$. We maintain the invariant that after phase i for all $i > 0$, the pebble is located at the northernmost tile of column $p_i(h)$. That is, in phase i, the robot moves the pebble $|a_i \cdot (h)_i|$ steps *NE*, if a_i is positive, and *SW*, otherwise. In the final phase $i = 0$, the robot moves the pebble by $|a_0 - 1|$ steps *NE*, if $a_0 \geq 1$, and *SW*, otherwise. For now, assume that each movement can be carried out without moving the pebble outside of the parallelogram. We will later describe how to lift this restriction.

We now describe how the pebble is moved by $|a_i \cdot (h)_i|$ steps. First, note that $a_i \cdot (h)_i = a_i \cdot g_i(h) \cdot \mathrm{lcm}_i(h)$. The first factor a_i is a constant. The second factor $g_i(h)$ can be determined as follows. We encode all possible values of $g_i(\cdot)$ for all $i \in \{0, \ldots, n\}$ into the robot's memory, which can be done since n is constant and $g_i(\cdot)$ has a constant period. Before the main algorithm's execution, the robot can compute $g_i(h)$ for all i by moving through column 0 from north to south: Starting with $g_i(1)$, in every step to the south the robot computes the subsequent function value until the period of $g_i(\cdot)$ is reached, in which case it restarts with $g_i(1)$. When it reaches the southernmost tile of the column, it knows $g_i(h)$ for all i.

We next show how the robot moves the pebble by $\mathrm{lcm}_i(h)$ steps, which, by repeating the movement $|a_i \cdot g_i(h)|$ times, concludes how the complete movement by $|a_i \cdot (h)_i|$ steps is performed. Assume the pebble is in some column c and $\mathrm{lcm}_i(h) \mid c$ (which we will prove by induction shortly). The robot alternates between the following two operations: (1) move the pebble into column c' by moving it one step *NE*, if $a_i > 0$, or *SE*, otherwise; (2) verify whether $\mathrm{lcm}_i(h) \mid c'$ as follows. The robot first performs the zig-zag movement from the proof of Theorem 2 to verify whether $h \mid c'$, i.e., whether a *NE* movement moves the robot onto a tile occupied by the pebble. It continues to analogously verify whether $h-1 \mid c'$, $h-2 \mid c'$, $\ldots$, $h-i+1 \mid c'$ by performing a modified zig-zag movement an additional $i-1$ times. Here, the zig-zags of the j-th verification are adjusted accordingly by moving j steps to the south prior to each sequence of *SE* movements. The robot stops alternating between the two above operations once the pebble has been moved to a column c' such that $\mathrm{lcm}_i(h) \mid c'$ for the first time. Then, the pebble must have been moved by $\mathrm{lcm}_i(h)$ steps.

It remains to prove that when the robot wants to move the pebble, currently occupying a node of column c, by $\mathrm{lcm}_i(h)$ steps for some i, then $\mathrm{lcm}_i(h) \mid c$. The invariant holds initially for $c = 0$. Now assume it holds immediately after having moved the pebble by $\mathrm{lcm}_i(h)$ steps

into column $c \pm \mathrm{lcm}_i(h)$. By induction hypothesis, $\mathrm{lcm}_i(h) \mid c$. Afterwards, the robot can move the pebble by either $\mathrm{lcm}_i(h)$ steps again, in which case $\mathrm{lcm}_i(h) \mid c \pm \mathrm{lcm}_i(h)$ holds, or it moves it by $\mathrm{lcm}_{i-1}(h)$ steps, and $\mathrm{lcm}_{i-1}(h) \mid c \pm \mathrm{lcm}_i(h)$ holds since $\mathrm{lcm}_{i-1}(h) \mid \mathrm{lcm}_i(h)$.

Finally, we show how the robot can resolve *overflows*, i.e., situations in which the above algorithm would move the pebble outside of the parallelogram. First, note that the execution of the algorithm after an overflow can, in principle, be continued by the robot by "mirroring" all movements beyond the westernmost or easternmost column, carrying them out into reverse direction. Assume that h is sufficiently large such that $|a_i \cdot (h)_i + \ldots + a_0| \leq p(h)$ for all i. For all small $h = O(\max_i(|a_i|))$ we can encode the constantly many possible function values into the robot's state and test them prior to the algorithm's execution by traversing the two sides of the parallelogram once. If throughout the execution of the algorithm the robot ever attempts to move the pebble into a column west of column 0 or east of "virtual" column 2ℓ (while performing the mirroring method from above), it would subsequently not be able to ever move the pebble back into column ℓ (following from the assumption that h is sufficiently large), and consequently $\ell \neq p(h)$. Therefore, the robot can prematurely terminate with a negative result whenever it encounters such a situation. ◀

The next theorem gives a lower bound on the amount of states needed to decide whether $\ell = h^a$, $a \in \mathbb{N}$, thereby proving that no robot with one pebble can decide whether $\ell = f(h)$ for $f(x) = \omega(x^a)\, \forall a \in \mathbb{N}$.

▶ **Theorem 5.** *A robot with k states and a single pebble cannot decide whether the tile configuration is a parallelogram of height h and length $\ell = f(h)$, $f(x) = \omega(x^{6k+2})$.*

Proof. First, we give a brief outline of the following proof. Assuming such a robot exists, we place the robot with its pebble on the northernmost tile of column 0 of a parallelogram P with height h and length $\ell = f(h)$. We begin by subdividing P horizontally into parallelograms of height h which we will refer to as *blocks*. We define the westernmost and easternmost blocks as the *outer blocks* O_ℓ and O_r, respectively, and denote all other blocks as *inner blocks*. We choose the length b of all inner blocks such that when the robot moves through a sequence of inner blocks (from NW to SE or SE to NW), while either carrying the pebble the entire time or not visiting it at all, its row and state repeat every b columns. The outer blocks will have length at least b.

As in the proof of Theorem 3, we consider the execution of the algorithm as a sequence of tuples of nodes and states, and divide it into subsequences $\pi_1, \ldots, \pi_m$. Subsequence π_i starts with the robot carrying the pebble into some outer block, and ends when it reaches the opposite outer block while carrying the pebble. The robot terminates in π_m, and, as this is the final subsequence, before entering the opposite block while carrying the pebble. We define r_{π_i} and q_{π_i} to be the robot's row and state at the beginning of π_i. By considering where the robot places and picks up the pebble, we identify a value d such that r_{π_i} and q_{π_i} remain the same if the robot is executed on a parallelogram P' of height h and length $f(h) + db$, and therefore falsely terminates with a positive result.

We begin the proof by identifying a value b. Consider the robot's execution without the pebble on a parallelogram of height h and infinite length in both directions, starting in row r and state q. If the robot moves by at most kh columns in NW or SE direction, we define $d_{r,q} := 1$. Otherwise, by the pigeonhole principle, there are two columns that are visited on a node of the same row and in the same state. In this case, we define $d_{r,q}$ to be the distance between these two columns. Note that the robot moves arbitrarily far in this case, its row and state repeating every $d_{r,q}$ columns. Let $D := \{d_{r,q}\}$. Analogously, we consider the execution

of the robot if it initially carries the pebble, and define $d^*_{r,q} := 1$, if it ever drops the pebble or moves by at most kh columns in NW or SE direction. Otherwise, there is a repetition every $d^*_{r,q}$ columns, and the robot carries the pebble indefinitely far. Let $D^* := \{d^*_{r,q}\}$.

We now show that $|D| \leq 3k$. If the robot ever is in row r in state q, or row r' in state q', and visits the northernmost (or southernmost) row in the same state q^* in both executions, the executions will be identical afterwards, and therefore, $d_{r,q} = d_{r',q'}$. Hence, there can only be k combinations of rows and states with distinct executions in which the robot visits the northern- or southernmost row, and these executions contribute at most $2k$ distinct distances to D. Now, let q, r, r' such that the robot does not visit the northern- and southernmost row when started in state q in row r or r'. Clearly, the robot performs the exact same actions in both executions. Therefore, $d_{r,q} = d_{r',q}$, and these executions contribute at most k additional distances. Therefore, $|D| \leq 3k$. Analogously, we have $|D^*| \leq 3k$.

We set $b := \prod_{\delta \in D \cup D^*} \delta$. If $b \leq kh$, redefine $b := (kh + 1)b$. It holds that $b = O(h^{6k})$. We subdivide P into a maximum number of blocks such that inner blocks have length b and outer blocks have length greater than b.

We now identify a value d. Consider the subsequence π_i, $i < m$. W.l.o.g., assume the robot moves the pebble from O_ℓ to O_r. Let (r_j, c_j, q_j), $j = 1, \ldots, l$ be the row, column and state in which the pebble is dropped during π_i. We distinguish two cases: In the first case, the robot moves by more than kh columns while carrying the pebble. In this case, it will move until reaching the easternmost column C_r due to having a repetition in its state and row. Therefore, $c_{j+1} - c_j \leq kh$ for $j < l$. In this case, set $d_i = 1$.

In the second case, the robot does never move by more than kh columns to the east while carrying the pebble during π_i. Thus, the pebble is dropped within the first (i.e., western) kh columns of each inner block. P contains $\Omega(f(h)/b) = \omega(h^2) > (kh)^2$ inner blocks, for sufficiently large h. For some column c, we denote its index inside its block as $\tilde{c}$. Hence, for each inner block, there exists a j such that c_j is in that block and $\tilde{c}_j < kh$. There are at most $h \cdot kh \cdot k = (kh)^2$ possibilities for $(r_j, \tilde{c}_j, q_j)$. By the pigeonhole principle, there exist $s < t$ such that $c_s < c_t$ and $(r_s, \tilde{c}_s, q_s) = (r_t, \tilde{c}_t, q_t)$. We set $d_i := (c_t - c_s)/b$, i.e., the number of inner blocks between c_s and c_t, and $d := \prod_{i=1}^{m} d_i$.

Let P' be the parallelogram of height h and length $f(h) + db$. Now we show that $(r_{\pi_{i+1}}, q_{\pi_{i+1}})$ is the same in the execution on P and P' for all $i < m$ and that the robot terminates with the same result during π_m. To that end, we will again look at a subsequence π_i, $i < m$ where, w.l.o.g, the pebble is carried from O_ℓ to O_r and compare the executions on P and P'. Let (r_j, c_j, q_j), $j = 1, \ldots, l$ as above. The first l times the pebble is dropped on P' are identical to those on P, since the only difference in the execution between dropping the pebble at a column c_j, $j \leq l$ and picking it back up again can be when the robot moves to column $\ell - 1$. As argued above, moving to column $\ell - 1$ on P and P' cannot be distinguished by the robot due to a repetition in row and state and by our choice of b. Thus, the pebble is picked up in the same state again.

Next, we need to look at what happens on P' after the pebble has been picked up for the l-th time. Here, we distinguish between the two cases from above. In the first case, the robot carries the pebble by more than kh columns to the east on P before entering O_r and, consequently, continues until reaching column $\ell - 1$. The same behavior occurs on P', only that the robot traverses an additional d blocks and, by our choice of b, enters O_r in the same row and state as on P.

In the second case, it never carries the pebble by more than kh columns to the east during π_i. We identified s, t such that $(r_s, \tilde{c}_s, q_s) = (r_t, \tilde{c}_t, q_t)$, i.e. the pebble is picked up in the same row, block-column, and state. Note that whenever the robot drops the pebble in some

column $\tilde{c}$ of two different inner blocks, it will pick the pebble up again in the same state: This is clearly true if the robot does not visit column 0 and $\ell - 1$ between dropping and placing the pebble. Otherwise, the robot traverses more than kh columns without seeing the pebble, i.e. its row and state repeat every δ columns for some $\delta \in D$. Consequently, by our choice of b, after both drops the robot will return to the pebble (and pick it up) in the same state. Therefore, in both the executions on P and P' the robot will repeatedly drop and pick up the pebble, each repetition occurring d_i blocks to the east from the previous one. As $d_i \mid d$, the robot enters O_r in the same row on P and P' and we thus also have identical $(r_{\pi_{i+1}}, q_{\pi_{i+1}})$.

It only remains to show that the robot terminates with the same result during π_m. W.l.o.g, let π_m begin in O_l on P, and recall that the pebble does not reach O_r before the robot terminates. As argued before, the executions on P and P' can only differ when the robot drops the pebble and moves to the easternmost column of the respective parallelogram. Due to a repetition in state and row, and by our choice of b, the robot again enters the respective easternmost columns in the same state and row, and afterwards picks up the pebble in the same state on both P and P'. Therefore, the robot ultimately terminates in the same state. ◀

3.3 A Robot With Two Pebbles

Next, we show that having two pebbles enables the robot to decide certain exponential functions. Note that by Theorem 5, the following result is optimal in the number of pebbles used.

▶ **Theorem 6.** *A robot with two pebbles can decide whether a given tile configuration is a parallelogram with height $h > 1$ and length*

$\ell = 2^{2^{\cdot^{\cdot^{2^h}}}}$ *, where the power tower is of constant height.*

Proof. Let $s + 1$ be the height of the power tower (i.e., there are s twos in the function) and denote the two pebbles as a and b. Pebble a always resides in the northernmost row of the parallelogram. We use a's column index as a register on which we perform basic arithmetic operations using b as a helper. Note that although the easternmost column of the parallelogram has index $\ell - 1$, we describe the algorithm as if there was an additional column ℓ. A movement from or to column ℓ can easily be simulated by the robot.

The algorithm is divided into three *stages*. The purpose of the first stage is to verify that ℓ is a power of 2. In the second stage, we move a from column ℓ to column $\log^{(s-1)}(\ell)$, where $\log^{(s-1)}(\cdot)$ denotes the application of the logarithm $s - 1$ times. In the third stage, we verify that a is in column 2^h after the second stage. If in any of the three stages the robot detects a violation of any assumption, i.e., if according to the algorithm a would have to be moved beyond column 0 or ℓ, or should be in column 0 or ℓ, but is not, the robot terminates with a negative result.

Before we describe the three stages in more detail, we describe how a's column index can be multiplied or divided by any constant $c \geq 1$, provided that the result is integer and between 0 and ℓ. We first place b at a tile south of a, and then move a into column 0. In case of a multiplication, we then alternatingly move a *NE* by c steps, and b *SE* by one step, until b reaches column 0. In case of a division, we correspondingly move b by c steps and a by one step.

In the first stage, the robot does the following. It first places a at the northernmost tile of column 1. It then repeatedly multiplies by 2 (i.e., multiplies a's column index by 2) using

the above-mentioned strategy. If b reaches column 0 immediately after a has reached column ℓ, ℓ is a power of 2.

At the beginning of the second stage, a is placed at the northernmost tile of column ℓ. The stage is divided into $s-1$ *phases*, where in each phase a is moved from column i to column $\log(i)$. Note that since s is a constant, the robot can count the number of phases. Furthermore, after the first phase ℓ is verified to be a power of 2, and, as we will show later, if a's column index is 2^x at the beginning of a phase, but x is not a power of 2, then the phase fails. Therefore, a's column index is ensured to be a power of 2 at the beginning of each phase.

In each phase, a is moved from some column 2^x to column x in a step-wise fashion. More precisely, in the j-th step it is moved from column $5^{j-1} \cdot 2^{x/2^{j-1}}$ to column $5^j \cdot 2^{x/2^j}$. This is continued until the resulting column index is not divisible by 4 anymore, i.e., when it becomes $5^{\log x} \cdot 2$. The general idea is to use the exponent of 5 as a counter on the number of times x needs to be divided by 2 until it becomes 1, which yields its logarithm. This idea is based on [7, Section 14].

It remains to show how a single step can be performed. First, the robot moves a from column $5^{j-1} \cdot 2^{x/2^{j-1}}$ into column $5^{j-1} \cdot 3^{x/2^j}$ by alternatingly dividing by 4 and multiplying by 3. After each repetition, the robot verifies whether a's column index is divisible by 2. This is done by traversing the northernmost row from west to east until a is reached, and counting the number of steps modulo 2. Note that if $x/2^{j-1}$ is even, which is the case if x is a power of 2, then the division by 4 is always possible. Otherwise, the division by 4 will fail at latest when $x/2^{j-1}$ becomes 1. Once a's column index is not divisible by 2 anymore, a is in column $5^{j-1} \cdot 3^{x/2^j}$.

Analogously, by alternatingly dividing a's column index by 3 and multiplying by 2 as long as the column index is divisible by 3, the robot afterwards moves a into column $5^{j-1} \cdot 2^{x/2^j}$. By multiplying with 5, a is finally moved into column $5^j \cdot 2^{x/2^j}$.

After each step, the robot verifies whether a's column index is divisible by 4. If so, it continues with the next step. Otherwise, $j = \log x$, and thus a must be in column $5^{\log x} \cdot 2$. From there, a can easily be moved into column x by first dividing by 2, and afterwards alternatingly dividing by 5 and multiplying by 2 until the column index is not divisible by 5 anymore.

Note that if a's column index is 2^x at the beginning of some phase, but x is not a power of 2, then at some step $x/2^{j-1}$ will be odd, and, as described above, the algorithm will fail. Further note that for $h \geq 8$, which can be verified beforehand, moving a from column 2^x (where, consequently, x must be at least 8) to $5^{\log x} \cdot 2$ can only decrease a's column index. Therefore, although a might be moved *NE* in the final steps of a phase, it can easily be seen it will be moved sufficiently far *SW* within the first steps such that it is never moved beyond column ℓ. If that happens nonetheless, the robot terminates with a negative result.

Finally, in the third stage, the robot verifies that a is in column 2^h by diving by 2 for h times. To count up to h, b initially resides in row 2 and is moved one step S after each division. When b reaches a southernmost tile, a must lie in column 4, which can easily be verified by the robot. ◀

▶ Remark. The algorithm of the previous theorem can be adapted for any power towers whose height is a linear function $\alpha h + \beta$ for constants α and β. To count the number of phases in the second stage, we move a south after each phase. The highest exponent of the power tower may also be a function linear in h, which can be handled correspondingly in the third stage.

▶ Remark. The algorithm can further be adapted for any other base β. Note that if β is a composite number, we need to apply the operations to its prime factors separately, taking into account their powers. Since it is well-known that for $n \geq 25$ there is always a prime between n and $(1 + 1/5)n$ [19], we can use the two smallest primes that are no prime factors of β instead of 3 and 5 in the second stage of the algorithm.

▶ Remark. In contrast to the negative results of the previous sections, it can be shown that for every computable function f there exists a computable function $f'(x) = \omega(f(x))$ that can be decided using two pebbles. The main idea used in the proof is to simulate a Turing machine to decide $f(x) = y$ for some x and y by simulating a *program machine* [18] as described in the following section. $f'(x)$ is then chosen as a product of prime factors, where the exponent of one prime factor is $f(x)$, and the exponents of the additional factors are chosen to provide exactly the space required for the simulation. Throughout the algorithm's execution, the values of the program machine's registers are represented as the additional prime factor's exponents. If the Turing machine terminates with a positive result, and the given space exactly matches the required space for the simulation, the robot terminates with a positive result.

3.4 A Family of Functions Requiring an Increasing Amount of Pebbles

The discussion from the previous sections naturally brings up the question whether there is a function family whose detection requires an increasing amount of pebbles. In this section, we answer that question positively. As the proofs are fairly straightforward, we mostly state the general ideas, leaving out some details.

In order to simplify analysis, we consider a robot with pebbles operating on a line segment instead of a parallelogram. We also assume that pebbles are distinguishable and can be placed onto the same tile. Note that it is easy to simulate colors of a finite amount of pebbles on a line segment by keeping track of the pebbles' order. Furthermore, the robot can simulate placing two pebbles onto each other by placing the second pebble within constant distance instead and save the offset from its intended position.

First, we introduce the notion of *program machines* as defined in [18, Section 11].

▶ **Definition 7.** A *program machine* consists of a finite set of registers, holding arbitrary numbers from $\mathbb{N}_0$, and a finite program of numbered instructions from the following instruction set.

- $zero(r)$: Set register $r := 0$.
- $increment(r)$: Set register $r := r + 1$.
- $decrementOrJump(r, n)$: If $r = 0$, jump to instruction n. Otherwise set $r := r - 1$.
- $halt$: Stop the execution.

▶ Remark. Using the instructions from Definition 7, it is possible to simulate instructions to copy the value of register r_1 to r_2, and to jump if (or if not) $r_1 = r_2$ [18].

▶ **Lemma 8.** *A 3 register program machine can simulate a deterministic Turing machine with* $\Gamma = \Sigma = \{0, 1\}$.

Proof. It is well known that a Turing machine can be simulated using two stacks containing the tape's bits behind and in front of the head, respectively. These stacks can be viewed as the binary encoding of natural numbers. They will be stored in registers one and two.

Using the third register as a scratch pad, doubling and halving a register is simple. To pop a bit from the stack, we divide by two and look at the remainder. To push a bit onto the stack, we multiply by two and add the bit. For more details, we refer the reader to [18, Section 11]. ◀

Since we are interested in a robot moving on a finite space, we now define *bounded program machines*.

▶ **Definition 9.** A *bounded program machine* is a program machine whose first register is initialized to the input x. The other registers are initialized to 0. No register may exceed x.

The following observation follows from the fact that a robot can easily simulate a bounded program machine using pebbles, and vice versa.

▶ **Observation 10.** *The computational power of a robot with n pebbles on a line of length $x \geq n$ is between that of an n and an $n+1$ register bounded program machine with input x.*

The next two lemmas show that deterministic linear bounded automata are essentially equivalent in their computational capabilities to bounded program machines and that the number of registers relates to the number of tape symbols needed. This enables us to apply well-known results from complexity theory to our model. We use the definition of deterministic linear bounded automata from [7] as Turing machines that never leave the cells in which their input was placed. Inputs are restricted to $\{0,1\}^*$.

▶ **Lemma 11.** *A deterministic linear bounded automaton with $|\Gamma| = n$ and $\Sigma = \{0,1\}$ with input $x \in \Sigma^*$ can be simulated using a $1 + 2\lceil \log n \rceil$ register bounded program machine with input $x' := (1x)_2$.*

Proof. The construction from Lemma 8 ensures that no register will be increased beyond x' when simulating the linear bounded automaton. The two stacks will now contain elements in Γ. We encode the symbols in binary and store their representation using $2\lceil \log n \rceil$ registers. ◀

▶ **Lemma 12.** *An n register bounded program machine with input $(1x)_2, x \in \{0,1\}^*$ can be simulated by a deterministic linear bounded automaton with 2^n tape symbols and input x.*

Proof. The tape stores the binary representation of the registers' values, each symbol representing one bit of n registers. Operations from Definition 7 are now trivial to perform. ◀

Finally, it can be shown that there exists a family of languages requiring deterministic linear bounded automata with an increasing amount of tape symbols. These will directly translate to sets of accepted line lengths.

▶ **Lemma 13.** *There exists a family of context sensitive languages $S_n \subseteq \{0,1\}^*$ not accepted by any deterministic linear bounded automaton with fewer than n symbols [7, Corollary 2].*

Together with Observation 10, the previous lemmas imply the following theorem.

▶ **Theorem 14.** *There exists a family $L_n \subseteq \mathbb{N}$ such that a robot exists that can detect whether a line has length $\ell \in L_n$ using a finite number of pebbles but none using less than n pebbles.*

To construct a family of functions for deciding side ratios of a parallelogram, we can set

$$f_n(h) = \begin{cases} 1, & h \in L_n \\ 2, & h \notin L_n \end{cases}.$$

▶ Remark. The resulting parallelogram differs from the previous examples in that its length is only 1 or 2. Note that the robot can perform the simulation of the bounded program machine using the parallelograms height, and a length of 2 does not give it too much power. Furthermore, we can modify the function family such that $\ell \geq h$ while still requiring n pebbles to decide $f_n(h)$.

4 Future Work

In this paper we have identified some simple functions that can or cannot be decided with a given set of pebbles. Beyond our study in Section 3.4, we are interested to find functions, such as superexponentials, that require more than only two pebbles. Furthermore, it is an interesting question whether, instead or in addition to using pebbles, multiple robots can help in shape recognition problems. For example, it is still an open question whether two robots, which are activated in an arbitrary order, are more powerful than a single robot with a pebble. Apart from that, there are many different model assumptions under which our problems can be investigated.

In this work we primarily focused on detecting parallelograms of certain side ratios. As our ultimate goal is to investigate shape recognition in general, we are very interested to examine whether our results and algorithms are applicable to other shapes as well. For example, it may be possible to recognize more complex structures such as irregular hexagons with certain side ratios, or to even come up with a more generic procedure to recognize larger families of shapes. Furthermore, rasterized disks or ellipses might be interesting shapes to consider. Other intriguing problems related to shape recognition include testing symmetry or simply-connectedness of a tile structure.

References

1 M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proc. 19th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 132–142, 1978.

2 A. Bonato and R. J. Nowakowski. *The Game of Cops and Robbers on Graphs*. AMS, 2011.

3 L. Budach. Automata and labyrinths. *Mathematische Nachrichten*, 86(1):195–282, 1978.

4 S. Das. Mobile agents in distributed computing: Network exploration. *Bulletin of the European Association for Theoretical Computer Science*, 109:54–69, 2013.

5 Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Universal shape formation for programmable matter. In *Proc. 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 289–299, 2016.

6 Constantine Evans and Erik Winfree. DNA sticky end design and assignment for robust algorithmic self-assembly. In *Proc. of DNA Computing and Molecular Computing (DNA)*, pages 61–75, 2013.

7 Eliot D. Feldman and James C. Owings. A class of universal linear bounded automata. *Information Sciences*, 6(Supplement C):187–190, 1973. doi:10.1016/0020-0255(73)90036-4.

8 F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, jun 2008.

9 P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345(2-3):331–344, 2005.

10 R. Gmyr, I. Kostitsyna, F. Kuhn, C. Scheideler, and T. Strothmann. Forming tile shapes with a single robot. In *Abstr. European Workshop on Computational Geometry (EuroCG)*, pages 9–12, 2017.

11 F. Hoffmann. One pebble does not suffice to search plane labyrinths. In *Proc. International Fundamentals of Computation Theory Conference (FCT)*, pages 433–444, 1981.

12 S. Hong and Y. Yang. On the periodicity of an arithmetical function. *Comptes Rendus Mathematique*, 346(13):717–721, 2008.

13 Ferran Hurtado, Enrique Molina, Suneeta Ramaswami, and Vera Sacristán. Distributed reconfiguraiton of 2D lattice-based modular robotic systems. *Autonomous Robots*, 38(4):383–413, 2015.

14 G. Kilibarda, V. B. Kudryavtsev, and Š. Uščumlić. Collectives of automata in labyrinths. *Discrete Mathematics and Applications*, 13(5):429–466, 2003.

15 G. Kilibarda, V. B. Kudryavtsev, and Š. Uščumlić. Independent systems of automata in labyrinths. *Discrete Mathematics and Applications*, 13(3):221–225, 2003.

16 E. Markou. Identifying hostile nodes in networks using mobile agents. *Bulletin of the European Association for Theoretical Computer Science*, 108:93–129, 2012.

17 Othon Michail and Paul G. Spirakis. Simple and efficient local codes for distributed stable network construction. *Distributed Computing*, 29(3):207–237, 2016.

18 Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., 1967.

19 Jitsuro Nagura. On the interval containing at least one prime number. *Proc. of the Japan Academy*, 28(4):177–181, 1952. doi:10.3792/pja/1195570997.

20 Matthew J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014.

21 A. Pelc. Deterministic rendezvous in networks: A comprehensive survey. *Networks*, 59(3):331–347, 2012.

22 John H. Reif and Sudheer Sahu. Autonomous programmable DNA nanorobotic devices using dnazymes. *Theoretical Computer Science*, 410:1428–1439, 2009.

23 A. N. Shah. Pebble automata on arrays. *Computer Graphics and Image Processing*, 3(3):236–246, 1974.

24 A.J. Thubagere, W. Li, R.F. Johnson, Z. Chen, S. Doroudi, Y.L. Lee, G. Izatt, S. Wittman, N. Srinivas, D. Woods, E. Winfree, and L. Qian. A cargo-sorting DNA robot. *Science*, 357(6356), 2017.

25 S.F.J. Wickham, J. Bath, Y. Katsuda, M. Endo, K. Hidaka, H. Sugiyama, and A.J. Turberfield. A DNA-based molecular motor that can navigate a network of tracks. *Nature Nanotechnology*, 7(3):169–173, 2012.

26 Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *Proc. 4th Conference of Innovations in Theoretical Computer Science (ITCS)*, pages 353–354, 2013.

Conflict Free Feedback Vertex Set: A Parameterized Dichotomy

Akanksha Agrawal
Institute of Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary
agrawal.akanksha@mta.sztaki.hu

Pallavi Jain
Institute of Mathematical Sciences, HBNI, Chennai, India
pallavij@imsc.res.in

Lawqueen Kanesh
Institute of Mathematical Sciences, HBNI, Chennai, India
lawqueen@imsc.res.in

Daniel Lokshtanov
Department of Informatics, University of Bergen, Bergen, Norway
daniello@ii.uib.no

Saket Saurabh
Department of Informatics, University of Bergen, Bergen, Norway
Institute of Mathematical Sciences, HBNI, Chennai, India
UMI ReLax
saket@imsc.res.in

Abstract

In this paper we study recently introduced conflict version of the classical Feedback Vertex Set (FVS) problem. For a family of graphs $\mathcal{F}$, we consider the problem $\mathcal{F}$-CF-Feedback Vertex Set ($\mathcal{F}$-CF-FVS, for short). The $\mathcal{F}$-CF-FVS problem takes as an input a graph G, a graph $H \in \mathcal{F}$ (where $V(G) = V(H)$), and an integer k, and the objective is to decide if there is a set $S \subseteq V(G)$ of size at most k such that $G - S$ is a forest and S is an independent set in H. Observe that if we instantiate $\mathcal{F}$ to be the family of edgeless graphs then we get the classical FVS problem. Jain, Kanesh, and Misra [CSR 2018] showed that in contrast to FVS, $\mathcal{F}$-CF-FVS is W[1]-hard on general graphs and admits an FPT algorithm if $\mathcal{F}$ is the family of d-degenerate graphs. In this paper, we relate $\mathcal{F}$-CF-FVS to the Independent Set problem on special classes of graphs, and obtain a complete dichotomy result on the Parameterized Complexity of the problem $\mathcal{F}$-CF-FVS, when $\mathcal{F}$ is a hereditary graph family. In particular, we show that $\mathcal{F}$-CF-FVS is FPT parameterized by the solution size if and only if $\mathcal{F}$+Cluster IS is FPT parameterized by the solution size. Here, $\mathcal{F}$+Cluster IS is the Independent Set problem in the (edge) union of a graph $G \in \mathcal{F}$ and a cluster graph H (G and H are explicitly given). Next, we exploit this characterization to obtain new FPT results as well as intractability results for $\mathcal{F}$-CF-FVS. In particular, we give an FPT algorithm for $\mathcal{F}$+Cluster IS when $\mathcal{F}$ is the family of $K_{i,j}$-free graphs. We show that for the family of bipartite graph $\mathcal{B}$, $\mathcal{B}$-CF-FVS is W[1]-hard, when parameterized by the solution size. Finally, we consider, for each $0 < \epsilon < 1$, the family of graphs $\mathcal{F}_\epsilon$, which comprise of graphs G such that $|E(G)| \leq |V(G)|^{2-\epsilon}$, and show that $\mathcal{F}_\epsilon$-CF-FVS is W[1]-hard, when parameterized by the solution size, for every $0 < \epsilon < 1$.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis, Theory of computation → Fixed parameter tractability, Theory of computation → W hierarchy

Keywords and phrases Conflict-free, Feedback Vertex Set, FPT algorithm, W[1]-hardness

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.53

Funding This research has received funding from the European Research Council under ERC grant no. 306992 PARAPPROX, ERC grant no. 715744 PaPaALG, ERC grant no. 725978 SYSTEMATICGRAPH, and DST, India for SERB-NPDF fellowship [PDF/2016/003508].

1 Introduction

Feedback Vertex Set (FVS) is one of the classical NP-hard problems that has been subjected to intensive study in algorithmic paradigms that are meant for coping with NP-hard problems, and particularly in the realm of Parameterized Complexity. In this problem, given a graph G and an integer k, the objective is to decide if there is $S \subseteq V(G)$ of size at most k such that $G - S$ is a forest. FVS has received a lot of attention in the realm of Parameterized Complexity. This problem is known to be in FPT, and the best known algorithm for it runs in time $\mathcal{O}(3.618^k n^{\mathcal{O}(1)})$ [8, 13]. Several variant and generalizations of Feedback Vertex Set such as Weighted Feedback Vertex Set [2, 7], Independent Feedback Vertex Set [1, 14], Connected Feedback Vertex Set [15], and Simultaneous Feedback Vertex Set [3, 6] have been studied from the viewpoint of Parameterized Complexity.

Recently, Jain et al. [12] defined an interesting generalization of well-studied vertex deletion problems – in particular for FVS. The CF-Feedback Vertex Set (CF-FVS, for short) problem takes as input graphs G and H, and an integer k, and the objective is to decide if there is a set $S \subseteq V(G)$ of size at most k such that $G - S$ is a forest and S is an independent set in H. The graph H is also called a *conflict graph*. Observe that the CF-FVS problem generalizes classical graph problems, Feedback Vertex Set and Independent Feedback Vertex Set. A natural way of defining CF-FVS will be by fixing a family $\mathcal{F}$ from which the conflict graph H is allowed to belong. Thus, for every fixed $\mathcal{F}$ we get a new CF-FVS problem. In particular we get the following problem.

$\mathcal{F}$-CF-Feedback Vertex Set ($\mathcal{F}$-CF-FVS)	**Parameter:** k
Input: A graph G, a graph $H \in \mathcal{F}$ (where $V(G) = V(H)$), and an integer k.	
Question: Is there a set $S \subseteq V(G)$ of size at most k, such that $G - S$ is a forest and S is an independent set in H?	

Jain et al. [12] showed that $\mathcal{F}$-CF-FVS is W[1]-hard when $\mathcal{F}$ is a family of all graphs and admits FPT algorithm when the input graph H is from the family of d-degenerate graphs and the family of nowhere dense graphs. The most natural question that arises here is the following.

Question 1: For which graph families $\mathcal{F}$, $\mathcal{F}$-CF-FVS is FPT?

Our Results. Starting point of our research is Question 1. We obtain a complete dichotomy result on the Parameterized Complexity of the problem $\mathcal{F}$-CF-FVS (for hereditary $\mathcal{F}$) in terms of another well-studied problem, namely, the Independent Set problem – the wall of intractability. Towards stating our results, we start by defining the problem $\mathcal{F}$+Cluster IS, which is of independent interest. A *cluster graph* is a graph formed from the disjoint union of complete graphs (or cliques).

$\mathcal{F}$+Cluster Independent Set ($\mathcal{F}$+Cluster IS)	**Parameter:** k
Input: A graph $G \in \mathcal{F}$, a cluster graph H (where $V(G) = V(H)$), and an integer k, such that H has exactly k connected components.	
Question: Is there a set $S \subseteq V(G)$ of size k, such that S is an independent set in both G and in H?	

We note that $\mathcal{F}$+Cluster IS is the Independent Set problem on the edge union of two graphs, where one of the graphs is from the family of graphs $\mathcal{F}$ and the other one is a cluster graph. Here, additionally we know the partition of edges into two sets, E_1 and E_2 such that the graph induced on E_1 is in $\mathcal{F}$ and the graph induced on E_2 is a cluster graph. We note that $\mathcal{F}$+Cluster IS has been studied in the literature for $\mathcal{F}$ being the family of interval graphs (with no restriction on the number of clusters) [18]. They showed the problem to be FPT. Recently, Bentert et al. [4] generalized the result from interval graphs to chordal graphs. This problem arises naturally in the study of scheduling problems. We refer the readers to [18, 4] for more details on the application of $\mathcal{F}$+Cluster IS.

We are now ready to state our results. We show that $\mathcal{F}$-CF-FVS is in FPT if and only if $\mathcal{F}$+Cluster IS is in FPT, where $\mathcal{F}$ is a family of hereditary graphs. We obtain a complete characterization of when the $\mathcal{F}$-CF-FVS problem is in FPT, for hereditary graph families. To prove the forward direction, i.e., showing that $\mathcal{F}$+Cluster IS is in FPT implies $\mathcal{F}$-CF-FVS is in FPT, we design a branching based algorithm, which at the base case generates instances of $\mathcal{F}$+Cluster IS, which is solved using the assumed FPT algorithm for $\mathcal{F}$+Cluster IS. Thus, we give "fpt-turing-reduction" from $\mathcal{F}$-CF-FVS to $\mathcal{F}$+Cluster IS. It is worth to note that there are very few known reductions of this nature. To show that $\mathcal{F}$-CF-FVS is in FPT implies that $\mathcal{F}$+Cluster IS is in FPT, we give an appropriate reduction from $\mathcal{F}$+Cluster IS to $\mathcal{F}$-CF-FVS, which proves the statement. We note that our result that $\mathcal{F}$-CF-FVS is in FPT implies $\mathcal{F}$+Cluster IS is in FPT, holds for all families of graphs.

Next, we consider two families of graphs. We first design FPT algorithm for the corresponding $\mathcal{F}$+Cluster IS problem. For the second class we give a hardness result. First, we consider the problem $K_{i,j}$-free+Cluster IS, which is the $\mathcal{F}$+Cluster IS problem for the family of $K_{i,j}$-free graphs. We design an FPT algorithm for $K_{i,j}$-free+Cluster IS based on branching together with solving the base cases using a greedy approach. This adds another family of graphs, apart from interval and chordal graphs, such that $\mathcal{F}$+Cluster IS is FPT.

We note that $K_{i,j}$-free graphs have at most $n^{2-\epsilon}$ edges, where n is the number of vertices in the input graph and $\epsilon = \epsilon(i,j) > 0$ [17, 11]. We complement our FPT result on $K_{i,j}$-free+Cluster IS with the W[1]-hardness result of the $\mathcal{F}$+Cluster IS problem when $\mathcal{F}$ is the family of graphs with at most $n^{2-\epsilon}$ edges. This result is obtained by giving an appropriate reduction from the problem Multicolored Biclique, which is known to be W[1]-hard [8, 10]. We also show that the $\mathcal{F}$+Cluster IS problem is W[1]-hard when $\mathcal{F}$ is the family of bipartite graphs. Again, this result is obtained via a reduction from Multicolored Biclique.

2 Preliminaries

In this section, we state some basic definitions and terminologies from Graph Theory that are used in this paper. For the graph related terminologies which are not explicitly defined here, we refer the reader to the book of Diestel [9].

Graphs. Consider a graph G. By $V(G)$ and $E(G)$ we denote the set of vertices and edges in G, respectively. When the graph is clear from the context, we use n and m to denote the number of vertices and edges in the graph, respectively. For $X \subseteq V(G)$, by $G[X]$ we denote the subgraph of G with vertex set X and edge set $\{uv \in E(G) \mid u, v \in X\}$. Moreover, by $G - X$ we denote graph $G[V(G) \setminus X]$. For $v \in V(G)$, $N_G(v)$ denotes the set $\{u \mid uv \in E(G)\}$, and $N_G[v]$ denotes the set $N_G(v) \cup \{v\}$. By $\mathsf{deg}_G(v)$ we denote the size of $N_G(v)$. A *path* $P = (v_1, \ldots, v_n)$ is an ordered collection of vertices, with endpoints v_1 and v_n, such that there is an edge between every pair of consecutive vertices in P. A *cycle* $C = (v_1, \ldots, v_n)$ is

a path with the edge v_1v_n. Consider graphs G and H. We say that G is an *H-free* graph if no subgraph of G is isomorphic to H. For $u, v \in V(G) \cap V(H)$, we say that u and v are in *conflict* in G with respect to H if $uv \in E(H)$.

3 W-hardness of $\mathcal{F}$-CF-FVS Problems

This section is devoted to showing W-hardness results for $\mathcal{F}$-CF-FVS problems for certain graph classes, $\mathcal{F}$. In Section 3.1, we show one direction of our dichotomy result. That is, if for a family of graphs $\mathcal{F}$, $\mathcal{F}$+Cluster IS is not in FPT when parameterized by the size of solution then $\mathcal{F}$-CF-FVS is also not in FPT when parameterized by the size of solution. This result is obtained by giving a parameterized reduction from $\mathcal{F}$+Cluster IS to $\mathcal{F}$-CF-FVS. Next, we show that the problem $\mathcal{F}$-CF-FVS is W[1]-hard, when parameterized by the size of solution, where $\mathcal{F}$ is the family of bipartite graphs (Section 3.2) or the family of graphs with sub-quadratic number of edges (Section 3.3). These results are obtained by giving an appropriate reduction from the problem Multicolored Biclique, which is known to be W[1]-hard [8, 10].

3.1 $\mathcal{F}$+Cluster IS to $\mathcal{F}$-CF-FVS

In this section, we show that, for a family of graphs $\mathcal{F}$, if $\mathcal{F}$+Cluster IS is not in FPT, then $\mathcal{F}$-CF-FVS is also not in FPT (where the parameters are the solution sizes). To prove this result, we give a parameterized reduction from $\mathcal{F}$+Cluster IS to $\mathcal{F}$-CF-FVS.

Let (G, H, k) be an instance of $\mathcal{F}$+Cluster IS. We construct an instance (G', H', k') of $\mathcal{F}$-CF-FVS as follows. We have $H' = G$, $k' = k$, and $V(G') = V(H)$. Let $\mathcal{C}$ be the set of connected components in H. Recall that we have $|\mathcal{C}| = k$. For each $C \in \mathcal{C}$, we add a cycle (in an arbitrarily chosen order) induced on vertices in $V(C)$ in G'. This completes the description of the reduction. Next, we show the equivalence between the instance (G, H, k) of $\mathcal{F}$+Cluster IS and the instance (G', H', k') of $\mathcal{F}$-CF-FVS.

▶ **Lemma 1.** *(G, H, k) is a* yes *instance of $\mathcal{F}$+Cluster IS if and only if (G', H', k') is a* yes *instance of $\mathcal{F}$-CF-FVS.*

Proof. In the forward direction, let (G, H, k) be a yes instance of $\mathcal{F}$+Cluster IS, and S be one of its solution. Since $H' = G$, therefore, S is an independent set in H'. Let $\mathcal{C}$ be the set of connected components in H. As S is a solution, it must contain exactly one vertex from each $C \in \mathcal{C}$. Moreover, G' comprises of vertex disjoint cycles for each $C \in \mathcal{C}$. Thus S intersects every cycle in G'. Therefore, S is a solution to $\mathcal{F}$-CF-FVS in (G', H', k').

In the reverse direction, let (G', H', k') be a yes instance of $\mathcal{F}$-CF-FVS, and S be one of its solution. Recall that G' comprises of k vertex disjoint cycles, each corresponding to a connected component $C \in \mathcal{C}$, where $\mathcal{C}$ is the set of connected components in H. Therefore, S contains exactly one vertex from each $C \in \mathcal{C}$. Also, $H' = G$, and therefore, S is an independent set in G. This implies that S is a solution to $\mathcal{F}$+Cluster IS in (G, H, k). ◀

Now we are ready to state the main theorem of this section.

▶ **Theorem 2.** *For a family of graphs $\mathcal{F}$, if $\mathcal{F}$+Cluster IS is not in FPT when parameterized by the solution size, then $\mathcal{F}$-CF-FVS is also not in FPT when parameterized by the solution size.*

3.2 W[1]-hardness on Bipartite Graphs

In this section, we show that for the family of bipartite graphs, $\mathcal{B}$, the $\mathcal{B}$-CF-FVS problem is W[1]-hard, when parameterized by the solution size. Throughout this section, $\mathcal{B}$ will denote the family of bipartite graphs. To prove our result, we give a parameterized reduction from the problem Multicolored Biclique to $\mathcal{B}$-CF-FVS. In the following, we formally define the problem Multicolored Biclique.

Multicolored Biclique (MBC) **Parameter:** k
Input: A bipartite graph G, a partition of A into k sets $A_1, A_2, \cdots, A_k$, and a partition of B into k sets $B_1, B_2, \cdots, B_k$, where A and B are a vertex bipartition of G.
Question: Is there a set $S \subseteq V(G)$ such that for each $i \in [k]$ we have $|S \cap A_i| = 1$ and $|S \cap B_i| = 1$, and $G[S]$ is isomorphic to $K_{k,k}$?

Let $(G, A_1, \cdots, A_k, B_1, \cdots, B_k)$ be an instance of Multicolored Biclique. We construct an instance (G', H', k') of $\mathcal{B}$-CF-FVS as follows. We have $V(G') = V(H') = V(G)$, and $E(H') = \{uv \mid u \in \cup_{i\in[k]} A_i, v \in \cup_{i\in[k]} B_i, \text{ and } uv \notin E(G)\}$. Next, for each $i \in [k]$, we add a cycle (in an arbitrary order) induced on vertices in A_i in G'. Similarly, we add for each $i \in [k]$, a cycle induced on vertices in B_i in G'. Notice that G' comprises of $2k$ vertex disjoint cycles, and H' is a bipartite graph. Finally, we set $k' = 2k$. This completes the description of the reduction.

▶ **Lemma 3.** *$(G, A_1, \cdots, A_k, B_1, \cdots, B_k)$ is a* yes *instance of* Multicolored Biclique *if and only if (G', H', k') is a* yes *instance of $\mathcal{B}$-CF-FVS.*

Now we are ready to sate the main theorem of this section.

▶ **Theorem 4.** *$\mathcal{B}$-CF-FVS parameterized by the solution size is W[1]-hard, where $\mathcal{B}$ is the family of bipartite graphs.*

3.3 W[1]-hardness on Graphs with Sub-quadratic Edges

In this section, we show that $\mathcal{F}$-CF-FVS is W[1]-hard, when parameterized by the solution size, where $\mathcal{F}$ is the family of graphs with sub-quadratic edges. To formalize the family of graphs with subquadratic edges, we define the following. For $0 < \epsilon < 1$, we define $\mathcal{F}_\epsilon$ to be the family comprising of graphs G, such that $|E(G)| \leq |V(G)|^{2-\epsilon}$. We show that for every $0 < \epsilon < 1$, the $\mathcal{F}_\epsilon$-CF-FVS problem is W[1]-hard, when parameterized by the solution size. Towards this, for each (fixed) $0 < \epsilon < 1$, we give a parameterized reduction from Multicolored Biclique to $\mathcal{F}_\epsilon$-CF-FVS.

Let $(G, A_1, \cdots, A_k, B_1, \cdots, B_k)$ be an instance of Multicolored Biclique. We construct an instance (G', H', k') of $\mathcal{F}_\epsilon$-CF-FVS as follows. Let $n = |V(G)|$, $m = |E(G)|$, and X be a set comprising of $n^{\frac{2}{2-\epsilon}} - n$ (new) vertices. The vertex set of G' and H' is $X \cup V(G)$. For each $i \in [k]$, we add a cycle (in arbitrary order) induced on vertices in A_i in G'. Similarly, we add for each $i \in [k]$, a cycle induced on vertices in B_i in G'. Also, we add a cycle induced on vertices in X to G'. We have $E(H') = \{uv \mid u \in \cup_{i\in[k]} A_i, v \in \cup_{i\in[k]} B_i, \text{ and } uv \notin E(G)\}$. Finally, we set $k' = 2k + 1$. Notice that since $|V(H')| = n^{\frac{2}{2-\epsilon}}$, and $|E(H')| < n^2$, therefore, $H \in \mathcal{F}_\epsilon$.

▶ **Lemma 5.** *$(G, A_1, \cdots, A_k, B_1, \cdots, B_k)$ is a* yes *instance of* Multicolored Biclique *if and only if (G', H', k') is a* yes *instance of $\mathcal{F}_\epsilon$-CF-FVS.*

Now we are ready to state the main theorem of this section.

▶ **Theorem 6.** *For $0 < \epsilon < 1$, $\mathcal{F}_\epsilon$-CF-FVS parameterized by the solution size is* W[1]-hard.

4 FPT algorithms for $\mathcal{F}$-CF-FVS for Restricted Conflict Graphs

For a hereditary (closed under taking induced subgraphs) family of graphs $\mathcal{F}$, we show that if $\mathcal{F}$+Cluster IS is FPT, then $\mathcal{F}$-CF-FVS is FPT. Throughout this section, whenever we refer to a family of graphs, it will refer to a hereditary family of graphs. To prove our result, for a family of graphs $\mathcal{F}$, for which $\mathcal{F}$+Cluster IS is FPT, we will design an FPT algorithm for $\mathcal{F}$-CF-FVS, using the (assumed) FPT algorithm for $\mathcal{F}$+Cluster IS. We note that this gives us a Turing parameterized reduction from $\mathcal{F}$-CF-FVS to $\mathcal{F}$+Cluster IS. Our algorithm will use the technique of compression together with branching. We note that the method of iterative compression was first introduced by Reed, Smith, and Vetta [16], and in our algorithm, we (roughly) use only the compression procedure from it.

In the following, we let $\mathcal{F}$ to be a (fixed hereditary) family of graphs, for which $\mathcal{F}$+Cluster IS is in FPT. Towards designing an algorithm for $\mathcal{F}$-CF-FVS, we define another problem, which we call $\mathcal{F}$-Disjoint Conflict Free Feedback Vertex Set (to be defined shortly). Firstly, we design an FPT algorithm for $\mathcal{F}$-CF-FVS using an assumed FPT algorithm for $\mathcal{F}$-Disjoint Conflict Free Feedback Vertex Set. Secondly, we give an FPT algorithm for $\mathcal{F}$-Disjoint Conflict Free Feedback Vertex Set using the assumed algorithm for $\mathcal{F}$+Cluster IS. In the following, we formally define the problem $\mathcal{F}$-Disjoint Conflict Free Feedback Vertex Set ($\mathcal{F}$-DCF-FVS, for short)

$\mathcal{F}$-Disjoint Conflict Free Feedback Vertex Set ($\mathcal{F}$-DCF-FVS) **Parameter:** k
Input: A graph G, a graph $H \in \mathcal{F}$, an integer k, a set $W \subseteq V(G)$, a set $R \subseteq V(H) \setminus W$, and a set $\mathcal{C}$, such that the following conditions are satisfied: 1) $V(G) \subseteq V(H)$, 2) $G - W$ is a forest, 3) the number of connected components in $G[W]$ is at most k, and 4) $\mathcal{C}$ is a set of vertex disjoint subsets of $V(H)$.
Question: Is there a set $S \subseteq V(H) \setminus (W \cup R)$ of size at most k, such that $G - S$ is a forest, S is an independent set in H, and for each $C \in \mathcal{C}$, we have $\lvert S \cap C \rvert \neq \emptyset$?

We note that in the definition of $\mathcal{F}$-DCF-FVS, there are three additional inputs (i.e. W, R and C). The purpose and need for these sets will become clear when we describe the algorithm for $\mathcal{F}$-DCF-FVS. In Section 4.1, we will prove the following theorem.

▶ **Theorem 7.** *Let $\mathcal{F}$ be a hereditary family of graphs for which there is an FPT algorithm for $\mathcal{F}$+Cluster IS running in time $f(k)n^{\mathcal{O}(1)}$, where n is the number of vertices in the input graph. Then, there is an FPT algorithm for $\mathcal{F}$-DCF-FVS running in time $16^k f(k)n^{\mathcal{O}(1)}$, where n is the (total) number of vertices in the input graphs.*

In the rest of the section, we show how we can use the FPT algorithm for $\mathcal{F}$-DCF-FVS to obtain an FPT algorithm for $\mathcal{F}$-CF-FVS.

An Algorithm for $\mathcal{F}$-CF-FVS using the algorithm for $\mathcal{F}$-DCF-FVS. Let $I = (G, H, k)$ be an instance of $\mathcal{F}$-CF-FVS. We start by checking whether or not G has a feedback vertex set of size at most k, i.e. a set Z of size at most k, such that $G - Z$ is a forest. For this we employ the algorithm for Feedback Vertex Set running in time $\mathcal{O}(3.619^k n^{\mathcal{O}(1)})$ of Kociumaka and Pilipczuk [13]. Here, n is the number of vertices in the input graph. Notice that if G does not have a feedback vertex set of size at most k, then (G, H, k) is a no instance of $\mathcal{F}$-CF-FVS, and we can output a trivial no instance of $\mathcal{F}$-DCF-FVS. Therefore, we assume that (G, k) is a yes instance of Feedback Vertex Set, and let Z be one of its solution. We note that such a set Z can be computed using the algorithm presented in [13]. We generate an instance I_Y of $\mathcal{F}$-DCF-FVS, for each $Y \subseteq Z$, where Y is the guessed (exact) intersection of the set Z with an assumed (hypothetical) solution to $\mathcal{F}$-CF-FVS in I. We now formally describe the construction of I_Y. Consider a set $Y \subseteq Z$, such that Y is an independent set in H. Let

$G_Y = G - Y$, $H_Y = H - Y$, $k_Y = k - |Y|$, $W_Y = Z \setminus Y$, $R_Y = (N_H(Y) \setminus W_Y) \cap V(H_Y)$, and $\mathcal{C}_Y = \emptyset$. Furthermore, let $I_Y = (G_Y, H_Y, k_Y, W_Y, R_Y, \mathcal{C}_Y)$, and notice that I_Y is a (valid) instance of $\mathcal{F}$-DCF-FVS. Now we resolve I_Y using the (assumed) FPT algorithm for $\mathcal{F}$-DCF-FVS, for each $Y \subseteq Z$, where Y is an independent set in H. It is easy to see that I is a yes instance of $\mathcal{F}$-CF-FVS if and only if there is an independent set $Y \subseteq Z$ in H, such that I_Y is a yes instance of $\mathcal{F}$-DCF-FVS. From the above discussions, we obtain the following lemma.

▶ **Lemma 8.** *Let $\mathcal{F}$ be a family of graphs for which $\mathcal{F}$-DCF-FVS admits an FPT algorithm running in time $f(k)c^k n^{\mathcal{O}(1)}$, where n is the (total) number of vertices in the input graph. Then $\mathcal{F}$-CF-FVS admits an FPT algorithm running in time $f(k)(1+c)^k n^{\mathcal{O}(1)}$, where n is the number of vertices in the input graphs.*

Using Theorem 7 and Lemma 8, we obtain the main theorem of this section.

▶ **Theorem 9.** *Let $\mathcal{F}$ be a hereditary family of graphs for which there is an FPT algorithm for $\mathcal{F}$+Cluster IS running in time $f(k)n^{\mathcal{O}(1)}$, where n is the number of vertices in the input graph. Then, there is an FPT algorithm for $\mathcal{F}$-CF-FVS running in time $17^k f(k)n^{\mathcal{O}(1)}$, where n is the number of vertices in the input graphs of $\mathcal{F}$-CF-FVS.*

4.1 FPT Algorithm for $\mathcal{F}$-DCF-FVS

The goal of this section is to prove Theorem 7. Let $\mathcal{F}$ be a (fixed) hereditary family of graphs, for which $\mathcal{F}$+Cluster IS admits an FPT algorithm. We design a branching based FPT algorithm for $\mathcal{F}$-DCF-FVS, using the (assumed) FPT algorithm for $\mathcal{F}$+Cluster IS.

Let $I = (G, H, k, W, R, \mathcal{C})$ be an instance of $\mathcal{F}$-DCF-FVS. In the following we describe some reduction rules, which the algorithm applies exhaustively, in the order in which they are stated.

▶ **Reduction Rule 1.** *Return that $(G, H, k, W, R, \mathcal{C})$ is a no instance of $\mathcal{F}$-DCF-FVS if one of the following conditions are satisfied:*

1. *if $k < 0$,*
2. *if $k = 0$ and G has a cycle,*
3. *$k = 0$ and $\mathcal{C} \neq \emptyset$,*
4. *$G[W]$ has a cycle,*
5. *if $|\mathcal{C}| > k$, or*
6. *there is $C \in \mathcal{C}$, such that $C \subseteq R$.*

▶ **Reduction Rule 2.** *If $k = 0$, G is acyclic, and $\mathcal{C} = \emptyset$, then return that $(G, H, k, W, R, \mathcal{C})$ is a yes instance of $\mathcal{F}$-DCF-FVS.*

In the following, we state a lemma, which is useful in resolving those instances where the graph G has no vertices.

▶ **Lemma 10.** *Let $(G, H, k, W, R, \mathcal{C})$ be an instance of $\mathcal{F}$-DCF-FVS, where Reduction Rules 1 is not applicable and $G - W$ has no vertices. Then, in polynomial time, we can generate an instance (G', H', k') of $\mathcal{F}$+Cluster IS, such that $(G, H, k, W, R, \mathcal{C})$ is a yes instance of $\mathcal{F}$-DCF-FVS if and only if (G', H', k') is a yes instance of $\mathcal{F}$+Cluster IS.*

Lemma 10 leads us to the following reduction rule.

▶ **Reduction Rule 3.** *If $G - W$ has no vertices, then return the output of algorithm for $\mathcal{F}$+Cluster IS with the instance generated by Lemma 10.*

▶ **Reduction Rule 4.** *If there is a vertex $v \in V(G)$ of degree at most one in G, then return $(G - \{v\}, H, k, W \setminus \{v\}, R, \mathcal{C})$.*

The safeness of Reduction Rule 4 follows from the fact that a vertex of degree at most one does not participate in any cycle.

▶ **Reduction Rule 5.** *Let $uv \in E(G)$ be an edge of multiplicity greater than 2 in G, and G' be the graph obtained from G by reducing the multiplicity of uv in G to 2. Then, return $(G', H, k, W, R, \mathcal{C})$.*

The safeness of Reduction Rule 5 follows from the fact that for an edge, multiplicity of 2 is enough to capture multiplicities of size larger than 2.

▶ **Reduction Rule 6.** *Let $v \in R$ be a degree 2 vertex in G with u and w being its neighbors in G. Furthermore, let G' be the graph obtained from G by deleting v and adding the (multi) edge uw. Then, return $(G', H - \{v\}, k, W, R \setminus \{v\}, \mathcal{C})$.*

The safeness of Reduction Rule 6 follows from the fact that a vertex in R cannot be part of any solution and any cycle (in G) containing v must contain both u and w.

▶ **Reduction Rule 7.** *If there is $v \in (V(G) \cap R)$, such that v has at least two neighbors in the same connected component of W, then return that $(G, H, k, W, R, \mathcal{C})$ is a no instance of $\mathcal{F}$-DCF-FVS.*

▶ **Reduction Rule 8.** *If there is $v \in V(G) \setminus (W \cup R)$, such that v has at least two neighbors in the same connected component of W, then return $(G - \{v\}, H - \{v\}, k - 1, W, R \cup N_H(v), \mathcal{C})$.*

▶ **Reduction Rule 9.** *Let $v \in V(G) \cap R$, such that $N_G(v) \cap W \neq \emptyset$. Then, return $(G, H, k, W \cup \{v\}, R \setminus \{v\}, \mathcal{C})$.*

Let η be the number of connected components in $G[W]$. In the following, we define the measure we use to compute the running time of our algorithm.

$$\mu(I) = \mu((G, H, k, W, R, \mathcal{C})) = k + \eta - |\mathcal{C}|$$

Observe that none of the reduction rules that we described increases the measure, and a reduction rule can be applied only polynomially many time. When none of the reduction rules are applicable, the degree of each vertex in G is at least two, multiplicity of each edge in G is at most two, degree two vertices in G do not belong to the set R, and $G[W]$ and $G - W$ are forests. Furthermore, for each $v \in V(G) \setminus W$, v has at most 1 neighbor (in G) in a connected component of $G[W]$.

In the following, we state the branching rules used by the algorithm. We assume that none of the reduction rules are applicable, and the branching rules are applied in the order in which they are stated. The algorithm will branch on vertices in $V(G) \setminus W$.

▶ **Branching Rule 1.** *If there is $v \in V(G) \setminus W$ that has at least two neighbors (in G), say $w_1, w_2 \in W$. Since Reduction Rule 7 and 8 are not applicable, w_1 and w_2 belong to different connected components of $G[W]$. Also, since Reduction Rule 9 is not applicable, we have $v \notin R$. In this case, we branch as follows.*

- **(i)** *v belongs to the solution. In this branch, we return $(G - \{v\}, H - \{v\}, k - 1, W, R \cup N_H(v), \mathcal{C})$.*
- **(ii)** *v does not belongs to the solution. In this branch, we return $(G, H, k, W \cup \{v\}, R, \mathcal{C})$.*

In one branch when v belongs to the solution, k decreases by 1, and η and $|\mathcal{C}|$ do not change. Hence, μ decreases by 1. In other branch when v is moved to W, number of components in η decreases by at least one, and k and $|\mathcal{C}|$ do not change. Therefore, μ decreases by at least 1. The resulting branching vector for the above branching rule is $(1, 1)$.

If Branching Rule 1 is not applicable, then each $v \in V(G) \setminus W$ has at most one neighbor (in G) in the set W. Moreover, since Reduction Rule 4 is not applicable, each leaf in $G - W$ has a neighbor in W.

In the following, we introduce some notations, which will be used in the description of our branching rules. Recall that $G - W$ is a forest. Consider a connected component T in $G - W$. A path P_{uv} from a vertex u to a vertex v in T is *nice* if u and v are of degree at least 2 in G, all internal vertices (if they exist) of P_{uv} are of degree exactly 2 in G, and v is a leaf in T. In the following, we state an easy proposition, which will be used in the branching rules that we design.

▶ **Proposition 1.** *Let $(G, H, k, W, R, \mathcal{C})$ be an instance of $\mathcal{F}$-DCF-FVS, where none of Reduction Rule 1 to 9 or Branching Rule 1 apply. Then there are vertices $u, v \in V(G) \setminus W$, such that the unique path P_{uv} in $G - W$ is a nice path.*

Consider $u, v \in V(G) \setminus W$, for which there is a nice path P_{uv} in T, where T is a connected component of $G - W$. Since Reduction Rule 4 is not applicable, either u has a neighbor in W, or u has degree at least 2 in T. From the above discussions, together with Proposition 1, we design the remaining branching rules used by the algorithm. We note that the branching rules that we describe next is similar to the one given in [3].

▶ **Branching Rule 2.** *Let $v \in V(G) \setminus W$ be a leaf in $G - W$ for which the following holds. There is $u \in V(G) \setminus W$, such that $N_G(u) \cap W \neq \emptyset$ and there is a nice path P_{uv} from u to v in $G - W$. Let $C = V(P_{uv}) \setminus \{u\}$, u' and v' be the neighbors (in G) of u and v in W, respectively. Observe that since Reduction Rule 9 is not applicable, we have $u, v \notin R$. We further consider the following cases, based on whether or not u' and v' are in the same connected component of $G[W]$.*

Case 2.A. *u' and v' are in the same connected component of $G[W]$. In this case, $G[V(P_{uv}) \cup W]$ contains exactly one cycle, and this cycle contains all vertices of $V(P_{uv})$ (consecutively). Since vertices in W cannot be part of any solution, either u belongs to the solution or a vertex from C belongs to the solution. Moreover, any cycle in G containing v must contain all vertices in $V(P_{uv})$, consecutively. This leads to the following branching rule.*

- **(i)** *u belongs to the solution. In this branch, we return $(G - \{u\}, H - \{u\}, k - 1, W, R \cup N_H(u), \mathcal{C})$.*
- **(ii)** *u does not belong to the solution. In this branch, we return $(G - C, H, k, W, R, \mathcal{C} \cup \{C\})$. In the first branch k decreases by one, and η and $|\mathcal{C}|$ do not change. Therefore, μ decreases by 1. On the second branch $|\mathcal{C}|$ increases by 1, and k and η do not change, and therefore, μ decreases by 1. The resulting branching vector for the above branching rule is $(1, 1)$.*

Case 2.B. *u' and v' are in different connected component of $G[W]$. In this case, we branch as follows.*

- **(i)** *u belongs to the solution. In this branch, we return $(G - \{u\}, H - \{u\}, W, k - 1, R \cup N_H(u), \mathcal{C})$.*
- **(ii)** *A vertex from C is in the solution. In this branch, we return $(G - C, H, k, W, R, \mathcal{C} \cup \{C\})$.*
- **(iii)** *No vertex in $\{u\} \cup C$ is in the solution. In this branch, we add all vertices in $\{u\} \cup C$ to W. That is, we return $(G, H, k, W \cup (\{u\} \cup C), R \setminus (\{u\} \cup C), \mathcal{C})$.*

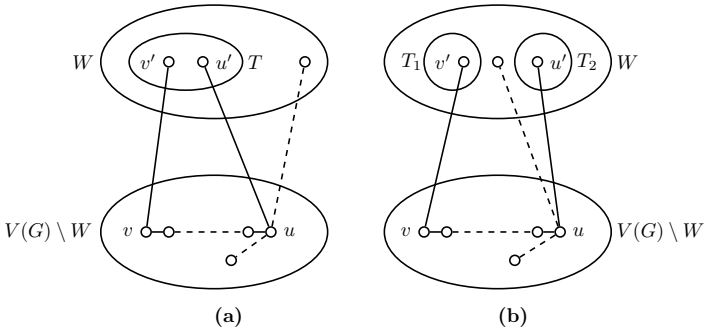


Figure 1 The cases handled by Branching Rule 2, (a) T is a connected component in $G[W]$, similarly in (b) T_1, T_2 are connected components in $G[W]$.

In the first branch k decreases by one, and η and $|\mathcal{C}|$ do not change. Therefore, μ decreases by 1. On the second branch $|\mathcal{C}|$ increases by 1, and k and η do not change, and therefore, μ decreases by 1. In the third branch, η decreases by one, and k and $|\mathcal{C}|$ do not change. The resulting branching vector for the above branching rule is $(1, 1, 1)$.

▶ **Branching Rule 3.** *There is $u \in V(G) \setminus W$ which has (at least) two nice paths, say P_{uv_1} and P_{uv_2} to leaves v_1 and v_2 (in $G - W$). Let $C_1 = V(P_{uv_1}) \setminus \{u\}$ and $C_2 = V(P_{uv_2}) \setminus \{u\}$. We further consider the following cases depending on whether or not v_1 and v_2 have neighbors (in G) in the same connected component of $G[W]$ and $u \in R$.*

Case 3.A. *v_1 and v_2 have neighbors (in G) in the same connected component of $G[W]$ and $u \in R$. In this case, $G[W \cup \{u\} \cup C_1 \cup C_2]$ contains (at least) one cycle, and u cannot belong to any solution. Therefore, we branch as follows.*

- **(i)** *A vertex from C_1 belongs to the solution. In this branch, we return $(G - C_1, H, k, W, R, \mathcal{C} \cup \{C_1\})$.*
- **(ii)** *A vertex from C_2 belongs to the solution. In this branch, we return $(G - C_2, H, k, W, R, \mathcal{C} \cup \{C_2\})$.*

Notice that in both the branches μ decreases by 1, and therefore, the resulting branching vector is $(1, 1)$.

Case 3.B. *v_1 and v_2 have neighbors (in G) in the same connected component of $G[W]$ and $u \notin R$. In this case, $G[W \cup \{u\} \cup C_1 \cup C_2]$ contains (at least) one cycle. We branch as follows.*

- **(i)** *u belongs to the solution. In this branch, we return $(G - \{u\}, H - \{u\}, k - 1, W, R \cup N_H(u), \mathcal{C})$.*
- **(ii)** *A vertex from C_1 belongs to the solution. In this branch, we return $(G - C_1, H, k, W, R, \mathcal{C} \cup \{C_1\})$.*
- **(iii)** *A vertex from C_2 belongs to the solution. In this branch, we return $(G - C_2, H, k, W, R, \mathcal{C} \cup \{C_2\})$.*

Notice that in all the three branches μ decreases by 1, and therefore, the resulting branching vector is $(1, 1, 1)$.

Case 3.C. *If v_1 and v_2 have neighbors in different connected components of $G[W]$ and $u \in R$. In this case, we branch as follows.*

- **(i)** *A vertex from C_1 belongs to the solution. In this branch, we return $(G - C_1, H, k, W, R, \mathcal{C} \cup \{C_1\})$.*

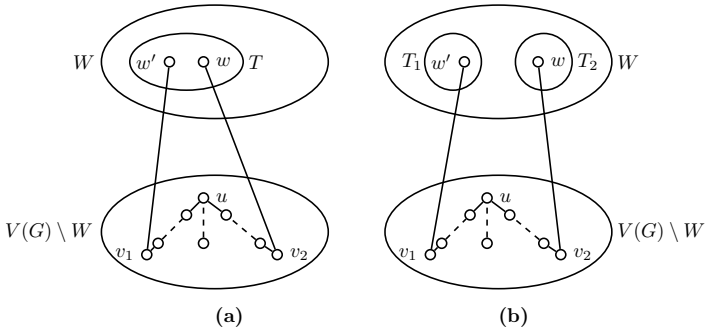


Figure 2 The cases handled by Branching Rule 3, In (a) T is a connected component in $G[W]$, similarly in (b) T_1, T_2 are connected components in $G[W]$.

(ii) *A vertex from* C_2 *belongs to the solution. In this branch, we return* $(G - C_2, H, k, W, R, \mathcal{C} \cup \{C_2\})$.

(iii) *No vertex from* $C_1 \cup C_2$ *belongs to the solution. In this case, we add all vertices in* $\{u\} \cup C_1 \cup C_2$ *to* W. *That is, the resulting instance is* $(G, H, k, W \cup (\{u\} \cup C_1 \cup C_2), R \setminus (\{u\} \cup C_1 \cup C_2), \mathcal{C})$.

Notice that in all the three branches μ *decreases by* 1, *and therefore, the resulting branching vector is* $(1, 1, 1)$.

Case 3.D. *If* v_1 *and* v_2 *have neighbors in different connected components of* $G[W]$ *and* $u \notin R$. *In this case, we branch as follows.*

(i) u *belongs to the solution. In this branch, we return* $(G - \{u\}, H - \{u\}, k - 1, W, R \cup N_H(u), \mathcal{C})$.

(ii) *A vertex from* C_1 *belongs to the solution. In this branch, we return* $(G - C_1, H, k, W, R, \mathcal{C} \cup \{C_1\})$.

(iii) *A vertex from* C_2 *belongs to the solution. In this branch, we return* $(G - C_2, H, k, W, R, \mathcal{C} \cup \{C_2\})$.

(iv) *No vertex from* $\{u\} \cup C_1 \cup C_2$ *belongs to the solution. In this case, we add all vertices in* $\{u\} \cup C_1 \cup C_2$ *to* W. *That is, the resulting instance is* $(G, H, k, W \cup (\{u\} \cup C_1 \cup C_2), R \setminus (\{u\} \cup C_1 \cup C_2), \mathcal{C})$.

Notice that in all the four branches μ *decreases by* 1, *and therefore, the resulting branching vector is* $(1, 1, 1, 1)$.

This completes the description of the algorithm. By showing the correctness of the presented algorithm, together with computation of the running time of the algorithm appropriately, we obtain the proof of Theorem 7.

5 FPT Algorithm for $K_{i,j}$-free+Cluster IS

In this section, we give an FPT algorithm for $K_{i,j}$-free+Cluster IS, which is the $\mathcal{F}$+Cluster IS where $\mathcal{F}$ is family of $K_{i,j}$-free graphs. Here, $i, j \in \mathbb{N}$, $1 \leq i \leq j$. In the following we consider a (fixed) family of $K_{i,j}$-free graphs. To design an FPT algorithm for $\mathcal{F}$+Cluster IS, we define another problem called Large $K_{i,j}$-free+Cluster IS. The problem Large $K_{i,j}$-free+Cluster IS is formally defined below.

LARGE $K_{i,j}$-free+CLUSTER IS	**Parameter:** k
Input: A $K_{i,j}$-free graph G, a cluster graph H (G and H are on the same vertex set), and an integer k, such that the following conditions are satisfied: 1) H has exactly k connected components, and 2) each connected component of H has at least k^k vertices. **Question:** Is there a set $S \subseteq V(G)$ of size k such that S is an independent set in both G and in H?	

In Section 5.1, we design a polynomial time algorithm for the problem LARGE $K_{i,j}$-free+CLUSTER IS. In the rest of this section, we show how to use the polynomial time algorithm for LARGE $K_{i,j}$-free+CLUSTER IS to obtain an FPT algorithm for $K_{i,j}$-free+CLUSTER IS.

▶ **Theorem 11.** *$K_{i,j}$-free+CLUSTER IS admits an FPT algorithm running in time $\mathcal{O}(k^{k^2} n^{\mathcal{O}(1)})$, where n is the number of vertices in the input graph.*

Proof. Let (G, H, k) be an instance of $K_{i,j}$-free+CLUSTER IS, and let $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ be the set of connected components in H. If $k \leq 0$, we can correctly resolve the instance in polynomial time (by appropriately outputting yes or no answer). Therefore, we assume $k \geq 1$. If for each $C \in \mathcal{C}$, we have $|V(C)| \geq k^k$, then (G, H, k) is also an instance of LARGE $K_{i,j}$-free+CLUSTER IS, and therefore we resolve it in polynomial time using the algorithm for LARGE $K_{i,j}$-free+CLUSTER IS (Section 5.1). Otherwise, there is $C \in \mathcal{C}$, such that $|V(C)| < k^k$. Any solution to $K_{i,j}$-free+CLUSTER IS in (G, H, k) must contain exactly one vertex from C. Moreover, if a vertex $v \in V(C)$ is in the solution, then none of its neighbors in G and in H can belong to the solution. Therefore, we branch on vertices in C as follows. For each $v \in V(C)$, create an instance $I_v(G - (N_H(v) \cup N_G(v)), H - (N_H(v) \cup N_G(v)), k-1)$ of $K_{i,j}$-free+CLUSTER IS. If number of connected components in $H - N[C]$ is less than $k-1$, then we call such an instance I_v as *invalid* instance, otherwise the instance is a *valid* instance. Notice that for $v \in V(C)$, if I_v is an invalid instance, then v cannot belong to any solution. Thus, we branch on valid instances of I_v, for $v \in V(C)$. Observe that (G, H, k) is a yes instance of $K_{i,j}$-free+CLUSTER IS if and only if there is a valid instance I_v, for $v \in V(C)$, which is a yes instance of $K_{i,j}$-free+CLUSTER IS. Therefore, we output the OR of results obtained by resolving valid instances I_v, for $v \in V(C)$.

In the above we have designed a recursive algorithm for the problem $K_{i,j}$-free+CLUSTER IS. In the following, we prove the correctness and claimed running time bound of the algorithm. We show this by induction on the measure $\mu = k$. For $\mu \leq 0$, the algorithm correctly resolve the instance in polynomial time. This forms the base case of our induction hypothesis. We assume that the algorithm correctly resolve the instance for each $\mu \leq \delta$, for some $\delta \in \mathbb{N}$. Next, we show that the correctness of the algorithm for $\mu = \delta + 1$. We assume that $k > 0$, otherwise, the algorithm correctly outputs the answer. The algorithm either correctly resolves the instance in polynomial time using the algorithm for LARGE $K_{i,j}$-free+CLUSTER IS, or applies the branching step. When the algorithm resolves the instance in polynomial time using the algorithm for LARGE $K_{i,j}$-free+CLUSTER IS, then the correctness of the algorithm follows from the correctness of the algorithm for LARGE $K_{i,j}$-free+CLUSTER IS. Otherwise, the algorithm applies the branching step. The branching is exhaustive, and the measure strictly decreases in each of the branches. Therefore, the correctness of the algorithm follows form the induction hypothesis. This completes the proof of correctness of the algorithm.

For the proof of claimed running time notice that the the worst case branching vector is is given by the k^k vector of all 1s, and at the leaves we resolve the instances in polynomial time. Thus, the claimed bound on the running time of the algorithm follows. ◀

5.1 Polynomial Time Algorithm for Large $K_{i,j}$-free+Cluster IS

Consider a (fixed) family of $K_{i,j}$-free graphs, where $1 \leq i \leq j$. The goal of this section is to design a polynomial time algorithm for Large $K_{i,j}$-free+Cluster IS. Let (G, H, k) be an instance of Large $K_{i,j}$-free+Cluster IS, where G is a $K_{i,j}$-free graph and H is a cluster graph with k connected components. We assume that $k > i + j + 2$, as otherwise, we can resolve the instance in polynomial time (using brute-force). Let $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ be the set of connected components in H, such that $|V(C_1)| \geq |V(C_2)| \geq \cdots \geq |V(C_k)|$.

We start by stating/proving some lemmata, which will be helpful in designing the algorithm.

▶ **Lemma 12.** [5] *The number of edges in a $K_{i,j}$-free graph are bounded by $n^{2-\epsilon}$, where $\epsilon = \epsilon(i,j) \in (0,1]$.*

▶ **Lemma 13.** *Let (G, H, k) be an instance of* Large $K_{i,j}$*-free+*Cluster IS*. There exists $v \in V(C_1)$, such that for each $C \in \mathcal{C} \setminus \{C_1\}$, we have $|N_G(v) \cap C| \leq \frac{2j|C|}{k}$.*

Proof. Consider a connected component $C \in \mathcal{C} \setminus \{C_1\}$, and let $x = |C_1|$ and $y = |C|$. Furthermore, let $E(C_1, C) = \{uv \in E(G) \mid u \in C_1, v \in V(C)\}$. In the following, we prove some claims which will be used to obtain the proof of the lemma.

▶ **Claim 14.** $|E(C_1, C)| \leq jy^i + jx$.

Proof. Consider the partition of $V(C_1)$ in two parts, namely, C_h^1 and C_ℓ^1, where $C_h^1 = \{v \in V(C_1) \mid |N_G(v) \cap V(C)| \geq i\}$ and $C_\ell^1 = V(C_1) \setminus C_h^1$.

$$|E(C_1, C)| = \sum_{v \in C_1} |N_G(v) \cap V(C)| = \sum_{v \in C_h^1} |N_G(v) \cap V(C)| + \sum_{v \in C_l^1} |N_G(v) \cap V(C)|.$$

By construction of C_ℓ^1, we have $\sum_{v \in C_\ell^1} |N_G(v) \cap V(C)| < ix$. In the following, we bound $\sum_{v \in C_h^1} |N_G(v) \cap V(C)|$. Since G is a $K_{i,j}$-free graph, therefore, any set of i vertices in $V(C)$ can have at most $j-1$ common neighbors (in G) from $V(C_1)$, and in particular from C_h^1. Moreover, every $v \in C_h^1$ has at least i neighbors in $N_G(v) \cap V(C)$. Therefore, $\sum_{v \in C_h^1} |N_G(v) \cap V(C)| \leq i(j-1)\binom{y}{i}$. Hence, $|E(C_1, C)| \leq i(j-1)\binom{y}{i} + ix \leq i(j-1)\frac{y^i}{i!} + ix \leq jy^i + jx$. ◀

Let $A_{\mathsf{deg}}(C_1, C)$ denote average degree of vertices in set C_1 in $G[E(C_1, C)]$. That is, $A_{\mathsf{deg}}(C_1, C) = \frac{|E(C_1, C)|}{|C_1|}$. In the following claim, we give a bound on $A_{\mathsf{deg}}(C_1, C)$.

▶ **Claim 15.** $A_{deg}(C_1, C) \leq \frac{2jy}{k^2}$.

Proof. From Claim 14, we have $|E(C_1, C)| \leq jy^i + jx$. Therefore, $A_{\mathsf{deg}}(C_1, C) \leq j + \frac{jy^i}{x}$. Using Lemma 12, we have $A_{\mathsf{deg}}(C_1, C) \leq \frac{(x+y)^{2-\epsilon}}{x} \leq 4x^{1-\epsilon}$. To prove the claim, us consider the following cases:

Case 1. $x \geq k^2 y^{i-1}$. In this case, using the inequality $A_{\mathsf{deg}}(C_1, C) \leq j + \frac{jy^i}{x}$, we have $A_{\mathsf{deg}}(C_1, C) \leq j + \frac{jy}{k^2}$. Since $y > k^2$ (and $k > 5$), we have $A_{\mathsf{deg}}(C_1, C) \leq \frac{2jy}{k^2}$.

Case 2. $x < k^2 y^{i-1}$. In this case, we use the inequality $A_{\mathsf{deg}}(C_1, C) \leq 4x^{1-\epsilon}$, to obtain $A_{\mathsf{deg}}(C_1, C) < 4k^{2(1-\epsilon)} y^{(i-1)(1-\epsilon)} < \frac{4k^2 y}{y^{(2-i)+\epsilon(i-1)}}$. Since $y \geq k^k$, we have $y^{(2-i)+\epsilon(i-1)} > \frac{2k^4}{j}$. Therefore, we have $A_{\mathsf{deg}}(C_1, C) < \frac{2jy}{k^2}$. ◀

Algorithm 1 (G, H, k): Greedy algorithm for Large $K_{i,j}$-free+Cluster IS.

1: $t = k$ and $S = \emptyset$;
2: **while** $t > 2j$ **do**
3: Let $C_1, \cdots, C_t$ be the connected components of H, sorted in decreasing order of their sizes;
4: Let $v \in V(C_1)$ be a vertex which satisfies the condition of Lemma 13;
5: Add v to S;
6: Decrease t by 1;
7: $G = G - (N_G(v) \cup N_H[v])$ and $H = H - (N_G(v) \cup N_H[v])$;
8: **end while**
9: Solve (G, H, t) by a brute force algorithm, as $t \leq 2j$;

In the following, we will give a probabilistic argument on the existence of a vertex with the desired properties in the lemma statement. For $v \in V(C_1)$, let $\mathsf{deg}(v, C)$ denote the size of $|N_G(v) \cap V(C)|$. From Claim 15, we have $A_{\mathsf{deg}}(C_1, C) \leq \frac{2jy}{k^2}$. Using Markov's inequality, the upper bound on the probability that $\mathsf{deg}(v, C) \geq \frac{2jy}{k}$ is $P(\mathsf{deg}(v, C) \geq \frac{2jy}{k}) \leq \frac{1}{k}$. Using Boole's inequality (the union bound), the probability that $\mathsf{deg}(v, C)$ is greater than or equal to $\frac{2j|C|}{k}$ for at least one $C \in \mathcal{C} \setminus \{C_1\}$ is bounded by $P(\cup_{C \in \mathcal{C} \setminus \{C_1\}} \mathsf{deg}(v, C) \geq \frac{2j|C|}{k}) \leq \frac{1}{k}.(k-1) < 1$. This implies that probability that $\mathsf{deg}(v, C) \leq \frac{2j|C|}{k}$, for each $C \in \mathcal{C} \setminus \{C_1\}$ is greater than 0. This completes the proof. ◀

We are now ready to describe our algorithm, which is given in Algorithm 1.

▶ **Lemma 16.** *Algorithm 1 for* Large $K_{i,j}$*-free+*Cluster IS *is correct and runs in polynomial time.*

Proof. We first prove the correctness of the algorithm using induction on, t. The base case is when $1 \leq t \leq 2j$. The algorithm correctly resolve the instance using brute force. For the induction hypothesis, we assume that the algorithm is correct for each $t \leq d-1$. Next, we show that the algorithm is correct for $t = d$. Let $C_1, \cdots, C_d$ be the set of connected components in H, sorted in decreasing order of their sizes. By Lemma 13, there is $v \in C_1$, such that for each $C \in \mathcal{C} \setminus \{C_1\}$, we have $\mathsf{deg}(v, C) \leq \frac{2j|C|}{d}$.

We delete all vertices in $N_H[v] \cup N_G(v)$ from G and H. Observe that from each $C \in \mathcal{C} \setminus \{C_1\}$, we have deleted at most $\frac{2j|C|}{d}$ vertices, which are neighbors of v in G. Let $C' = C \setminus (N_H[v] \cup N_G(v)) = C \setminus N_G(v)$. It is enough to show that $|C'| \geq (d-1)^{(d-1)}$. Note that $|C'| \geq |C| - \frac{2j|C|}{d}$. As base case is not applicable, we can assume that $d > 2j$. Hence, $|C'| \geq |C|(1 - \frac{2j}{d}) \geq d^d(1 - \frac{2j}{d}) \geq d^{d-1}(d - 2j) \geq (d-1)^{(d-1)}$.

This concludes the proof of correctness of the algorithm. At each step we either sort the components on the basis of their size or find a vertex of lower degree which can be carried out in polynomial time, or solve the instance using brute force approach, where the solution size we are seeking for is bounded by a constant (at most $2j$). Moreover, the algorithm terminates after at most k iterations. Thus, the running time of the algorithm is bounded by a polynomial in the size of the input. ◀

Using Lemma 16, we obtain the following theorem.

▶ **Theorem 17.** *The problem* Large $K_{i,j}$*-free+*Cluster IS *admits a polynomial time algorithm.*

References

1. Akanksha Agrawal, Sushmita Gupta, Saket Saurabh, and Roohani Sharma. Improved algorithms and combinatorial bounds for independent feedback vertex set. In *IPEC*, volume 63 of *LIPIcs*, pages 2:1–2:14, 2016.
2. Akanksha Agrawal, Sudeshna Kolay, Daniel Lokshtanov, and Saket Saurabh. A faster FPT algorithm and a smaller kernel for block graph vertex deletion. In *LATIN*, volume 9644 of *LNCS*, pages 1–13, 2016.
3. Akanksha Agrawal, Daniel Lokshtanov, Amer E. Mouawad, and Saket Saurabh. Simultaneous feedback vertex set: A parameterized perspective. In *STACS*, pages 7:1–7:15, 2016.
4. Matthias Bentert, René van Bevern, and Rolf Niedermeier. (wireless) scheduling, graph classes, and c-colorable subgraphs. *CoRR*, abs/1712.06481, 2017. URL: `http://arxiv.org/abs/1712.06481`.
5. Béla Bollobás. *Extremal graph theory*. Courier Corporation, 2004.
6. Leizhen Cai and Junjie Ye. Dual connectedness of edge-bicolored graphs and beyond. In *MFCS*, volume 8635, pages 141–152, 2014.
7. 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.
8. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
9. Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
10. Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical computer science*, 410(1):53–61, 2009.
11. Zoltán Füredi. On the number of edges of quadrilateral-free graphs. *Journal of Combinatorial Theory, Series B*, 68(1):1–6, 1996.
12. Pallavi Jain, Lawqueen Kanesh, and Pranabendu Misra. Conflict free version of covering problems on graphs: Classical and parameterized. In *CSR*, pages 194–206, 2018.
13. Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Information Processing Letters*, 114(10):556–560, 2014.
14. Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, and Saket Saurabh. On parameterized independent feedback vertex set. *Theoretical Computer Science*, 461:65–75, 2012.
15. Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. FPT algorithms for connected feedback vertex set. *Journal of Combinatorial Optimization*, 24(2):131–146, 2012.
16. Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.
17. Jan Arne Telle and Yngve Villanger. FPT algorithms for domination in biclique-free graphs. In *ESA*, pages 802–812, 2012.
18. René van Bevern, Matthias Mnich, Rolf Niedermeier, and Mathias Weller. Interval scheduling and colorful independent sets. *Journal of Scheduling*, 18(5):449–469, 2015.

Largest Weight Common Subtree Embeddings with Distance Penalties

Andre Droschinsky

Department of Computer Science, TU Dortmund University
Otto-Hahn-Str. 14, 44221 Dortmund, Germany
andre.droschinsky@tu-dortmund.de
https://orcid.org/0000-0002-7983-3739

Nils M. Kriege

Department of Computer Science, TU Dortmund University
Otto-Hahn-Str. 14, 44221 Dortmund, Germany
nils.kriege@tu-dortmund.de
https://orcid.org/0000-0003-2645-947X

Petra Mutzel

Department of Computer Science, TU Dortmund University
Otto-Hahn-Str. 14, 44221 Dortmund, Germany
petra.mutzel@tu-dortmund.de
https://orcid.org/0000-0001-7621-971X

Abstract

The largest common embeddable subtree problem asks for the largest possible tree embeddable into two input trees and generalizes the classical maximum common subtree problem. Several variants of the problem in labeled and unlabeled rooted trees have been studied, e.g., for the comparison of evolutionary trees. We consider a generalization, where the sought embedding is maximal with regard to a weight function on pairs of labels. We support rooted and unrooted trees with vertex and edge labels as well as distance penalties for skipping vertices. This variant is important for many applications such as the comparison of chemical structures and evolutionary trees. Our algorithm computes the solution from a series of bipartite matching instances, which are solved efficiently by exploiting their structural relation and imbalance. Our analysis shows that our approach improves or matches the running time of the formally best algorithms for several problem variants. Specifically, we obtain a running time of $\mathcal{O}(|T|\,|T'|\Delta)$ for two rooted or unrooted trees T and T', where $\Delta = \min\{\Delta(T), \Delta(T')\}$ with $\Delta(X)$ the maximum degree of X. If the weights are integral and at most C, we obtain a running time of $\mathcal{O}(|T|\,|T'|\sqrt{\Delta}\log(C\min\{|T|,|T'|\}))$ for rooted trees.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases maximum common subtree, largest embeddable subtree, topological embedding, maximum weight matching, subtree homeomorphism

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.54

Funding This work was supported by the German Research Foundation (DFG), priority programme "Algorithms for Big Data" (SPP 1736).

1 Introduction

The maximum common subgraph problem asks for a graph with a maximum number of vertices that is isomorphic to induced subgraphs of two input graphs. This problem arises in many domains, where it is important to find the common parts of objects which can be represented as graphs. An example of this are chemical structures, which can be interpreted directly as labeled graphs. Therefore, the problem has been studied extensively in cheminformatics [5, 16, 17]. Although elaborated backtracking algorithms have been developed [16, 2], solving large instances in practice is a great challenge. The maximum common subgraph problem is NP-hard and remains so even when the input graphs are restricted to trees [6]. However, in trees it becomes polynomial-time solvable when the common subgraph is required to be connected, i.e., it must be a tree itself. This problem is then referred to as *maximum common subtree problem* and the first algorithm solving it in polynomial-time is attributed to J. Edmonds [12]. Also requiring that the common subgraph must be connected (or even partially biconnected) several extensions to tree-like graphs have been proposed, primarily for applications in cheminformatics [19, 17, 3]. Some of these approaches are not suitable for practical applications due to high constants hidden in the polynomial running time. Other algorithms are efficient in practice, but restrict the search space to specific common subgraphs. Instead of developing maximum common subgraph algorithms for more general graph classes, which has proven difficult, a different approach is to represent molecules simplified as trees [15]. Then, vertices typically represent groups of atoms and their comparison requires to score the similarity of two vertices by a weight function. This, however, is often not supported by algorithms for tree comparison. Moreover, it maybe desirable to map a path in one tree to a single edge in the other tree, skipping the inner vertices. Formally, this is achieved by *graph homeomorphism* instead of isomorphism.

Various variants for comparing trees have been proposed and investigated [18]. Most of them assume rooted trees, which may be ordered or unordered. Algorithms tailored to the comparison of evolutionary trees typically assume only the leaves to be labeled, while others support labels on all vertices or do not consider labels at all. The well-known agreement subtree problem, for example, considers the case, where only the leaf nodes are labeled, with no label appearing more than once per tree [11]. We discuss the approaches most relevant for our work. Gupta and Nishimura [8] investigated the *largest common embeddable subtree* problem in unlabeled rooted trees. Their definition is based on topological embedding (or homeomorphism) and allows to map edges of the common subtree to vertex-disjoint paths in the input trees. The algorithm uses the classical idea to decompose the problem into subproblems for smaller trees, which are solved via bipartite matching. A solution for two trees with at most n vertices is computed in time $\mathcal{O}(n^{2.5} \log n)$ using a dynamic programming approach. Fig. 1a illustrates the difference between maximum common subgraph and largest common embeddable subtree. Lozano and Valiente [10] investigated the *maximum common embedded subtree* problem, which is based on edge contraction. In both cases the input graphs are rooted unlabeled trees. Note, the definition of their problems is not equivalent. The first is polynomial time solvable, while the second is NP-hard for unordered trees, but polynomial time solvable for ordered trees. Many algorithms do not support trees, where leaves and the inner vertices both have labels. A notable exception is the approach by Kao et al. [9], where only vertices with the same label may be mapped. This algorithm generalizes the approach by Gupta/Nishimura and improves its running time to $\mathcal{O}(\sqrt{d}D \log \frac{2n}{d})$, where D denotes the number of vertex pairs with the same label and d the maximum degree of all vertices.

We consider the problem of finding a *largest weight common subtree embedding* (LaWeCSE), where matching vertices are not required to have the same label, but their degree of agreement is determined by a weight function. We build on the basic ideas of Gupta and Nishimura [8].

To prevent arbitrarily long paths which are mapped to a common edge we study a linear distance penalty for paths of length greater than 1. Note that, by choosing a high distance penalty, we solve the maximum common subtree (MCS) problem as a special case. By choosing weight 1 for equal labels and sufficiently small negative weights otherwise, we solve a problem equivalent to the one studied by Kao et al. [9].

Our contribution. We propose and analyze algorithms for finding largest weight common subtree embeddings. Our method requires to solve a series of bipartite matching instances as subproblem, which dominates the total running time. We build on recent results by Ramshow and Tarjan [13, 14] for unbalanced matchings. Let T and T' be labeled rooted trees with $k := |T|$ and $l := |T'|$ vertices, respectively, and $\Delta := \min\{\Delta(T), \Delta(T')\}$ the smaller degree of the two input trees. For real-valued weight functions we prove a time bound of $\mathcal{O}(kl\Delta)$. For integral weights bounded by a constant C we prove a running time of $\mathcal{O}(kl\sqrt{\Delta}\log(\min\{k,l\}C))$. This is an improvement over the algorithm by Kao et al. [9] if there are only few labels and the maximum degree of one tree is much smaller than the maximum degree of the other. In addition, we support weights and a linear penalty for skipped vertices.

Moreover, the algorithm by Kao et al. [9] is designed for rooted trees only. A straight forward approach to solve the problem for unrooted trees is to try out all pairs of possible roots, which results in an additional $\mathcal{O}(kl)$ factor. However, our algorithm exploits the fact that there are many similar matching instances using techniques related to [1, 4]. This includes computing additional matchings of cardinality two. For unrooted trees and real-valued weight functions we prove the same $\mathcal{O}(kl\Delta)$ time bound as for rooted trees. This leads to an improvement over the formally best algorithm for solving the maximum common subtree problem, for which a time bound of $\mathcal{O}(kl\,(\Delta + \log d))$ has been proven [4].

2 Preliminaries

We consider finite simple undirected graphs unless stated otherwise. Let $G = (V, E)$ be a graph, we refer to the set of *vertices* V by $V(G)$ or V_G and to the set of *edges* by $E(G)$ or E_G. An edge connecting two vertices $u, v \in V$ is denoted by uv or vu. The *order* $|G|$ of a graph G is its number of vertices. The *neighbors* of a vertex v are defined as $N(v) := \{u \in V_G \mid vu \in E_G\}$. The *degree* of a vertex $v \in V_G$ is $\delta(v) := |N(v)|$, the *degree* $\Delta(G)$ of a graph G is the maximum degree of its vertices. In case of a directed graph (*digraph*) we call its edges *arcs*, denoted by (u, v), i.e., an edge from u to v.

A *path* P is a sequence of pairwise disjoint vertices connected through edges (or arcs) and denoted as $P = (v_0, e_1, v_1, \ldots, e_l, v_l)$, where $e_i = v_{i-1}v_i$ (or $e_i = (v_{i-1}, v_i)$), $i \in \{1, \ldots, l\}$. We alternatively specify the vertices $(v_0, \ldots, v_l)$ or edges $(e_1, \ldots, e_l)$ only. The *length* of a path is its number of edges. A connected graph with a unique path between any two vertices is a *tree*. A tree T with an explicit root vertex $r \in V(T)$ is called *rooted* tree, denoted by T^r. In a rooted tree T^r we denote the set of *children* of a vertex v by $C(v)$ and its *parent* by $p(v)$, where $p(r) = r$. For any tree T and two vertices $u, v \in V(T)$ the *rooted subtree* T^u_v is induced by the vertex v and its descendants related to the tree T^u. If the root r is clear from the context, we may abbreviate $T_v := T^r_v$. We refer to the root of a rooted tree T by $r(T)$.

If the vertices of a graph G can be separated into exactly two disjoint sets V, U such that $E(G) \subseteq V \times U$, then the graph is called *bipartite*. In many cases the disjoint sets are already given as part of the input. In this case we write $G = (V \sqcup U, E)$, where $E \subseteq V \times U$.

For a graph $G = (V, E)$ a *matching* $M \subseteq E$ is a set of edges, such that no two edges share a vertex. An edge $e \in M$ is denoted *matched*. A vertex incident to an edge $e \in M$ is denoted *matched*; otherwise it's *free*. For an edge $uv \in M$, the vertex u is the *partner* of v and vice versa. The *cardinality of a matching* M is its number of edges $|M|$. A *weighted graph* is a graph endowed with a function $w : E \to \mathbb{R}$. The weight of a matching M in a weighted graph is $W(M) := \sum_{e \in M} w(e)$. We call a matching M of a weighted graph G a *maximum weight matching* (MWM) if there is no other matching M' of G with $W(M') > W(M)$. A matching M in G is a *MWM of cardinality* k (MWM_k) if there is no other matching M' of cardinality k in G with $W(M') > W(M)$.

For convenience we define the maximum of an empty set as $-\infty$.

3 Gupta and Nishimura's algorithm

In this section we formally define a *Largest Common Subtree Embedding* (LaCSE) and present a brief overview of Gupta and Nishimura's algorithm to compute such an embedding. The following two definitions are based on [8].

▶ **Definition 1** (Topological Embedding)**.** A rooted tree T is *topologically embeddable* in a rooted tree T' if there is an injective function $\psi : V(T) \to V(T')$, such that $\forall a, b, c \in V(T)$

i) If b is a child of a, then $\psi(b)$ is a descendant of $\psi(a)$.

ii) For distinct children b, c of a, the paths from $\psi(a)$ to $\psi(b)$ and from $\psi(a)$ to $\psi(c)$ have exactly $\psi(a)$ in common.

T is *root-to-root topologically embeddable* in T', if $\psi(r(T)) = r(T')$.

▶ **Definition 2** ((Largest) Common Subtree Embedding; (La)CSE)**.** Let T and T' be rooted trees and S be topologically embeddable in both T and T'. For such a S let $\psi : V(S) \to V(T)$ and $\psi' : V(S) \to V(T')$ be topological embeddings.

- Then $\varphi := \psi' \circ \psi^{-1} : \psi(V_S) \to \psi'(V_S)$ is a *Common Subtree Embedding*.
- If there is no other tree S' topologically embeddable in both T and T' with $|S'| > |S|$, then S is a *Largest Common Embeddable Subtree* and φ is a *Largest Common Subtree Embedding*.
- An CSE with $\varphi(r(T)) = r(T')$ is a *root-to-root CSE*.
- A root-to-root CSE is *largest*, if it is of largest weight among all root-to-root common subtree embeddings.

Algorithm from Gupta and Nishimura. Gupta and Nishimura [8] presented an algorithm to compute the size of a largest common embeddable subtree based on dynamic programming, which is similar to the computation of a largest common subtree, described in, e.g., [12, 4]. Let T and T' be rooted trees and $\mathcal{L}$ be a table of size $|T||T'|$. For each pair of vertices $u \in T, v \in T'$ the value $\mathcal{L}(u, v)$ stores the size of a LaCSE between the rooted subtrees T_u and T'_v. Gupta and Nishimura proved, that an entry $\mathcal{L}(u, v)$ is determined by the maximum of the following three quantities.

- $M_1 = \max\{\mathcal{L}(u, c) \mid c \in C(v)\}$
- $M_2 = \max\{\mathcal{L}(b, v) \mid b \in C(u)\}$
- $M_3 = W(M)+1$, where M is a MWM of the complete bipartite graph $(C(u) \sqcup C(v), C(u) \times C(v))$ with edge weight $w(bc) = \mathcal{L}(b, c)$ for each pair $(b, c) \in C(u) \times C(v)$.

Here, M_1 represents the case, where the vertex v is not mapped. To satisfy ii) from Def. 1, we may map at most one child $c \in C(v)$. M_2 represents the case, where u is not mapped and at most one child $b \in C(u)$ is allowed. M_3 represents the case $\varphi(u) = v$. To maximize

the number of mapped descendants we compute a maximum weight matching, where the children of u and v are the vertex sets $C(u)$ and $C(v)$, respectively, of a bipartite graph. The edge weights are determined by the previously computed solutions, i.e., the LaCSEs between the children of u and v and their descendants, namely between T_b and T'_c for each pair of children (b, c). The algorithm proceeds from the leaves to the roots. From the above recursive formula, we get $\mathcal{L}(u, v) = 1$ if u or v is a leaf, which was separately defined in [8].

A maximum value in the table yields the size of a LaCSE. We obtain the size of a root-to-root LaCSE from M_3 of the root vertices $r(T), r(T')$. Note, with storing $\mathcal{O}(|\mathcal{L}|)$ additional data, it is easy to obtain a (root-to-root) LaCSE φ.

▶ **Theorem 3** (Gupta, Nishimura, [8]). *Computing a LaCSE between two rooted trees of order at most n is possible in time $\mathcal{O}(n^{2.5} \log n)$.*

4 Largest Weight Common Subtree Embeddings

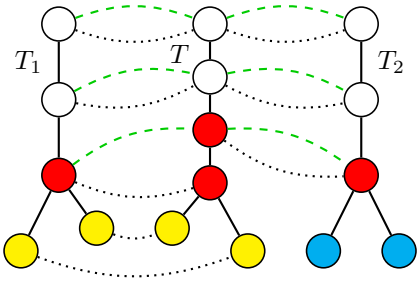
First, we introduce weighted common subtree embeddings between labeled trees. Part of the input is an integral or real weight function on all pairs of the labels. Next, we consider a linear distance penalty for skipped vertices in the input trees. After formalizing the problem and presenting an algorithm, we prove new upper time bounds.

Vertex Labels. In many application domains the vertices of the trees need to be distinguished. A common representation of a vertex labeled tree T is (T, l), where $l : V(T) \to \Sigma$ with Σ as a finite set of labels. Let $\omega : \Sigma \times \Sigma \to \mathbb{R} \cup \{-\infty\}$ assign a weight to each pair of labels. Instead of maximizing the number of mapped vertices, we want to maximize the sum of the weights $\omega(l(u), l(\varphi(u)))$ of all vertices u mapped by a common subtree embedding φ. For simplicity, we will omit l and l' for the rest of this paper and define $\omega(u, v) := \omega(l(u), l'(v))$ for any two vertices $(u, v) \in V(T) \times V(T')$.

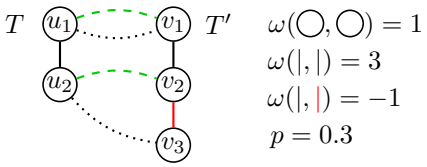
Edge Labels. Although not as common as vertex labels, edge labels are useful to represent different bonds between atoms or relationship between individuals. In a common subtree embedding we do not map edges to edges but paths to paths. Since in an embedding inner vertices on mapped paths do not contribute to the weight, we do the same with edges. I.e., both paths need to have length 1 for their edge labels to be considered. Here again, we want to maximize the weight $\omega(e, e') := \omega(l(e), l'(e'))$ of these edges mapped to each other (additional to the weight of the mapped vertices).

Distance Penalties. Depending on the application purpose it might be desirable that paths do not have an arbitrary length. Here, we introduce a linear distance penalty for paths of length greater than 1. I.e., each inner vertex on a path corresponding to an edge of the common embeddable subtree lowers the weight by a given penalty p. By assigning p the value ∞ we effectively compute a maximum common subtree. The following definition formalizes a LaCSE under a weight function ω and a distance penalty p.

▶ **Definition 4** (Largest Weight Common Subtree Embedding; LaWeCSE). Let (T, l) and (T', l') be rooted vertex and/or edge labeled trees. Let φ be a common subtree embedding between T and T'. Let $\omega : \Sigma \times \Sigma \to \mathbb{R} \cup \{-\infty\}$ assign a weight to each pair of labels. Let $p \in \mathbb{R}^{\geq 0} \cup \{\infty\}$ be a distance penalty. We refer to a path $P = (u_0, e_1, u_1, \ldots, u_k)$ in the tree T corresponding to a single edge in the common embeddable subtree as *topological path*. Let $\varphi(P)$ be the corresponding path $(v_0, e'_1, v_1, \ldots, v_l)$ in T'. Then



(a) Labeled MCS (green dashed lines) and LaCSE (black dotted lines) between T and T_i, $i \in \{1, 2\}$.



(b) The black embedding has weight 1.7, since the vertex v_2 is skipped and therefore the penalty p is applied; the weight between the edges is not added. The green embedding has weight 5; 2 from the vertices, 3 from the topological path (u_1, u_2) mapped to (v_1, v_2).

Figure 1 a) Although 'intuitively' T is more similar to T_1 than to T_2, both MCSs have size 3. However, the LaCSE between T and T_1 has 6 mapped vertices. **b)** Two weighted embeddings; one with a skipped vertex, the other where the edge labels contribute to the weight.

- $\omega_{\mathrm{p}}(P, \varphi(P)) = \omega(e_1, e_1') := \omega(l(e_1), l'(e_1'))$, if $l = k = 1$, or
- $\omega_{\mathrm{p}}(P, \varphi(P)) = -p \cdot (l + k - 2)$, otherwise.
- The *weight* $\mathcal{W}(\varphi)$ is the sum of the weights $\omega(u, \varphi(u))$ of all vertices u mapped by φ plus the weights $\omega_{\mathrm{p}}(P, \varphi(P))$ of all topological paths P.
- If φ is of largest weight among all common subtree embeddings, then φ is a *Largest Weight Common Subtree Embedding* (LaWeCSE).

The definition of root-to-root LaWeCSE is analogue to Def. 2. A closer look at the definition of ω_{p} reveals that each inner vertex (the skipped vertices) on a topological path or its mapped path subtracts p from the embedding's weight. Fig. 1b illustrates two weighted common subtree embeddings.

The dynamic programming approach. To compute a LaWeCSE, we need to store some additional data during the computation. In Gupta and Nishimura's algorithm there is a table $\mathcal{L}$ of size $|T||T'|$ to store the weight of LaCSEs between subtrees of the input trees. In our algorithm we need a table $\mathcal{L}$ of size $2|T||T'|$. An entry $\mathcal{L}(u, v, t)$ stores the weight of a LaWeCSE between the rooted subtrees T_u and T_v' of *type* $t \in \{\curlywedge, \diamond\}$. Type $\curlywedge$ represents a *root-to-root* embedding between T_u and T_v'; $\diamond$ an embedding, where u or v is *skipped*. Skipped in the sense, that at least one of u, v will be an inner vertex when mapping some additional ancestor nodes of u or v during the dynamic programming. For type $\diamond$ we subtract the penalty p from the weight for the skipped vertices before storing it in our table. We obtain the weight of a LaWeCSE and a root-to-root LaWeCSE, respectively, from the maximum value of type $\curlywedge$ and from $\mathcal{L}(r(T), r(T'), \curlywedge)$, respectively. The following lemma specifies the recursive computation of an entry $\mathcal{L}(u, v, t)$.

▶ **Lemma 5.** *Let $u \in V(T)$ and $v \in V(T')$. For $t \in \{\curlywedge, \diamond\}$ let $M_t^T = \max\{\mathcal{L}(b, v, t) \mid b \in C(u)\}$ and $M_t^{T'} = \max\{\mathcal{L}(u, c, t) \mid c \in C(v)\}$. Then*

- *$\mathcal{L}(u, v, \diamond) = \max\{M_\diamond^T, M_\curlywedge^T, M_\diamond^{T'}, M_\curlywedge^{T'}\} - p$*

Let $G = (C(u) \sqcup C(v), C(u) \times C(v))$ be a bipartite graph with edge weights $w(bc) = \max\{\mathcal{L}(b, c, \diamond), \mathcal{L}(b, c, \curlywedge) + \omega(ub, vc)\}$ for each pair $(b, c) \in C(u) \times C(v)$. Then

- *$\mathcal{L}(u, v, \curlywedge) = \omega(u, v) + W(M)$, where M is a MWM on G.*

Proof. Since we defined the maximum of an empty set as $-\infty$, this covers the base case, where one vertex is a leaf, e.g., if u is a leaf, then $\max\{M_\diamond^T, M_\curlywedge^T, M_\diamond^{T'}, M_\curlywedge^{T'}\} = \max\{M_\diamond^{T'}, M_\curlywedge^{T'}\}$. Further, $W(M) = 0$, if G has no positive weight edges. Then $\mathcal{L}(u, v, \curlywedge) = \omega(u, v)$.

The type $\diamond$ represents the case of an embedding between T_u and T'_v which is not root-to-root. From the definition of M_t^T the vertex u is skipped and from the definition of $M_t^{T'}$ the vertex v is skipped, so it is indeed not root-to-root. Since either u or v was skipped, we subtract the penalty p. This ensures we have taken inner vertices of later steps of the dynamic programming into account.

The type $\curlywedge$ implies that u is mapped to v. Each edge in G represents the weight of a LaWeCSE from one child of u to one child of v. A maximum matching yields the best combination which satisfies Def. 1. If a child b of u is mapped to a child c of v, the paths bu and cv have length one. Then from Def. 4 we have to add the weight $\omega(bu, cv)$. Otherwise at least one path has length greater than one and we have to subtract the distance penalty p for each inner vertex. We already did that while computing $\mathcal{L}(b, c, \diamond)$. ◀

Time and space complexity. We next analyze upper time and space bounds. Thereby we distinguish between real- and integer-valued weight functions ω. If we use dynamic programming starting from the leaves to the roots, we need to compute each value $\mathcal{L}(u, v, t)$ only once.

▶ **Theorem 6.** *Let T and T' be rooted vertex and/or edge labeled trees. Let ω be a weight function, $\Delta = \min\{\Delta(T), \Delta(T')\}$, and p be a distance penalty.*

- *A LaWeCSE between T and T' can be computed in time $\mathcal{O}(|T|\,|T'|\Delta\}$ and space $\mathcal{O}(|T|\,|T'|)$.*
- *If the weights are integral and bounded by a constant C, a LaWeCSE can be computed in time $\mathcal{O}(|T|\,|T'|\sqrt{\Delta}\log(C\min\{|T|, |T'|\}))$.*

We first need to provide two results regarding maximum weight matchings.

▶ **Lemma 7** ([14])**.** *Let G be a weighted bipartite graph containing m edges and vertex sets of sizes s and t. W.l.o.g. $s \leq t$. We can compute a MWM on G in time $\mathcal{O}(ms + s^2 \log s)$ and space $\mathcal{O}(m)$. If G is complete bipartite, we may simplify the time bound to $\mathcal{O}(s^2 t)$.*

If the weights are integral and bounded by a constant, the following result for a minimum weight matching was shown in [7]. We solve MWM by multiplying the weights by -1.

▶ **Lemma 8.** *Let G as in Lemma 7 and the weights be integral and bounded by a constant C. We can compute a MWM on G in time $\mathcal{O}(m\sqrt{s}\log C)$. If G is complete bipartite, we may simplify the time bound to $\mathcal{O}(s^{1.5} t \log C)$.*

Unfortunately, there is no space bound given in [7]. However, since their algorithm is based on flows, the space bound is probably $\mathcal{O}(m)$. If we assume this to be correct, the space bound in Theorem 6 applies to integral weights too.

Proof of Theorem 6. We observe that the entries of type $\curlywedge$ in the table $\mathcal{L}$ dominate the computation time. We assume the bipartite graphs on which we compute the MWMs to be complete. We further observe that each edge bc representing the weight of the LaWeCSE between the subtrees T_b and T'_c is contained in exactly one of the matching graphs.

Let us assume real weights first. $\mathcal{L}$ requires $\mathcal{O}(|T|\,|T'|)$ space. We may also compute each MWM within the same space bound. This proves the total space bound. From Lemma 7 the

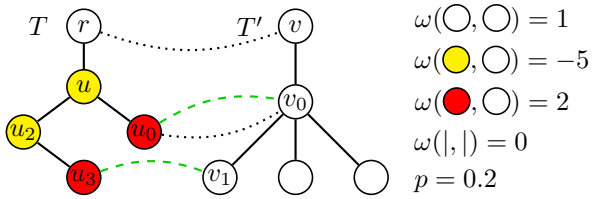


Figure 2 The weight of a LaWeCSE between T^r and T'^v is 2.8 (black dotted lines), since we have one skipped vertex u for penalty 0.2. The weight of a LaWeCSE between T^{u_0} and T'^{v_0} is 3.6 (green dashed lines) for two skipped vertices. The latter one is also a LaWeCSE_u.

time to compute all the MWMs is bounded by

$$\mathcal{O}\left(\sum_{v\in V_{T'}}\sum_{u\in V_T}|C(u)|\,|C(v)|\min\{|C(u)|,|C(v)|\}\right)$$
$$\subseteq \mathcal{O}\left(\sum_{v\in V_{T'}}|C(v)|\sum_{u\in V_T}|C(u)|\Delta\right) = \mathcal{O}(|T|\,|T'|\Delta).$$

Let us assume ω to be integral and bounded by a constant C next. This implies a weight of each single matching edge of at most $\bar{C} := 2C\cdot\min\{|T|,|T'|\}$, since no more than $2\min\{|T|,|T'|\}$ edges and vertices in total may contribute to the weight. Negative weight edges never contribute to a MWM and may safely be omitted. From Lemma 8 the time bound is

$$\mathcal{O}\left(\sum_{v\in V_{T'}}\sum_{u\in V_T}|C(u)|\,|C(v)|\sqrt{\min\{|C(u)|,|C(v)|\}}\log\bar{C}\right)$$
$$\subseteq \mathcal{O}\left(\sum_{v\in V_{T'}}|C(v)|\sum_{u\in V_T}|C(u)|\sqrt{\Delta}\log\bar{C}\right) = \mathcal{O}\left(|T|\,|T'|\sqrt{\Delta}\log(C\min\{|T|,|T'|\})\right). \quad \blacktriangleleft$$

5 Largest Weight Common Subtree Embeddings for Unrooted Trees

In this section we consider a LaWeCSE between unrooted trees. I.e., we want to find two root vertices $r\in V(T)$, $s\in V(T')$ and a common subtree embedding φ between T^r and T'^s such that there is no embedding φ' between $T^{r'}$ and $T'^{s'}$, $r'\in V(T)$, $s'\in V(T')$, with $\mathcal{W}(\varphi')>\mathcal{W}(\varphi)$. We abbreviate this as LaWeCSE_u. In Sect. 5.1 we present a basic algorithm and a first improvement by fixing the root of T. In Sect. 5.2 we speed up the computation by exploiting similarities between the different chosen roots of T'. In each section we prove the correctness and upper time bounds of our algorithms.

5.1 Basic algorithm and fixing one root

The basic idea is to compute for each pair of vertices $(u,v)\in V(T)\times V(T')$ a (rooted) LaWeCSE from T^u to T'^v and output a maximum solution. This is obviously correct and the time bound is $\mathcal{O}(|T|^2\,|T'|^2\Delta)$.

In our previous work [4] we showed how to compute a maximum common subtree between unrooted trees by arbitrarily choosing one root vertex r of T and then computing MCSs between T^r and T'^s for all $s\in V(T')$. The key idea in the proof is that for any maximum

common subtree isomorphism φ between T and T', either r is mapped by φ, or there is a unique vertex $u \in V(T)$ with shortest distance to r, such that all vertices mapped by φ are contained in T_u. The dynamic programming approach for LaWeCSE already considers the maximum solutions between the rooted subtrees of T^r and T'^s. However, Fig. 2 shows that this strategy alone sometimes fails, if we want to find a LaWeCSE_u between the input trees. A LaWeCSE between T^r and T' rooted at any vertex results in a weight of at most 2.8. In contrast, rooting T at u_0 results in a LaWeCSE of weight 3.6.

In a maximum common subtree between trees T and T', let $r \in V(T)$ be an arbitrarily chosen root of T. If any two children u_1, u_2 of their parent node $u \in V(T)$ are mapped to vertices of T', then u is also mapped. This statement is independent from the chosen root $r \in V(T)$, since a common subtree is connected. If we want to compute a (rooted) LaWeCSE, the statement is also true (for the given root). This follows from Def. 1 ii). However, if we choose u_1 as root in a LaWeCSE_u, we may skip u and map u_2, forming the topological path (u_1, u, u_2). Whatever we do, if we skip vertex u as an inner vertex of a topological path, this is the only path containing u; otherwise we violate Def. 1. We record this as a lemma.

▶ **Lemma 9.** *Let T and T' be unrooted trees. Let φ be a LaWeCSE_u from T to T' and $u \in V(T)$ be an inner vertex of a topological path with its neighbors $N(u) = \{u_1, u_2, \ldots, u_k\}$. Then φ maps vertices from exactly two of the rooted subtrees $T^u_{u_1}, \ldots, T^u_{u_k}$ to T'.*

To compute a LaWeCSE_u φ, additionally to the strategy from [4], we need to cover the case, that there is no single vertex u mapped by φ, such that all vertices mapped by φ are contained in T^r_u, cf. Fig. 2 with r as chosen root. In this case let u be the unique inner vertex of a topological path P, such that all vertices mapped by φ are contained in T^r_u. An example is the yellow vertex u in Fig. 2. Further, let $P_1 = (u_0, \ldots, u_{i-1}, u_i = u, u_{i+1}, \ldots, u_k)$ be the topological path containing u and $\varphi(P_1) = P_2 = (v_0, \ldots, v_l)$ with $\varphi(u_0) = v_0$ and $\varphi(u_k) = v_l$.

Then there is a LaWeCSE ϕ_1 between the rooted subtrees $T^r_{u_{i-1}}$ and $T'^{v_1}_{v_0}$ containing u_0 and all its descendants mapped by φ. There is another LaWeCSE ϕ_2 between the rooted subtrees $T^r_{u_{i+1}}$ and $T'^{v_0}_{v_1}$ containing u_k and all its descendants mapped by φ. Note, choosing $T'^{v_{j+1}}_{v_j}$ and $T'^{v_j}_{v_{j+1}}$ for any $j \in \{0, \ldots, l+1\}$ as rooted subtrees yields the same LaWeCSEs ϕ_1 and ϕ_2, i.e., it does not matter where we split the path P_2.

For any vertex $v \in V(T')$ let $\mathcal{L}^v$ refer to the table $\mathcal{L}$ corresponding to T^r and T'^v. Then $L_1 := \max\{\mathcal{L}^{v_1}(u_{i-1}, v_0, t) \mid t \in \{\curlywedge, \diamond\}\}$ is the weight of the LaWeCSE ϕ_1 minus the penalty for the inner vertices $u_1, \ldots, u_{i-1}$; the penalty is 0, if $i = 1$. $L_2 := \max\{\mathcal{L}^{v_0}(u_{i+1}, v_1, t) \mid t \in \{\curlywedge, \diamond\}\}$ is the weight of the LaWeCSE ϕ_2 minus the penalty for the inner vertices $u_{i+1}, \ldots, u_{k-1}$ and $v_1, \ldots, v_{l-1}$. I.e., the penalty p for each inner vertex on the paths excluding u is included in $L_1 + L_2$. Therefore $\mathcal{W}(\varphi) = L_1 + L_2 - p$. Before summarizing this strategy in the following Lemma 10, we exemplify it on Fig. 2.

The LaWeCSE_u φ is depicted by green dashed lines with mapping $u_0 \mapsto v_0$ and $u_3 \mapsto v_1$. The yellow inner vertex u fulfills the condition that T^r_u contains all vertices mapped by φ. We have paths $P_1 = (u_0, u, u_2, u_3)$ and $\varphi(P_1) = P_2 = (v_0, v_1)$. Then $L_1 = \mathcal{L}^{v_1}(u_0, v_0, \curlywedge) = 2$ for the mapping $u_0 \mapsto v_0$. Further $L_2 = \mathcal{L}^{v_0}(u_2, v_1, \diamond) = 1.8$ for the mapping $u_3 \mapsto v_1$ and the skipped vertex u_2. We obtain $\mathcal{W}(\varphi) = L_1 + L_2 - p = 2 + 1.8 - 0.2 = 3.6$.

▶ **Lemma 10.** *Let T and T' be trees. Let $r \in V(T)$ be arbitrarily chosen. Let $\mathcal{W}(r, v)$ be the weight of a LaWeCSE from T^r to T'^v and $\mathcal{T}(u, v, w) = \max\{\mathcal{L}^w(u, v, t) \mid t \in \{\curlywedge, \diamond\}\}$. Then the weight of a LaWeCSE_u is the maximum of the following two quantities.*

- $M_1 = \max\{\mathcal{W}(r, v) \mid v \in V(T')\}$
- $M_2 = \max_{u \in V(T),\ vw \in E(T')}\{\mathcal{T}(u_1, v, w) + \mathcal{T}(u_2, w, v) \mid u_1 \neq u_2 \in C(u)\} - p$

▶ **Lemma 11.** *Let the preconditions be as in Lemma 10. Then M_1 can be computed in time $\mathcal{O}(|T|\,|T'|^2\Delta)$ and M_2 in time $\mathcal{O}(|T|\,|T'|)$; both can be computed in space $\mathcal{O}(|T|\,|T'|)$.*

Proof. For any vertices $s, v \in V(T')$ consider the rooted subtree T'^s_v. Let $p(v)$ be the parent vertex of v with $p(s) = s$. Then $T'^s_v = T'^{p(v)}_v = T'^w_v$ for each vertex $w \in V(T') \setminus V(T'^s_v)$, i.e., w is a vertex of T', which is not contained in the rooted subtree T'^s_v. Therefore we can identify each table entry $\mathcal{L}^s(u, v, t)$ by $\mathcal{L}^{p(v)}(u, v, t)$. In other words, all the table entries needed to compute M_1 are determined first by a node $u \in V(T)$, and second by either an edge $wv \in E(T')$ or the root vertex s. Therefore, the space needed to store all the table entries and thus compute M_1 is $\mathcal{O}(|T|\,|T'|)$. We will use these values to retrieve $\mathcal{T}(u, v, w)$ in constant time.

From Theorem 6 for each $v \in V(T')$ we can compute $\mathcal{W}(r, v)$ in time $\mathcal{O}(|T|\,|T'|\Delta)$. Thus, the time for M_1 is bounded by $\mathcal{O}(|T|\,|T'|^2\Delta)$. For any edge $vw \in E(T')$ we observe that the only rooted subtrees from T' to consider are T'^v_w and T'^w_v. Let $L(b, v)$ and $L(b, w)$, $b \in C(u)$ be the weight of a LaWeCSE from T^r_b to T'^w_v and T'^v_w, respectively. Let G be a bipartite graph with vertices $C(u) \sqcup \{v, w\}$ and edges between these vertices with weights defined by $L(b, v)$ and $L(b, w)$, respectively. Let M be a MWM_2 on G. Then $W(M) = \max\{\mathcal{T}(u_1, v, w) + \mathcal{T}(u_2, w, v) \mid u_1 \neq u_2 \in C(u)\}$. This follows from the construction of G. Note, a MWM_2 contains exactly 2 edges. Let $C(u) = \{b_1, \ldots, b_k\}$ such that $L(b_i, v) \geq L(b_{i+1}, v)$ for any $i < k$. I.e., the vertices b_i are ordered, such that $L(b_1, v)$ and $L(b_2, v)$ have weight at least $L(b_i, v)$ for all $i > 2$. We remove all edges incident to v except b_1v and b_2v. Analog we remove all but the two edges of greatest weight incident to w. Let G' be the graph with those edges removed and M' be a MWM_2 on G'. We next prove $W(M') = W(M)$. Let M be a matching on G. Assume the partner of v is b_i, $i > 2$. Then let $b = b_1$ if b_1 is not the partner of w, and $b = b_2$ otherwise. Replacing $vb_i \in M$ by vb results in a matching M' such that $W(M') \geq W(M)$. We may argue analog for w.

Since G' contains at most 4 edges we may compute M' in constant time. The time to remove the edges from G to G' is $\mathcal{O}(k)$. Therefore the time to compute $\max\{\mathcal{T}(u_1, v, w) + \mathcal{T}(u_2, w, v) \mid u_1 \neq u_2 \in C(u)\}$ for given u and vw is $\mathcal{O}(|C(u)|)$. The time to compute M_2 is $\mathcal{O}(\sum_{u \in V_T, vw \in E_{T'}} |C(u)|) = \mathcal{O}(|T|\,|T'|)$.

We may compute M_2 from $\mathcal{L}$ and additional space $\mathcal{O}(|T|)$, which is $\mathcal{O}(|T|\,|T'|)$ in total. ◀

5.2 Exploiting similarities

In this section we improve the running time from $\mathcal{O}(|T|\,|T'|^2\Delta)$ to $\mathcal{O}(|T|\,|T'|\Delta)$. To this end, we need to speed up the computation in Lemma 5. Specifically, we exploit similarities between the graphs on which we compute the maximum weight matchings. We further need to speed up the computation of $M_t^{T'}$ related to the root vertices from T'. We have to take special care of the sequence, in which we compute the table entries, to avoid circular dependencies.

Speeding up the dynamic programming approach. In Lemma 5 the recursion computes maximum values among certain table entries. We first include the current root $s \in V(T')$ into the notation. We use the definition of $\mathcal{L}^s$ from Sect. 5.1 refering to the table where s is the root of $V(T')$. Let $u \in V(T)$ and $v \in V(T')$ be the vertices in the current recursion of Lemma 5. For all $t \in \{\curlywedge, \diamond\}$ let $M_t^{T,s} = \max\{\mathcal{L}^s(b, v, t) \mid b \in C(u)\}$ and $M_t^{T',s} = \max\{\mathcal{L}^s(u, c, t) \mid c \in C(v)\}$. From the proof of Lemma 11 we know that $T'^s_v = T'^w_v$

for each vertex $w \in V(T') \setminus V(T'^s_v)$. This implies $M_t^{T,s} = M_t^{T,w}$ and $M_t^{T',s} = M_t^{T',w}$ for all w as before. I.e., it is sufficient to distinguish all the $M_t^{T,s}$ and $M_t^{T',s}$ first by a node $u \in V(T)$, and second by either an edge $wv \in E(T')$ or a single vertex $s \in V(T')$.

This observation allows us to upper bound the time to compute all the $M_t^{T,s}$ by

$$\mathcal{O}\left(\sum_{u\in V_T, wv\in E_{T'}} |C(u)| + \sum_{u\in V_T, s\in V_{T'}} |C(u)|\right) = \mathcal{O}\left(\sum_{wv\in E_{T'}} |T| + \sum_{s\in V_{T'}} |T|\right) = \mathcal{O}\left(|T|\,|T'|\right).$$

Let $N(v) = \{c_1, c_2, \dots c_l\}$ and $C_i := \{c_1, \dots, c_{i-1}, c_{i+1}, \dots, c_l\}$ for $1 \le i \le l$, i.e, C_i contains all the vertices from $N(v)$ except c_i. We observe $M_t^{T',v} = \max\{\mathcal{L}^v(u,c,t) \mid c \in N(v)\}$ and $M_t^{T',c_i} = \max\{\mathcal{L}^{c_i}(u,c,t) \mid c \in C_i\} = \max\{\mathcal{L}^v(u,c,t) \mid c \in C_i\}$ for each $i \in \{1,\dots,l\}$. Let j be an index, such that $M_t^{T',v} = \mathcal{L}^v(u,c_j,t)$. Then for each $i \neq j$ we have $M_t^{T',c_i} = M_t^{T',v}$. Therefore, we may compute $M_t^{T',v}$ and M_t^{T',c_i} for all $i \in \{1,\dots,l\}$ in time $\mathcal{O}(\delta(v))$. Hence, the time to compute all the $M_t^{T',s}$ is bounded by $\mathcal{O}\left(\sum_{u\in V_T, v\in V_{T'}} \delta(v)\right) = \mathcal{O}\left(\sum_{u\in V_T} |T'|\right) = \mathcal{O}(|T|\,|T'|)$.

▶ **Lemma 12.** *Assume there is a sequence of all pairs (u,v) such that all necessary values are available to compute $M_t^{T,s}$ and $M_t^{T',s}$ for $s \in N(v) \cup \{v\}$; then the total time is $\mathcal{O}(|T|\,|T'|)$.*

Exploiting similarities between the matching graphs. In Lemma 5 we need to compute a MWM for each $(u,v) \in V(T) \times V(T')$. When considering all roots $s \in V(T')$, we have one matching graph G with vertices $C(u) \sqcup N(v)$, $N(v) = \{c_1, \dots, c_l\}$ as well as l graphs G_{c_i}, $1 \le i \le l$. This follows analog to the observation regarding $M_t^{T',v}$ from the previous paragraph. A graph G_c, $c \in N(v)$, is the same as G except that the vertex c and incident edges are removed. Let $s := \min\{\delta(u), \delta(v)\}$ and $t := \max\{\delta(u), \delta(v)\}$. We now prove a total time bound of $\mathcal{O}(s^2 t)$ for computing a MWM on G as well as on G_c for all $c \in N(v)$. We distinguish two cases.

i) $s \ge \log t$. In our previous work we presented an algorithm to compute a maximum common subtree of maximum weight, a special case of LaWeCSE_u [4]. A subproblem is to compute MWMs on graphs structurally identical to G and G_{c_i}, $1 \le i \le l$. We showed that we can compute all those MWMs in time $\mathcal{O}(st(s + \log t))$. Under the premise $s \ge \log t$ the time bound is $\mathcal{O}(s^2 t)$.

ii) $s < \log t$. Then one vertex set is much smaller than the other. From Lemma 14 we can compute all those MWMs in time $\mathcal{O}(s^4 + s^2 t)$. Under the premise $s < \log t$ that is $\mathcal{O}(s^2 t)$.

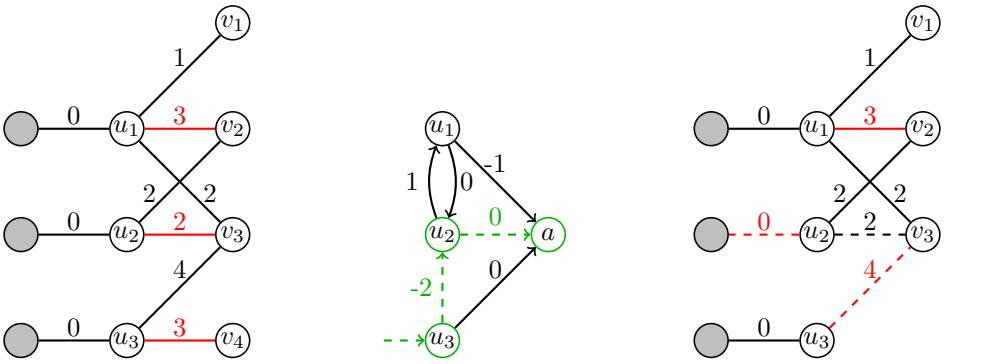
▶ **Lemma 13.** *Assume there is a sequence of all pairs (u,v) such that all necessary values are available to compute the MWMs; then the total time is $\mathcal{O}(|T|\,|T'|\Delta)$.*

Proof. The time to compute all the MWMs is

$$\mathcal{O}\left(\sum_{u\in V_T}\sum_{v\in V_{T'}} \delta(u)\delta(v)\min\{\delta(u),\delta(v)\}\right) \subseteq \mathcal{O}\left(\sum_{u\in V_T}\sum_{v\in V_{T'}} \delta(u)\delta(v)\Delta\right) = \mathcal{O}(|T|\,|T'|\Delta).$$ ◀

▶ **Lemma 14.** *Let G be a weighted bipartite graph with vertex sets U and V, $s := |U| \le |V| =: t$. Let either $C = U$ or $C = V$. We can compute a MWM on G and a MWM on each graph G_c, $c \in C$, in total time $\mathcal{O}(s^4 + s^2 t)$.*

Proof. From Lemma 7 we know there is an algorithm which computes a MWM on G in time $\mathcal{O}(s^2 t)$. This algorithm first copies the s vertices of U and then adds an edge of weight 0



(a) G' and a MWM M' (red).

(b) $\overline{G'_{v_4}}$ and a shortest path P from v_4's partner u_3 to a (green, dashed lines).

(c) A MWM (red) on G'_{v_4} obtained from a single augmenting path (dashed lines).

Figure 3 **(a)** A MWM$_s$, $s = |U| = 3$, on G', which is also a MWM. **(b)** The graph $\overline{G'_{v_4}}$, on which we compute a shortest path from u_3 to a. Such a path corresponds to a augmenting path of maximal weight in G'_{v_4}. **(c)** Applying the path yields a MWM$_3$ on G'_{v_4}, which is also a MWM.

between each vertex of U and its copy. We denote this graph by G'. The algorithm computes a MWM$_s$ M' on G' (M' is also a MWM), which corresponds to a MWM on G. An example is depicted in Figure 3a.

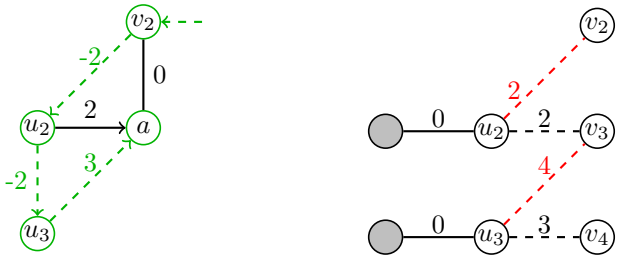
The graph G' with one vertex $c \in C$ removed is denoted by G'_c. If c is not matched, we are done. If c is matched, let u (in case $c \in V$) respectively v (in case $c \in U$) be the partner of c. Let $M'_c := M' \setminus \{cu\}$ or $M'_c := M' \setminus \{cv\}$, respectively. We observe $|M'_c| = s - 1$.

First, assume $c \in V$. In a MWM$_s$ of G'_c each vertex of U including u must be matched. An odd length M'_c-augmenting path P of maximal weight (the path's weight refers to the difference in the matching's weight after augmentation) incident to u yields a MWM$_s$ on G'_c and thus a MWM on G_c. This follows from the fact, that any M'_c-alternating cycle or path on G'_c not incident to u has nonpositive weight; otherwise M' was no MWM. We can find such a path using the Bellman-Ford algorithm in time $\mathcal{O}(st + s^3)$ as follows. Let $\overline{G'_c} = (U \cup \{a\}, A)$ be the digraph, where A is the union of the following two sets of directed arcs.

1. For each alternating path $\bar{u}vu'$ in G'_c, where $\bar{u}, u' \in U, v \in V$, and vu' is matched, we add the arc $(\bar{u}, u')$ with weight $w(vu') - w(\bar{u}v)$.
2. For each vertex $\bar{u} \in U$ let v be a free vertex adjacent to u, such that the edge uv has maximum weight among all such edges. We add an arc (u, a) of weight $-w(uv)$.

The time to construct the graph is bounded by $\mathcal{O}(st)$. Since $\overline{G'_c}$ has $\mathcal{O}(s)$ vertices and $\mathcal{O}(s^2)$ edges, we may compute a shortest path P from c's partner u to a in time $\mathcal{O}(s^3)$. We obtain a MWM on G'_c by augmenting M'_c with the edges that correspond to P in $\overline{G'_c}$. Figure 3b depicts an example for $\overline{G'_c}$, as well as a shortest path P. Figure 3c depicts the resulting MWM. Since at most s vertices of V are matched by M, the total time to compute a MWM on each of the graphs G_c, $c \in V$, is $\mathcal{O}(s^4 + s^2 t)$.

Second, assume $c \in U$. Each vertex in U is matched. Therefore the cardinality of M'_c is $s - 1$. This time we need to find an alternating path of even length (we removed a vertex from U) and of maximal weight incident to c's partner v. Any alternating cycle or path not incident to v cannot augment M'_c to greater weight; otherwise M' was no MWM. This path may have length 0, e.g., if M'_c is a MWM of M'. The total time to compute such a path is $\mathcal{O}(s^3)$ as follows. Let $\overline{G'_c} = (U \cup \{v, a\}, A)$, where A is the union of the following four sets of directed arcs.



(a) $\overline{G'_{u_1}}$ and a shortest path P from u_1's partner v_2 to a (green, dashed).

(b) A MWM (red) on G'_{u_1} (isolated vertices removed) obtained from a single augmenting path (dashed lines).

Figure 4 **(a)** The graph $\overline{G'_{u_1}}$, on which we compute a shortest path from v_2 to a. Such a path corresponds to an augmenting path of maximal weight in G'_{u_1}. **(b)** Applying the path yields a MWM_2 on G'_{u_1}, which is also a MWM.

1. For each edge $vu \in E(G'_c)$, $u \in U$, we add the arc (v, u) with weight $-w(vu)$.
2. For each alternating path $uv'u'$ in G'_c, where $u, u' \in U, v' \in V$, and uv' is matched, we add the arc (u, u') with weight $w(uv') - w(v'u')$.
3. For each matching edge uv', where $u \in U, v' \in V$, we add the arc (u, a) with weight $w(uv')$.
4. We add the arc (v, a) with weight 0.

The time to construct the graph is bounded by $\mathcal{O}(s^2)$. An example is depicted in Figure 4a. Figure 4b depicts the resulting MWM. Since $\overline{G'_c}$ has $\mathcal{O}(s)$ vertices and $\mathcal{O}(s^2)$ edges, we may compute a shortest path P from c's partner v to a in time $\mathcal{O}(s^3)$. Again, P yields the augmenting path in G'_c. Since all the s vertices of U are matched by M, the total time to compute a MWM on each of the graphs G_c, $c \in U$, is $\mathcal{O}(s^4)$. ◀

Sequence of computation. For given vertices $(u, v) \in V(T) \times V(T')$ we denote the matching graph G without removed vertices as *main instance* and the matching graphs G_c as its *sub instances*. Analog for $t \in \{\curlywedge, \diamond\}$ we define $M_t^{T',v}$ as main instance and $M_t^{T',c}$ for each $c \in N(v)$ as its *sub instances*.

Let $u \in V(T)$ and $vw \in E(T')$. We observe, for type $t \in \{\curlywedge, \diamond\}$ the following values depend circularly on each other. To compute $M_t^{T',w}$ recursively for the vertices u, w we need $\mathcal{L}^w(u, v, t)$. Computing $\mathcal{L}^w(u, v, t)$ requires $M_t^{T',v}$ computed recursively for the vertices u, v. The latter one requires $\mathcal{L}^v(u, w, t)$. Finally $\mathcal{L}^v(u, w, t)$ requires $M_t^{T',w}$ computed recursively for the vertices u, w, which was the start of the circular dependency.

We further observe, the MWMs depend on table entries of both types. We may break the dependencies by solving at most one sub instance before solving the main instance, as shown next.

We iterate over all roots $s \in V(T')$ and compute a rooted LaWeCSE between T^r and T'^s as in Lemma 5. During the recursion on vertices (u, v) the following cases may happen.

1. The first instance to compute on (u, v) is a main instance. Then we instantly compute all its sub instances from it.
2. The first instance to compute on (u, v) is a sub instance. Then we compute only the sub instance without deriving it from the main instance.
   - **a.** If the second instance is a main instance, we instantly compute its sub instances.
   - **b.** Otherwise let $c_1 \in N(v)$ and $c_2 \in N(v)$ be the vertices corresponding to the first and second sub instance, respectively. Let us consider table entries first. When

we computed the sub instance corresponding to c_1, all necessary table entries for $M_t^{T',v}$ except $\mathcal{L}^v(u, c_1, t)$ were available. For the second sub instance $\mathcal{L}^v(u, c_1, t)$ is also available. Thus we may instantly compute the main instance and all other sub instances including the one corresponding to c_2. We may argue analog for the MWMs.

▶ **Theorem 15.** *Let T and T' be (unrooted) vertex and/or edge labeled trees. Let ω be a weight function, $\Delta = \min\{\Delta(T), \Delta(T')\}$, and p be a distance penalty. A LaWeCSE$_u$ between T and T' can be computed in time $\mathcal{O}(|T|\,|T'|\Delta)$ and space $\mathcal{O}(|T|\,|T'|)$.*

6 Conclusions

We presented an algorithm which solves the largest weight common subtree embedding problem in time $\mathcal{O}(|T|\,|T'|\Delta)$. For rooted trees of integral weights bounded by a constant we proved a bound of $\mathcal{O}(|T|\,|T'|\sqrt{\Delta}\log(C\min\{|T|, |T'|\}))$. Our approach generalizes the maximum common subtree problem [4] and the largest common subtree embedding problem, both unlabeled [8] and labeled [9], by supporting weights between labels and a distance penalty for skipped vertices.

A remaining open problem is whether the time bound for unrooted trees can be improved when the weights are integral and bounded by a constant. Since weight scaling algorithms for matchings do not work incrementally [13], there is no obvious way to exploit the similarities in the given matching graphs.

References

1. Moon Jung Chung. $O(n^{2.5})$ time algorithms for the subgraph homeomorphism problem on trees. *Journal of Algorithms*, 8(1):106–112, 1987. doi:10.1016/0196-6774(87)90030-7.
2. James Trimble Ciaran McCreesh, Patrick Prosser. A partitioning algorithm for maximum common subgraph problems. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 712–719, 2017. doi:10.24963/ijcai.2017/99.
3. Andre Droschinsky, Nils Kriege, and Petra Mutzel. *Finding Largest Common Substructures of Molecules in Quadratic Time*, pages 309–321. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-51963-0_24.
4. Andre Droschinsky, Nils M. Kriege, and Petra Mutzel. Faster Algorithms for the Maximum Common Subtree Isomorphism Problem. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.MFCS.2016.33.
5. Hans-Christian Ehrlich and Matthias Rarey. Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(1):68–79, 2011. doi:10.1002/wcms.5.
6. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
7. Andrew V. Goldberg, Sagi Hed, Haim Kaplan, and Robert E. Tarjan. Minimum-cost flows in unit-capacity networks. *Theory of Computing Systems*, 61(4):987–1010, Nov 2017. doi:10.1007/s00224-017-9776-7.
8. Arvind Gupta and Naomi Nishimura. Finding largest subtrees and smallest supertrees. *Algorithmica*, 21:183–210, 1998. doi:10.1007/PL00009212.

9 Ming-Yang Kao, Tak-Wah Lam, Wing-Kin Sung, and Hing-Fung Ting. An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. *Journal of Algorithms*, 40(2):212–233, 2001. doi:10.1006/jagm.2001.1163.

10 Antoni Lozano and Gabriel Valiente. On the maximum common embedded subtree problem for ordered trees. In *In C. Iliopoulos and T Lecroq, editors, String Algorithmics, chapter 7. King's College London Publications*, 2004.

11 Daniel M. Martin and Bhalchandra D. Thatte. The maximum agreement subtree problem. *Discrete Applied Mathematics*, 161(13-14):1805–1817, 2013. doi:10.1016/j.dam.2013.02.037.

12 David W. Matula. Subtree isomorphism in $O(n^{5/2})$. In P. Hell B. Alspach and D.J. Miller, editors, *Algorithmic Aspects of Combinatorics*, volume 2 of *Annals of Discrete Mathematics*, pages 91–106. Elsevier, 1978. doi:10.1016/S0167-5060(08)70324-8.

13 L. Ramshaw and R. E. Tarjan. A weight-scaling algorithm for min-cost imperfect matchings in bipartite graphs. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 581–590, Oct 2012. doi:10.1109/FOCS.2012.9.

14 Lyle Ramshaw and Robert Tarjan. On minimum-cost assignments in unbalanced bipartite graphs, 04 2012. URL: http://www.hpl.hp.com/techreports/2012/HPL-2012-40R1.html.

15 Matthias Rarey and J. Scott Dixon. Feature trees: A new molecular similarity measure based on tree matching. *Journal of Computer-Aided Molecular Design*, 12:471–490, 1998. doi:10.1023/A:1008068904628.

16 John W. Raymond and Peter Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16(7):521–533, 2002. URL: http://ipsapp008.lwwonline.com/content/getfile/4830/45/6/abstract.htm.

17 Leander Schietgat, Jan Ramon, and Maurice Bruynooghe. A polynomial-time maximum common subgraph algorithm for outerplanar graphs and its application to chemoinformatics. *Annals of Mathematics and Artificial Intelligence*, 69(4):343–376, 2013. doi:10.1007/s10472-013-9335-0.

18 Gabriel Valiente. *Algorithms on Trees and Graphs*. Springer-Verlag, Berlin, 2002.

19 Atsuko Yamaguchi, Kiyoko F. Aoki, and Hiroshi Mamitsuka. Finding the maximum common subgraph of a partial k-tree and a graph with a polynomially bounded number of spanning trees. *Inf. Process. Lett.*, 92(2):57–63, 2004. doi:10.1016/j.ipl.2004.06.019.

Enumerating Minimal Transversals of Hypergraphs without Small Holes

Mamadou M. Kanté
Université Clermont Auvergne, LIMOS, CNRS
Aubiére, France
mamadou.kante@uca.fr

Kaveh Khoshkhah[1]
Institute of Computer Science, University of Tartu
Tartu, Estonia
khoshkhah@theory.cs.ut.ee

Mozhgan Pourmoradnasseri
Université Clermont Auvergne, LIMOS, CNRS
Aubiére, France
mozhgan.pourmoradnasseri@isima.fr

Abstract

We give a polynomial delay algorithm for enumerating the minimal transversals of hypergraphs without induced cycles of length 3 and 4. As a corollary, we can enumerate, with polynomial delay, the vertices of any polyhedron $\mathcal{P}(A, \underline{1}) = \{x \in \mathbb{R}^n \mid Ax \geq \underline{1},\ x \geq \underline{0}\}$, when A is a balanced matrix that does not contain as a submatrix the incidence matrix of a cycle of length 4. Other consequences are a polynomial delay algorithm for enumerating the minimal dominating sets of graphs of girth at least 9 and an incremental delay algorithm for enumerating all the minimal dominating sets of a bipartite graph without induced 6 and 8-cycles.

2012 ACM Subject Classification Mathematics of computing → Graph enumeration, Mathematics of computing → Graph algorithms, Theory of computation → Graph algorithms analysis

Keywords and phrases Triangle-free Hypergraph, Minimal Transversal, Balanced Matrix, Minimal Dominating Set

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.55

Funding M. M. Kanté and M. Pourmoradnasseri are supported by the French Agency for Research under the GraphEN project ANR-15-CE40-0009.

1 Introduction

The task of an *enumeration algorithm* is generating all the feasible solutions of a given property, such as enumerating all the maximal cliques of a graph or all the triangulations of a given set of points in a d-dimensional space. In enumeration algorithms the size of the output is often exponential in the size of the input, therefore, to define the tractability of enumeration problems, the complexity is measured based on the needed total time of the algorithm depending on the size of the input and the size of the output. If the total running time of the algorithm is bounded by a polynomial on the size of the input and the output, the algorithm is called *output-polynomial*. For a good survey of various combinatorial

[1] Kaveh Khoshkhah was supported by the Estonian Research Council, ETAG (Eesti Teadusagentuur), through PUT Exploratory Grant #620.

enumeration problems and their known time complexities see [32]. It is worth noticing that, unless P=NP, there are enumeration problems where no output polynomial enumeration algorithm exist [22, 23].

In the area of enumeration algorithms, enumerating all the inclusion-wise minimal transversals of a hypergraph[2], known as HYPERGRAPH DUALISATION, is a long-standing open problem which arises in different areas of computer science such as data mining [3], game theory [29, 19], artificial intelligence [12, 31], databases [8, 18], learning theory [1] and integer programming [5, 13]. Despite all the attempts, the complexity of the problem is not settled yet. The best-known algorithm for solving HYPERGRAPH DUALISATION in the general case is by Fredman and Khachiyan [14] (see also [24]) which solves the equivalent problem of monotone Boolean duality in quasi-polynomial time. Nevertheless, for several well-structured hypergraph classes, output-polynomial algorithms for HYPERGRAPH DUALISATION is known, *e.g.*, [11, 25, 22] to cite a few. It was also proved in [20] that the HYPERGRAPH DUALISATION is equivalent to the enumeration of minimal dominating sets in graphs, allowing to tackle this old problem in the realm of graph theory (see for instance [20, 16]). In this paper, we investigate the HYPERGRAPH DUALISATION problem in the class of hypergraphs without induced small cycles. A *k-cycle* in a hypergraph $\mathcal{H}$ is a sequence $(x_0, E_0, x_1, E_1, x_2, \cdots, x_{k-1}, E_{k-1}, x_0)$ where the x_i's belong to $V(\mathcal{H})$, the E_i's are in $\mathcal{H}$, $x_0 \in E_{k-1} \cap E_0$ and, for $1 \leq i \leq k-1$, $x_i \in E_i \cap E_{i-1}$. A *chord* in a k-cycles is a pair (x_i, E_j) where $j \notin \{i, (i-1) \mod k\}$. An enumeration algorithm is said to be of *polynomial delay* if the time between two outputs is bounded a polynomial on the input. Our main theorem is the following.

▶ **Theorem 1.** *Let $\mathcal{H}$ be a hypergraph without chordless 3 and 4-cycles. Then, one can enumerate with polynomial delay the minimal transversals of $\mathcal{H}$.*

The first consequence of our main theorem is a polynomial delay algorithm for listing the minimal dominating sets of graphs with girth at least 9, answering a question from [17]. The girth of a graph in G is the shortest chordless cycle[3] in G. Notice that the recent paper [26], proposes an enumeration algorithm, with a constant time delay, which enumerates all the (not necessarily minimal) dominating sets of a given graph of girth at least 9.

▶ **Corollary 2.**

a. *There is a polynomial delay algorithm that enumerates all the minimal dominating sets of a given input graph G of girth at least 9. More precisely, the result holds if G does not contain induced $(4, 5, 6, 7, 8)$-cycles.*

b. *There is an incremental delay algorithm that enumerates all the minimal dominating sets of a given bipartite graph without chordless 6 and 8-cycles.*

Enumeration algorithms also appear in computational geometry. The famous Minkowski-Weyl theorem states that every convex polyhedron can be represented as the intersection of finitely many affine half spaces, known as *H-representation*, and by the Minkowski sum of a polytope and a finitely generated cone, known as *V-representation*, while the two representations are equivalent. There are various open enumeration problems in this area, such as *facet enumeration* and *convex hull problem*. We refer to [15] for more study. One of the important enumeration problems in computational geometry is *vertex enumeration problem*

[2] A hypergraph $\mathcal{H}$ is a collection of subsets of a ground set $V(\mathcal{H})$ and a *transversal* of $\mathcal{H}$ is a subset T of $V(\mathcal{H})$ that intersects all sets in $\mathcal{H}$.

[3] A k-cycle in a graph G is a sequence $(v_1, v_2, \ldots, v_k)$ where v_i and v_{i+1} are adjacent in G and v_1 is adjacent with v_k. A k-cycle $(v_1, \ldots, v_k)$ is chordless if there are no other edges between the v_i's.

which asks for generating all the vertices of a polyhedron given by its H-representation. Khachiyan et al. [23] proved that enumerating all the vertices of a rational polyhedron, given as an intersection of finitely many half spaces is an NP-hard enumeration problem. It's worth mentioning that the complexity of the problem in the case of polytopes (bounded polyhedrons), yet remains open. The hardness of enumerating the vertices of a polyhedron is more interesting when it comes true even for 0/1 polyhedrons (polyhedrons with vertices in $\{0,1\}^n$) [6] which is in contrast with the fact that all the vertices of a 0/1 polytope are enumerable in *polynomial delay* [9]. Regardless of the hardness of vertex enumeration for general polyhedrons, it is interesting to ask for which classes of polyhedrons the problem is tractable.

As a consequence of our main theorem, we also obtain a polynomial delay algorithm for enumerating the vertices of a large subclass of 0/1 polyhedrons given by *balanced* matrices. For doing so we use the known equivalent characterisation in terms of a HYPERGRAPH DUALISATION problem. Let us define the problem formally. Let $A \in \{0,1\}^{m\times n}$ and $\underline{1}$ and $\underline{0}$ be respectively all ones and all zeros vectors with appropriate size and $\mathcal{P}(A,\underline{1}) = \{x \in \mathbf{R}^n \mid Ax \geq \underline{1},\ x \geq \underline{0}\}$ be a polyhedron with only integral vertices. In other words, $\mathcal{P}(A,\underline{1})$ is the *set covering* polyhedron with the 0/1 *ideal* matrix A. It is well-known and not hard to see that the vertices of $\mathcal{P}(A,\underline{1})$ are in bijection with the minimal transversals of the hypergraph $\mathcal{H}[A]$, where the columns of A correspond to vertices of $\mathcal{H}[A]$ and the rows of A are incident vectors of the hyperedges of $\mathcal{H}[A]$ [27]. This gives an equivalence between the vertex enumeration problem for $\mathcal{P}(A,\underline{1})$ and the HYPERGRAPH DUALISATION problem for such hypergraphs $\mathcal{H}[A]$. The existence of the quasi-polynomial algorithm for enumerating the minimal transversals of a hypergraph [14] suggests that the complexity of vertex enumeration for $\mathcal{P}(A,\underline{1})$ is unlikely to be in NP.

In the Recent paper [13], Elbassioni and Makino have given an incremental polynomial time algorithm for enumerating the vertices of $\mathcal{P}(A,\underline{1})$ when A is a 0/1 totally unimodular matrix. Totally unimodular matrices are an important class of matrices for integer programming with the property that every square submatrix of it has determinant 0 or 1. The generating method in [13] is based on enumerating the transversals of the associated hypergraph and Seymour's fundamental decomposition theorem for totally unimodular matrices [30]. As an interesting open problem, one may ask about the existence of an output polynomial algorithm for enumerating the vertices of $\mathcal{P}(A,\underline{1})$ when A is a *balanced* matrix [13]. A 0/1 matrix is *balanced* if it does not contain a submatrix that is the incidence matrix of a cycle of odd length (see [30, Chapter 21] or [10]). Totally unimodular matrices are a proper subset of balanced matrices. A consequence of our main theorem is the following.

▶ **Theorem 3.** *There is a polynomial delay algorithm for listing the vertices of any given* 0/1 *polyhedron* $\mathcal{P}(A,\underline{1})$ *whenever* A *is a balanced matrix without any submatrix that is the incident matrix of a* 4*-cycle.*

As the algorithm needs some technical definitions, we postpone the details of the algorithm to Section 2. The main technical part of the paper is in Section 3 where we prove the main theorem.

2 Definitions and Preliminaries

The power set of a set V is denoted by 2^V, and for two sets A and B, we let $A \setminus B$ denote the set $\{x \in A \mid x \notin B\}$.

A hypergraph $\mathcal{H}$ is a collection of subsets of a finite ground set. The elements of $\mathcal{H}$ are called the *hyperedges* of $\mathcal{H}$ and the *vertex set* of $\mathcal{H}$ is $V(\mathcal{H}) := \bigcup_{E\in\mathcal{H}} E$. Given $S \subseteq V(\mathcal{H})$,

we denote by $\mathcal{H}[S]$ the *hypergraph induced by* S, that is, $\mathcal{H}[S] := \{E \cap S \mid E \in \mathcal{H}\}$. Any subset $\mathcal{H}'$ of $\mathcal{H}$ is called a *sub-hypergraph* of $\mathcal{H}$. Notice that if there exists $E \in \mathcal{H}$ such that $E \subseteq V(\mathcal{H}) \setminus S$, then $\emptyset \in \mathcal{H}[S]$.

Given a hypergraph $\mathcal{H}$ and a subset $S \subseteq V(\mathcal{H})$ of its vertex set, we denote by $\mathcal{H}(S)$ the sub-hypergraph $\{E \in \mathcal{H} \mid S \cap E \neq \emptyset\}$; and for $v \in V(\mathcal{H})$, we write $\mathcal{H}(v)$ instead of $\mathcal{H}(\{v\})$. Notice that $\mathcal{H}(v)$ is the set of hyperedges containing v.

We assume that each hypergraph is given with an ordering $\leq$ of its set of vertices.

A *k-hole* in a hypergraph is a chordless cycle of length k.

A *transversal* (or hitting set) of a hypergraph $\mathcal{H}$ is a set $T \subseteq V(\mathcal{H})$ that has a nonempty intersection with every hyperedge $E \in \mathcal{H}$. A transversal T is said *minimal* if no proper subset of T is a transversal. For $T \subseteq V(\mathcal{H})$ and $x \in T$, we let $\mathcal{P}_T(x) := \{E \in \mathcal{H} \mid E \cap T = \{x\}\}$, and call it the set of *privates* of x with respect to T (we may drop the "with respect to T" when T is clear from the context). We say that $B \subseteq V(\mathcal{H})$ *breaks* $x \in T$, if for every $E \in \mathcal{P}_T(x)$, $E \cap B \neq \emptyset$ and if $B = \{b\}$, we say that b breaks x, for short.

We call $T \subseteq V(\mathcal{H})$ *irredundant* if $\mathcal{P}_T(x) \neq \emptyset$ for all $x \in T$. It is well-known that T is a minimal transversal if and only if T is a transversal and is irredundant. We denote by $\mathsf{tr}(\mathcal{H})$ the set of minimal transversals of $\mathcal{H}$. Observe that if all hyperedges of $\mathcal{H}$ are non-empty, then $\mathsf{tr}(\mathcal{H}) \neq \emptyset$. For more definitions and details on hypergraphs, we refer to [4].

▶ **Definition 4.** For $\ell \in V(\mathcal{H})$ and $E \in \mathcal{H}$ such that $\ell \in E$, we let $S(\ell, E)$, called a *double star*, be the sub-hypergraph $\mathcal{H}(\ell)[E]$. A double star is called *valid* if $(\mathcal{H} \setminus \mathcal{H}(\ell))[V(\mathcal{H}) \setminus E]$ does not contain the empty set.

The notion of double star is defined and used in [10] for the decomposition of balanced matrices. We rephrase it in terms of hypergraphs. Observe also that if a hypergraph is *Sperner* (no hyperedge contains another hyperedge), then every double star is valid. Even though for the Hypergraph Dualisation problem, it is enough to consider Sperner hypergraphs, we prefer giving the definition above for general hypergraphs for a better readability of our algorithms as we manipulate induced sub-hypergraphs. We often use the notation S for the double star $S(\ell, E)$ in customary whenever ℓ and E are clear from the context.

▶ **Fact 5.** *If $\emptyset \notin \mathcal{H}$, then $\mathcal{H}$ has a valid double star and it can be found in polynomial time*

Proof. Let $T \in \mathsf{tr}(\mathcal{H})$, which exists because $\emptyset \notin \mathcal{H}$. Let $\ell \in T$ and $E \in \mathcal{P}_T(\ell)$. Since T is a transversal and $E \cap T = \{\ell\}$, then each hyperedge in $\mathcal{H} \setminus \mathcal{H}(\ell)$ has a nonempty intersection with $T \setminus \{\ell\}$. Now, since $T \setminus \{\ell\} \subseteq V(\mathcal{H}) \setminus E$ because $E \cap T = \{\ell\}$, we can conclude that $(\mathcal{H} \setminus \mathcal{H}(\ell))[V(\mathcal{H}) \setminus E]$ does not contain the $\emptyset$ as a hyperedge. ◀

The algorithm for enumerating minimal transversals uses the standard technique which consists, for a hypergraph $\mathcal{H}$, in choosing a vertex ℓ and enumerate the minimal transversals that do not contain ℓ, denoted by $\mathbf{Inc}(\mathcal{H}, \ell)$, and those that do contain ℓ, denoted by $\mathbf{Exc}(\mathcal{H}, \ell)$. For enumerating the minimal transversals that do not contain ℓ, it suffices to make a recursive call to $\mathcal{H}[V(\mathcal{H}) \setminus \{\ell\}]$, once we ensure that one exists (which can be checked in polynomial time). But, enumerating the minimal transversals containing ℓ is a tough task and is exactly what makes the enumeration of $\mathsf{tr}(\mathcal{H})$ difficult because such a strategy causes to ask at each step the following NP-complete problem [7]: Given $X \subseteq V(\mathcal{H})$, does there exist a minimal transversal including X? In order to avoid this NP-complete problem, we use the following strategy, which depends heavily on the fact that 3-holes and 4-holes are forbidden:

1. We always choose ℓ to be in a valid double star $S(\ell, E)$.
2. We secondly show that, for any minimal transversal T of $\mathcal{H} \setminus S(\ell, E)$, $T \cup \{\ell\}$ is a minimal transversal of $\mathcal{H}$. Such minimal transversals are called *basic*.
3. We then show that any minimal transversal containing ℓ is either basic or can be obtained from a basic one by successively applying a *flipping method*. The flipping method yields a parent-child relation between the minimal transversals containing ℓ.
4. We finally use this parent-child relation to enumerate (with polynomial delay) the minimal transversals containing ℓ.

Let $\mathcal{H}$ be a hypergraph with $n := |V(\mathcal{H})| + \sum_{F \in \mathcal{H}} |F|$. An *enumeration algorithm* for $\mathsf{tr}(\mathcal{H})$ is an algorithm that lists all the minimal transversals without repetitions. An enumeration algorithm $\mathbb{A}$ for $\mathsf{tr}(\mathcal{H})$ which terminates in time $p(n, |\mathsf{tr}(\mathcal{H})|)$ for some polynomial $p(x, y)$ is called *output-polynomial* and it is called *polynomial space* if it uses a space bounded by a polynomial in n. Assume now that $T_1, \ldots, T_m$ are the elements of $\mathsf{tr}(\mathcal{H})$ enumerated in the order in which they are generated by $\mathbb{A}$. Let us denote by $T(\mathbb{A}, i)$ the time $\mathbb{A}$ requires until it outputs T_i, also $T(\mathbb{A}, m+1)$ is the time required by $\mathbb{A}$ until it stops. Let $delay(\mathbb{A}, 1) = T(\mathbb{A}, 1)$ and $delay(\mathbb{A}, i) = T(\mathbb{A}, i) - T(\mathbb{A}, i-1)$. The *delay* of $\mathbb{A}$ is $\max\{delay(\mathbb{A}, i)\}$. Algorithm $\mathbb{A}$ is a *polynomial delay* algorithm if there is a polynomial $p(x)$ such that the delay of $\mathbb{A}$ is at most $p(n)$.

The remainder of this paper is as follows. In Section 3 we define the flipping method, the basic minimal transversals and the resulting parent-child relation. We also prove that if $S(\ell, E)$ is a valid double star, then any minimal transversal containing ℓ can be obtained from a basic minimal transversal by following the parent-child relation. The algorithm for enumerating the children of a minimal transversal containing ℓ is given in Section 3.2.

3 Enumeration of minimal transversals including ℓ from a valid double star $S(\ell, E)$

3.1 Basic transversals, flipping operation and parent-child relation

In this section, we introduce the family of minimal transversals $\mathcal{B}$, called *basic transversals*, which will be used as a *base* for generating all the minimal transversals T containing the vertex ℓ. In the first step, a valid double star $S(\ell, E)$ is fixed for $\mathcal{H}$.

▶ **Fact 6.** *For every minimal transversal T of $(\mathcal{H} \setminus \mathcal{H}(\ell))[V(\mathcal{H}) \setminus E]$, $T \cup \{\ell\}$ is a minimal transversal of $\mathcal{H}$.*

Proof. Since T is a minimal transversal of $(\mathcal{H} \setminus \mathcal{H}(\ell))[V(\mathcal{H}) \setminus E]$, $T \subseteq V(\mathcal{H}) \setminus E$ and each vertex of T has a private in $\mathcal{H} \setminus \mathcal{H}(\ell)$. As $\{\ell\}$ is a minimal transversal of $\mathcal{H}(\ell)$ and $E \cap T = \emptyset$, we can conclude that $T \cup \{\ell\}$ is a minimal transversal of $\mathcal{H}$. ◀

We denote by $\mathcal{B}(\ell, E)$ the set $\{T \cup \{\ell\} \mid T \in \mathsf{tr}((\mathcal{H} \setminus \mathcal{H}(\ell))[V(\mathcal{H}) \setminus E])\}$ and call it the set of *basic transversals* of $\mathcal{H}$. We will show that one can generate all the minimal transversals of $\mathcal{H}$ that contain ℓ by doing flipping operations, starting from $\mathcal{B}(\ell, E)$.

Recall that $S(\ell, E)$ is a fixed double star of a fixed hypergraph $\mathcal{H}$. The objective is to define a way to generate the set of all minimal transversals containing the vertex ℓ, starting with the basic transversals. We first define a parent-child relation based on a flipping operation which results in removing one vertex, from E, at a time. As a consequence, each minimal transversal containing ℓ will be reachable, by following the parent-child relation,

Algorithm 1: GreedyPair.

Input: $T \subseteq V$ and the largest succedent vertex x in T
Output: $(Y, (Z_y)_{y \in Y})$

```
Function GreedyPair(H, T, x)
    Y := ∅;
    while P_T(x) is not empty do
        Choose the smallest y ∈ V(P_T(x)) \ E such that ∃ F ∈ P_T(x),
          F \ E ⊆ {v ∈ V(H) | v ≤ y};
        Y := Y ∪ {y};
        Z_y := {z ∈ T | P_T(z) ⊆ H(y)};
        T := (T ∪ {y}) \ Z_y;
```

from a basic transversal. In a second step, we explain how to generate the children of any minimal transversal containing ℓ. Let's denote by $\mathbf{Inc}(\mathcal{H}, \ell, E)$ the set of minimal transversals of $\mathcal{H}$ containing ℓ where $S(\ell, E)$ is a valid double star of $\mathcal{H}$.

▶ **Definition 7.** Let T be an irredundant set of $\mathcal{H}$. A vertex x in T is called a *succedent vertex* if $x \in E \setminus \{\ell\}$.

Observe that a minimal transversal is basic if and only if it does not contain any succedent vertex. Also, if x is a succedent vertex of $T \in \mathbf{Inc}(\mathcal{H}, \ell, E)$, then $\mathcal{P}_T(x) \subseteq \mathcal{H} \setminus \mathcal{H}(\ell)$ because $\ell \in T$.

▶ **Definition 8.** Let T be an irredundant set containing succedent vertices and let $x \in T$ be the largest succedent vertex of T with respect to the ordering $\leq$. We call $(Y, (Z_y)_{y \in Y})$ a *greedy pair of* $\mathcal{P}_T(x)$ if

1. $Y \subseteq V(\mathcal{P}_T(x)) \setminus E$ and $Z_y \subseteq T$ for each $y \in Y$,
2. Y is a minimal transversal of $\mathcal{P}_T(x)$,
3. for each $y \in Y$, there is a hyperedge $F \in \mathcal{P}_{T_y}(x)$ such that $y \in F$ and $F \setminus E \subseteq \{v \in V(\mathcal{H}) \mid v \leq y\}$, where $T_y := (T \cup \{y' \in Y \mid y' < y\}) \setminus (\cup_{y' < y} Z_{y'})$ for each $y \in Y$,
4. for each $y \in Y$, $Z_y := \{z \in T \mid \mathcal{P}_{T_y}(z) \subseteq \mathcal{H}(y)\}$ with T_y as defined above.

▶ **Fact 9.** *The greedy pair of $\mathcal{P}_T(x)$ is unique and is computed in polynomial time by the function* GreedyPair *depicted in Algorithm 1.*

Proof. It is easy to see that the function GreedyPair in algorithm 1 runs in polynomial time and its output, $(Y, (Z_y)_{y \in Y})$, is a greedy pair. Assume that there is another greedy pair $(Y', (Z'_y)_{y \in Y'})$. It is enough to show that $Y = Y'$ since $(Z_y)_{y \in Y}$ and $(Z'_y)_{y \in Y'}$ are determined completely by Y and Y', respectively. Let us enumerate Y and Y' as $y_1 < y_2 < \cdots < y_k$ and $y'_{i_1} < y'_{i_p} < \cdots < y'_{i_p}$, respectively. Let j the smallest such that $y'_{i_j} \neq y_j$. If y'_{i_j} does not exist, then Y' cannot be a transversal of $\mathcal{P}_T(x)$ and similarly if y_j does not exist. So, let us assume that such a j exists. If $y'_{i_j} < y_j$, then there is an edge $F_{i_j} \in \mathcal{P}_T(x)$ which does not intersect Y because F_{i_j} does not intersect $\{y_1, \ldots, y_{j-1}\}$ and $F_{i_j} \setminus E \subseteq \{w \in V(\mathcal{H}) \mid w < y_{i_j}\}$ (Condition (3) of Definition 8). Similarly, if $y_j < y_{i_j}$, there is an edge $F_j \in \mathcal{P}_T(x)$ which does not intersect Y' as F_j does not intersect $\{y_1, \ldots, y_{j-1}\}$ and $F_j \setminus E \subseteq \{w \in V(\mathcal{H}) \mid w < y_j\}$. In both cases, we contradict the fact that Y or Y' is a transversal of $\mathcal{P}_T(x)$. ◀

If $(Y, (Z_y)_{y \in Y})$ is the greedy pair of $\mathcal{P}_T(x)$, for a minimal transversal T, then Y is intended to replace x in T, but even though $(T \setminus \{x\}) \cup Y$ is a transversal, it is not necessarily minimal.

The set $\cup_{y\in Y} Z_y$ is the set to remove to obtain a minimal transversal. The next lemma allows to prove that $(T \cup Y) \setminus ((\cup_{y\in Y} Z_y) \cup \{x\})$ is a minimal transversal.

▶ **Lemma 10.** *Let T be an irredundant set of $\mathcal{H}$ containing succedent vertices and let x be the largest succedent vertex of T. Let $(Y, (Z_y)_{y\in Y})$ be the greedy pair of $\mathcal{P}_T(x)$ and let $Z := \cup_{y\in Y} Z_y$. Then, the following properties hold:*

a. *For each vertex x' of $T \cap E$ different from x, and each $F' \in \mathcal{P}_T(x')$, we have $F' \cap Y = \emptyset$.*
b. *For each hyperedge $F \in \mathcal{H} \setminus \mathcal{H}(x)$, we have $|F \cap Y| \leq 1$.*
c. *For each $z \in Z$, there is exactly one $y \in Y$ such that $\mathcal{H}(y) \cap \mathcal{P}_T(z) \neq \emptyset$.*
d. *If there are $z_i, z_j \in Z$ and $y \in Y$ such that $\mathcal{H}(y) \cap \mathcal{P}_T(z_i) \neq \emptyset$ and $\mathcal{H}(y) \cap \mathcal{P}_T(z_j) \neq \emptyset$, then there is no hyperedge in $\mathcal{H} \setminus \mathcal{H}(y)$ which includes both z_i and z_j.*
e. *If there are $z \in Z$ and $y \in Y$ such that $\mathcal{H}(y) \cap \mathcal{P}_T(z) \neq \emptyset$, then $\mathcal{H}(x) \cap \mathcal{H}(z) \subseteq \mathcal{H}(x) \cap \mathcal{H}(y)$.*

Proof. All the proofs are by contradicting the fact that $\mathcal{H}$ is $(3, 4)$-hole free.

a. Let $x' \neq x$ be a vertex of $T \cap E$ and $F' \in \mathcal{P}_T(x')$ and $y \in Y \cap F'$. Also, assume that $F \in \mathcal{P}_T(x)$, containing y. By the definition of the double star $S(\ell, E)$, the hyperedge E contains all the succedent vertices and therefore, (y, F, x, E, x', F', y) is a 3-hole in $\mathcal{H}$.
b. Let $F \in \mathcal{H} \setminus \mathcal{H}(x)$ be a hyperedge containing two vertices $y_i \leq y_j$, both from Y. Let $F_i \in \mathcal{P}_Y(y_i) \cap \mathcal{H}(x)$ and $F_j \in \mathcal{P}_Y(y_j) \cap \mathcal{H}(x)$, which exist by the definition of a greedy pair. Then $(y_i, F, y_j, F_j, x, F_i, y_i)$ constitutes a 3-hole in $\mathcal{H}$.
c. Let $N_i \in \mathcal{H}(y_i) \cap \mathcal{P}_T(z)$ and $N_j \in \mathcal{H}(y_j) \cap \mathcal{P}_T(z)$ for two distinct vertices y_i and y_j of Y. Notice that $N_i \neq N_j$ by (b). Let $F_i \in \mathcal{P}_T(x) \cap \mathcal{P}_Y(y_i)$ and $F_j \in \mathcal{P}_T(x) \cap \mathcal{P}_Y(y_j)$. Then $(z, N_i, y_i, F_i, x, F_j, y_j, N_j, z)$ gives a 4-hole in $\mathcal{H}$.
d. Let $z_i, z_j \in Z$ such that there is a vertex $y \in Y$, and $F_i \in \mathcal{H}(y) \cap \mathcal{P}_T(z_i)$ and $F_j \in \mathcal{H}(y) \cap \mathcal{P}_T(z_j)$. Suppose that there is a hyperedge $F \in \mathcal{H} \setminus \mathcal{H}(y)$ that contains both z_i and z_j. Then, $(z_j, F, z_i, F_i, y, F_j, z_j)$ induces a 3-hole in $\mathcal{H}$.
e. Let $F_z \in \mathcal{H}(y) \cap \mathcal{P}_T(z)$ and $N_z \in \mathcal{H}(x) \cap \mathcal{H}(z)$ such that $y \notin N_z$. Let $N_y \in \mathcal{H}(x) \cap \mathcal{H}(y)$. Then, $(x, N_z, z, F_z, y, N_y, x)$ is a 3-hole in $\mathcal{H}$. ◀

From Lemma 10, we can deduce the following.

▶ **Proposition 11.** *Let $T \in \mathbf{Inc}(\mathcal{H}, \ell, E)$ be a non-basic minimal transversal and let x be the largest succedent vertex of T. Let $(Y, (Z_y)_{y\in Y})$ be the greedy pair of $\mathcal{P}_T(x)$. Then, $T^* := (T \cup Y) \setminus ((\cup_{y\in Y} Z_y) \cup \{x\})$ belongs to $\mathbf{Inc}(\mathcal{H}, \ell, E)$. Moreover, T^* has one less succedent vertices than T.*

Proof. Let $Z := \cup_{y\in Y} Z_y$. By definition of T^*, it is clear that $\ell \in T^*$ and it has less succedent vertices than T since x is removed from T. By the definition of the greedy pair, each vertex $y \in Y$ has a private with respect to T^*, and by Lemma 10(a), each vertex of $(T \cap E) \setminus \{x\}$ has a private with respect to T^*. Also, by the definition of the greedy pair, each vertex z of $T \setminus (E \cup Z)$ has a private with respect to T^*. It remains to show that T^* is a transversal. If there is $F \in \mathcal{H}$ such that $F \cap T^* = \emptyset$, then by the definition of Z and by Lemma 10(e), F is not the private of any vertex $z \in Z$ and then, there are at least two distinct vertices z and z' both contained in $F \cap Z$. By Lemma 10(d), $z \in Z_{y_i}$ and $z' \in Z_{y_j}$ for two distinct y_i and y_j in Y. Let y_j be the largest such that there is $z' \in Z_{y_j}$ and $z' \in F$. Then, F necessarily belongs to the private of z' with respect to $(T \cup \{y' \in Y \mid y' < y_j\}) \setminus (\cup_{y'<y_j} Z_{y'})$, which contradicts the fact that $z' \in Z_{y_j}$. This concludes the proof. ◀

We are now ready to define the parent-child relation.

Algorithm 2: Enum$(\mathcal{H}, \leq)$.

Input: A $(3,4)$-hole free hypergraph $\mathcal{H}$ and a linear ordering $\leq$ of $V(\mathcal{H})$.

```
begin
    Let S(ℓ, E) be a valid double star of H
    foreach T ∈ Enum ((H \ H(ℓ))[V(H) \ E], ≤ do
        output (T ∪ {ℓ})
        Enum-Children (T ∪ {ℓ})
    Enum (H[V(H) \ {ℓ}], ≤)
```

▶ **Definition 12.** Let T be a non-basic minimal transversal T in $\mathbf{Inc}(\mathcal{H}, \ell, E)$. Let x be the largest succedent vertex of T and let $(Y, (Z_y)_{y\in Y})$ be the greedy pair of $\mathcal{P}_T(x)$. We call $T^* := (T \cup Y) \setminus ((\cup_{y\in Y} Z_y) \cup \{x\})$ the *parent* of T, and call T the *child* of T^* with respect to $(x, Y, (Z_y)_{y\in Y})$.

The following will be used to characterise the children of any minimal transversal in $\mathbf{Inc}(\mathcal{H}, \ell, E)$.

▶ **Fact 13.** *If T is a child of T^* with respect to $(x, Y, (Z_y)_{y\in Y})$ then for every $y \in Y$, $\mathcal{P}_T(x) \cap \mathcal{P}_{T^*}(y) \neq \emptyset$ and for all $z \in Z_{\min(Y)}$, $\mathcal{P}_T(z) \subseteq \mathcal{P}_{T^*}(\min(Y))$.*

▶ **Lemma 14.** *Let T be a child of T^* with respect to $(x, Y, (Z_y)_{y\in Y})$ and let $y_1 := \min(Y \setminus Y_0)$ where $Y_0 := \{y \in Y \mid \mathcal{P}_{T^*}(y) \subseteq \mathcal{H}(x)\}$. Let $Z' := \cup_{y\in Y\setminus\{y_1\}} Z_y$ and $C_{y_1} := V(\mathcal{H}) \setminus (T^* \cup E \cup \{w \in V(\mathcal{H}) \mid \exists t \in T \setminus Z_{y_1} \text{ s.t. } \mathcal{P}_{T\setminus Z_{y_1}}(t) \subseteq \mathcal{H}(w)\})$. Then, $Z_{y_1} \cup \{x\}$ is a minimal transversal of $\mathcal{P}_{(T^*\cup Z')\setminus(Y\setminus\{y_1\})}(y_1)[C_{y_1} \cup \{x\}]$.*

Proof. Notice first that by Fact 13, for each $z \in Z_{y_1}$, $\mathcal{P}_T(z) \subseteq \mathcal{P}_{T^*}(y_1)$ and since $T \setminus Z_{y_1} = (T^* \setminus Y) \cup (Z' \cup \{x\})$ and T is a minimal transversal, we can conclude that $Z_{y_1} \cup \{x\}$ is an irredundant set of $\mathcal{P}_{(T^*\cup Z')\setminus(Y\setminus\{y_1\})}(y_1)$ by Lemma 10(c). Now, because T is a minimal transversal of $\mathcal{H}$, no $z \in Z_{y_1}$ breaks the private of some $t \in T \setminus Z_{y_1}$. So, $Z_{y_1} \subseteq C_{y_1}$. Assume that there is $F \in \mathcal{P}_{(T^*\cup Z')\setminus(Y\setminus\{y_1\})}(y_1)$ such that $F \cap (Z_{y_1} \cup \{x\}) = \emptyset$. Because $T = (T^* \cup Z' \cup Z_{y_1} \cup \{x\}) \setminus Y$, we would have $T \cap F = \emptyset$, contradicting the fact that T is a minimal transversal. ◀

The proofs of the following are trivial from the definitions.

▶ **Lemma 15.** *For every non-basic minimal transversal T in $\mathbf{Inc}(\mathcal{H}, \ell, E)$, the parent of T can be computed in polynomial time.*

▶ **Proposition 16.** *The directed graph with vertex set $\mathbf{Inc}(\mathcal{H}, \ell, E)$ and arc set the pairs (T^*, T) such that T^* is the parent of T is acyclic.*

The algorithm consists now in doing a DFS traversal of the directed graph of Proposition 16. The description is given in Algorithm 2. In order to prove that it runs with polynomial delay, it remains to show that for each non-basic minimal transversal, its children can be enumerated with polynomial delay.

We will now prove in the next section that the children of any $T \in \mathbf{Inc}(\mathcal{H}, \ell, E)$ can be enumerated with polynomial delay and polynomial space.

3.2 Enumerating the children of $T \in \mathbf{Inc}(\mathcal{H}, \ell, E)$

Remember that $\mathcal{H}$ is given with an ordering $\leq$ of $V(\mathcal{H})$. Also, $S(\ell, E)$ is a valid double star of $\mathcal{H}$. For a minimal transversal $T^* \in \mathbf{Inc}(\mathcal{H}, \ell, E)$, let $\mathbf{Cov}(T^*) := \{x \in E \setminus T^* \mid \text{for each } x' \in T^* \cap E,\ x' \leq x \text{ and } \mathcal{P}_{T^* \cup \{x\}}(x') \neq \emptyset\}$. The set $\mathbf{Cov}(T^*)$ is the set of vertices in E that can be added to T^* without breaking the privates of any $x' \in E \cap T^*$ and are therefore candidates for generating the children of T^*.

Let $x \in \mathbf{Cov}(T^*)$ and let $Y_0 := \{y \in T^* \setminus E \mid \mathcal{P}_{T^*}(y) \subseteq \mathcal{H}(x)\}$.

▶ **Lemma 17.** *If $Y_0 \neq \emptyset$, then $T_0 := (T^* \setminus Y_0) \cup \{x\}$ is a minimal transversal of $\mathcal{H}$.*

Proof. By the definition of Y_0 and of T_0, we can conclude that T_0 is an irredundant set. If there is $N \in \mathcal{H}$ not intersected by T_0, then $N \in \mathcal{H}(y) \cap \mathcal{H}(y')$ for two distinct vertices in Y_0. Let $F \in \mathcal{P}_{T^*}(y)$ and $F' \in \mathcal{P}_{T^*}(y')$. Then, (x, F, y, N, y', F', x) is a 3-hole in $\mathcal{H}$, a contradiction. ◀

▶ **Lemma 18.** *If T is a child of T^* with respect to $(x, Y, (Z_y)_{y \in Y})$, then $Y_0 \subseteq Y$.*

Proof. Let T be a child of T^* with respect to $(x, Y, (Z_y)_{y \in Y})$. We first claim that $Y_0 \cap T = \emptyset$. Suppose that there is $y_0 \in T \cap Y_0$. Since T is a child of T^*, then $T^* = (T \cup Y) \setminus (Z \cup \{x\})$ for some greedy pair $(Y, (Z_y)_{y \in Y}$ with $Z := \cup_{y \in Y} Z_y$. Therefore, $y_0 \notin Y$ and thus $y_0 \in T \setminus (Z \cup \{x\})$. Now, because $\mathcal{P}_{T^*}(y_0) \subseteq \mathcal{H}(x)$, we can conclude that there is $y \in Y$ and $F \in \mathcal{H}(y) \cap \mathcal{H}(y_0)$ with $F \in \mathcal{P}_T(y_0)$. Because $y \in Y$, then there is $F' \in \mathcal{P}_T(x)$ such that $F' \in \mathcal{P}_{T^*}(y_1)$. Therefore, $(x, F_0, y_0, F, y_1, F', x)$ is a 3-hole, contradicting that $\mathcal{H}$ is $(3,4)$-hole free, for some $F_0 \in \mathcal{P}_{T^*}(y_0)$.

If T is a child of T^*, then by the previous claim, we know that $Y_0 \cap T = \emptyset$. Because $T^* = (T \cup Y) \setminus (\{x\} \cup Z)$, we can conclude that Y_0 is necessarily a subset of Y, otherwise it would not be a subset of T^*. ◀

Level-0-child. If $Y_0 \neq \emptyset$ and $T_0 := (T^* \setminus Y_0) \cup \{x\}$ is a child of T^* with respect to $(x, Y_0, \emptyset)$, we call T_0 the *level*-0 child of T^*.

Note. If $Y_0 = \emptyset$, we "symbolically" call $T^* \cup \{x\}$ the *level*-0 child of T^* with respect to $(x, \emptyset, \emptyset)$. Also, we say that $(\emptyset, \emptyset)$ is the greedy pair of $\mathcal{P}_{T^* \cup \{x\}}(x)$. Notice that $T^* \cup \{x\}$ is not a minimal transversal.

We will now characterise the other children of T^*. Before, let us first prove the following which is the base of the characterisation.

▶ **Lemma 19.** *Let T be an irredundant set containing succedent vertices and let x be the largest succedent vertex of T. Let $y_1 \in Y$ be the smallest such that $Z_{y_1} \neq \emptyset$. Then, $(Y, (Z_y)_{y \in Y})$ is the greedy pair of $\mathcal{P}_T(x)$ if and only if $(Y, (Z'_y)_{y \in Y})$ is the greedy pair of $\mathcal{P}_{T \setminus Z_{y_1}}(x)$ where*

$$Z'_y := \begin{cases} Z_y & \text{if } y \neq y_1, \\ \emptyset & \text{otherwise.} \end{cases}$$

Proof. Let $(Y', (W_y)_{y \in Y'})$ be the greedy pair of $\mathcal{P}_{T \setminus Z_{\min(Y)}}(x)$. Because $(W_y)_{y \in Y'}$ is determined by Y', it is enough to prove that $Y = Y'$. Let us enumerate Y as $y_{t_1} < y_{t_2} < \cdots < y_{t_p} < y_1 < y_2 < \cdots < y_k$ with $Z_{y_{t_j}} = \emptyset$ for all $1 \leq j \leq p$. Then, Y' is the sequence $y_{t_1} < y_{t_2} < \cdots < y_{t_p} < y'_{i_1} < y'_{i_2} < \cdots < y'_{i_t}$. Let j be the smallest such that $y'_{i_j} \neq y_j$. Let $F_{i_j} \in \mathcal{P}_{T \setminus Z_{y_1}}(x)$ such that $F_{i_j} \setminus E \subseteq \{v \in V(\mathcal{H}) \mid v \leq y'_{i_j}\}$ and let $F_j \in \mathcal{P}_T(x)$ such that $F_j \setminus E \subseteq \{v \in V(\mathcal{H}) \mid v \leq y_j\}$. Observe that because $\mathcal{P}_T(x) \subseteq \mathcal{P}_{T \setminus Z_{y_1}}(x)$, then

$F_j \in \mathcal{P}_{T\setminus Z_{y_1}}(x)$. If $y_j < y'_{i_j}$, then y_j should necessarily belong to Y', contradicting the choice of j. So $y'_{i_j} < y_j$. Assuming that $F_{i_j} \in \mathcal{P}_T(x) \subseteq \mathcal{P}_{T\setminus Z_{y_1}}(x)$ contradicts the fact that $y'_{i_j} \notin Y$. So, $F_{i_j} \notin \mathcal{P}_T(x)$, *i.e.*, there is $z_1 \in Z_{y_1}$ such that $z_1 \in F_{i_j}$. Let $F_1 \in \mathcal{P}_T(z_1)$. Because $z_1 \in Z_{y_1}$, we have that $F_1 \in \mathcal{H}(y_1)$. Then, $(z_1, F_1, y_1, F_j, x, F_{i_j}, z_1)$ is a 3-hole in $\mathcal{H}$. ◀

Level-i children. Let $R := \{y \in T^* \setminus (E \cup Y_0) \mid \mathcal{P}_{T^*}(y) \cap \mathcal{H}(x) \neq \emptyset\}$. The high-level idea for generating the children of T^* with respect to x consists in, recursively, deleting a *correct* set of vertices $Y_0 \subseteq Y \subseteq R$ from T^* and assigning the privates of vertices in Y to x. The choice of Y is determined by checking whether there is a $Z \in \mathsf{tr}(\cup_{y\in Y}\mathcal{P}_{T^*}(y) \setminus \mathcal{H}(x))$ containing x such that $T := (T^* \setminus Y) \cup Z$ is a minimal transversal and child of T^*, which can be checked. As we will see, this step is independent of the choice of Z. In a second step, we will enumerate the set of suitable minimal transversals Z of $\cup_{y\in Y}\mathcal{P}_{T^*}(y) \setminus \mathcal{H}(x)$.

A collection $\mathcal{I}$ of subsets of a ground set V is an *accessible system* if for each $I \in \mathcal{I}$, there is an $i \in I$ such that $I \setminus \{i\} \in \mathcal{I}$. The two following lemmas show that the set of children of T^* is like an accessible system following the Y-parts of the greedy pairs.

▶ **Lemma 20.** *Suppose that $Y_0 = \emptyset$. Let T be a child of T^* with respect to $(x, Y, (Z_y)_{y\in Y})$, with $Y \subseteq R$. Then, $T' := (T \cup \{\min(Y)\}) \setminus Z_{\min(Y)}$ is a child of T^* with respect to $(x, Y \setminus \{\min(Y)\}, (Z_y)_{y\in Y\setminus\{\min(Y)\}})$.*

Proof. If $|Y| = 1$, then $T' = T^* \cup \{x\}$. That is a symbolic *level-0* child of T^* with respect to $(x, \emptyset, \emptyset)$. If $|Y| \geq 2$, let $y_1 := \min(Y)$. By Fact 13, $\mathcal{P}_T(z) \subseteq \mathcal{P}_{T^*}(y_1)$. Then using Lemma 10(d) and by definition, T' is a transversal of $\mathcal{H}$. In order to prove that it is minimal, we must show T' is irredundant. By the construction of Z_1, if y_1 breaks $t \in T$ then $t \in Z_1$. Also, since $|Y| \geq 2$, by Fact 13, $\mathcal{P}_T(x) \cap \mathcal{P}_{T^*}(y) \neq \emptyset$, when $y \in Y \setminus \{y_1\}$ and by the minimality of T^*, $\mathcal{H}(y_1) \cap \mathcal{P}_{T^*}(y) = \emptyset$. Therefore $\mathcal{P}_{T\cup\{y\}}(x) \neq \emptyset$ and T' is irredundant.

Now, by the GreedyPair Algorithm and uniqueness of the greedy pair, we can easily check that the greedy pair of $\mathcal{P}_{T'}(x)$ is exactly $(Y \setminus \{\min(Y)\}, (Z_y)_{y\in Y\setminus\{\min(Y)\}})$ and $T' \cup (Y \setminus \{\min(Y)\}) \setminus ((Z_y)_{y\in Y\setminus\{\min(Y)\}}) = T^*$ is the parent of T'. ◀

▶ **Lemma 21.** *Suppose that $Y_0 \neq \emptyset$. Let T be a child of T^* with respect to $(x, Y, (Z_y)_{y\in Y})$, with $Y \subseteq R \cup Y_0$ and $|Y \setminus Y_0| \geq 1$. Then, $T' := (T \cup \{\min(Y \setminus Y_0)\}) \setminus Z_{\min(Y\setminus Y_0)}$ is a child of T^* with respect to $(x, Y \setminus \{\min(Y \setminus Y_0)\}, (Z_y)_{y\in Y\setminus\{\min(Y\setminus Y_0)\}})$.*

Proof. Recall by Lemma 18 that $Y_0 \subsetneq Y$. By the similar argument as in Lemma 20, we can conclude that T' is a minimal transversal of $\mathcal{H}$ as in addition $\mathcal{P}_T(x) \setminus (\cup_{y\in Y\setminus Y_0}\mathcal{P}_{T^*}(y)) \neq \emptyset$. Again, GreedyPair Algorithm shows that T' is a child of T^* with respect to $(x, Y \setminus \{\min(Y \setminus Y_0)\}, (Z_y)_{y\in Y\setminus\{\min(Y\setminus Y_0)\}})$. ◀

In the following, we want to characterise the other children of T^*. For $1 \leq i \leq |R|$, we call T a *level-i child* of T^* if T is a child with respect to $(x, Y, (Z_y)_{y\in Y})$ with $|Y \setminus Y_0| = i$. We have seen by Lemmas 20 and 21 that if T is a level-i child of T^* with respect to $(x, Y, (Z_y)_{y\in Y})$, then $(T \setminus Z_{\min(Y\setminus Y_0)}) \cup \{\min(Y \setminus y_0)\}$ is a level-$(i-1)$ child of T^* with respect to $(x, Y \setminus \{\min(Y \setminus Y_0)\}, (Z_y)_{y\in Y\setminus\{\min(Y\setminus Y_0)\}})$. We have already seen how to generate the level-0 child. It remains now to explain how to generate the level-i children from the level-$(i-1)$ children.

Let T be a level-$(i-1)$ child of T^* with respect to $(x, Y, (Z_y)_{y\in Y})$. Let $\mathbf{Correct}(Y) := \{y \in R \setminus Y \mid y < \min(Y \setminus Y_0)\}$. Recall that $\mathbf{Correct}(Y) \subseteq T$. For $y \in \mathbf{Correct}(Y)$, we let $C_y := V(\mathcal{P}_{T\setminus\{x\}}(y)) \setminus (T^* \cup E \cup \{w \in V(\mathcal{H}) \mid \exists t \in T \setminus \{y\} \text{ s.t. } \mathcal{P}_{T\setminus\{y\}}(t) \subseteq \mathcal{H}(w)\})$. The set C_y is the set of vertices that are candidates, other than x, for computing minimal transversals

of $\mathcal{P}_{T^*}(y)$, and such that they do not break the privates of any other vertices in T^*, except those of y. The following characterises the level-i children from level-$(i-1)$ children.

▶ **Proposition 22.** *Let $i \geq 1$. T is a level-i child of T^* if and only if there is T' a level-$(i-1)$ child of T^* with respect to $(x, Y, (Z_y)_{y\in Y})$ and $y_1 \in \mathbf{Correct}(Y)$ such that $(Y \cup \{y_1\}, (Z_y)_{y\in Y})$ is the greedy pair of $\mathcal{P}_{T'\setminus\{y_1\}}(x)$ and $T := (T'\setminus\{y_1\}) \cup Z_{y_1}$ with $Z_{y_1} \cup \{x\}$ a minimal transversal of $\mathcal{P}_{T'\setminus\{x\}}(y_1)[C_{y_1} \cup \{x\}]$.*

Proof. Let T be a level-i child of T^* with respect to $(x, Y, (Z_y)_{y\in Y})$ and let $y_1 := \min(Y \setminus Y_0)$. By Lemma 21, $T' := (T \cup \{y_1\}) \setminus Z_{y_1}$ is a child of T^* with respect to $(x, Y\setminus\{y_1\}, (Z_y)_{y\in Y\setminus\{y_1\}})$ and by Lemma 19, $(Y, (Z'_y)_{y\in Y})$ is the greedy pair of $\mathcal{P}_{T'\setminus\{y_1\}}(x)$ where

$$Z'_y := \begin{cases} Z_y & \text{if } y \neq y_1, \\ \emptyset & \text{otherwise.} \end{cases}$$

By Lemma 14, $Z_{y_1} \cup \{x\}$ is a minimal transversal of $\mathcal{P}_{T'\setminus\{x\}}(y)[C_{y_1} \cup \{x\}]$.

Let T' be a level-$(i-1)$ child of T^* with respect to $(x, Y, (Z_y)_{y\in Y})$ and let $y_1 \in \mathbf{Correct}(Y)$ be such that $(Y \cup \{y_1\}, (Z_y)_{y\in Y})$ is the greedy pair of $\mathcal{P}_{T'\setminus\{y_1\}}(x)$. Let $Z_{y_1} \subseteq C_{y_1}$ such that $Z_{y_1} \cup \{x\}$ is a minimal transversal of $\mathcal{P}_{T'\setminus\{x\}}(y_1)$. Let $T := (T' \setminus \{y_1\}) \cup Z_{y_1}$, which by definition is a transversal of $\mathcal{H}$. By definition of C_{y_1}, no $z \in Z_{y_1}$ breaks the privates, with respect to $T' \setminus \{y_1\}$, of some vertex $t \in T' \setminus \{y\}$ and since $Z_{y_1} \cup \{x\}$ is a minimal transversal of $\mathcal{P}_{T'\setminus\{x\}}(y_1)$, each vertex in $Z_{y_1} \cup \{x\}$ has a private with respect to T. Now, if there are $z, z' \in Z_{y_1} \cup \{x\}$ that break the privates, with respect to T, of some $t \in T' \setminus (\{x\} \cup Z_{y_1})$, then by definition of C_{y_1}, there are $F_1, F_2 \in \mathcal{P}_{T'\setminus\{y_1\}}(t)$, $F \in \mathcal{P}_{Z_{y_1}\cup\{x\}}(z)$, $F' \in \mathcal{P}_{Z_{y_1}\cup\{x\}}(z')$, both belonging to $\mathcal{P}_{T'\setminus\{x\}}(y)$. But, then $(z, F_1, t, F_2, z', F', y, F, z)$ is a 4-hole in $\mathcal{H}$. So, T is a minimal transversal of $\mathcal{H}$. By Lemma 19, $(Y \cup \{y_1\}, (Z'_y)_{y\in Y\cup\{y_1\}})$ is the greedy pair of $\mathcal{P}_T(x)$ where

$$Z'_y := \begin{cases} Z_y & \text{if } y \neq y_1, \\ Z_{y_1} & \text{otherwise.} \end{cases}$$

Therefore, T is a level-i child of T^*. ◀

Combining Lemma 21 and Proposition 22, we are now ready to prove that the algorithm Enum given in Algorithm 2 runs with polynomial delay.

Proof of Theorem 1. We claim that the algorithm Enum depicted in Algorithm 2 and combined with the algorithm Enum-Children enumerates the minimal transversals of a $(3, 4)$-hole free hypergraph with polynomial delay.

For any minimal transversal $T \in \mathsf{tr}(\mathcal{H})$, either $T \in \mathbf{Inc}(\mathcal{H}, \ell, E)$ or T belongs to $\mathsf{tr}(\mathcal{H}[V(\mathcal{H}) \setminus \{\ell\}])$. Since we enumerate both in Lines 3-6, we can conclude that the algorithm enumerates all the minimal transversals of $\mathcal{H}$ because of Proposition 16 and also, each non-basic minimal transversal has a parent. It remains to show that we enumerate all the children of each minimal transversal in $\mathbf{Inc}(\mathcal{H}, \ell, E)$.

Now, the function given in Algorithm 3 first outputs the level-0-child if it exists, otherwise it stops. Then, it calls Algorithm 4, which does a DFS on the tree where you have an arc (T', T) if T' is a level-$(i-1)$ child and T is a level-i child obtained from T' as stated in Proposition 22. One can, therefore, conclude that the algorithm correctly outputs all the children of a minimal transversal in $\mathbf{Inc}(\mathcal{H}, \ell, E)$. We can, therefore, conclude that by combining Algorithms 2, 3 and 4 we output exactly the set of minimal transversals.

Algorithm 3: Enum-Children$(\mathcal{H}, \leq, \ell, E, T)$.

Input: A $(3,4)$-hole free hypergraph $\mathcal{H}$, a linear ordering $\leq$ of $V(\mathcal{H})$, a valid double star $S(\ell, E)$ and $T \in \mathbf{Inc}(\mathcal{H}, \ell, E)$.

```
begin
    foreach x ∈ Cov(T) do
        Let Y_0 := {y ∈ T | P_T(y) ⊆ H(x)} and T_0 := ((T \ Y_0) ∪ {x})
        if (Y_0, ∅) is the greedy pair of P_{T\Y_0}(x) then
            if Y_0 ≠ ∅ then
                output T_0
            Let R := {y ∈ T \ (E ∪ Y_0) | P_T(y) ∩ H(x) ≠ ∅}
            Enum-ChildrenAux (H, ≤, ℓ, E, T_0, x, R, Y_0)
```

Algorithm 4: Enum-ChildrenAux$(\mathcal{H}, \leq, \ell, E, T, x, R, Y)$.

Input: A $(3,4)$-hole free hypergraph $\mathcal{H}$, a linear ordering $\leq$ of $V(\mathcal{H})$, a valid double star $S(\ell, E)$, T a transversal of $\mathcal{H}$, a succedent $x \in T$, R the set of candidates and Y the already chosen candidates.

```
begin
    foreach y ∈ Correct(Y) do
        if Y ∪ {y} is the Y-part of the greedy pair of P_{T\{y}}(x) then
            foreach Z_y containing x in Enum (P_{T\{x}}(x)[C_y ∪ {x}], ≤) do
                output ((T ∪ Z_y) \ {y})
                Enum-ChildrenAux (H, ≤, ℓ, E, (T ∪ Z_y) \ {y}), x, R, Y ∪ {y})
```

Let us now analyse its time complexity. Let $n := |V(\mathcal{H})| + \sum_{E \in \mathcal{H}} |E|$. We first notice that one can combine both algorithms Enum and Enum-ChildrenAux into a single one which will do a DFS traversal of the tree of recursive calls (see for instance [2, 21]). Second, each call of Enum or of Enum-ChildrenAux, after a pre-processing polynomial in n, either outputs a new minimal transversal or exits. Therefore, the tree of combined recursive calls of both algorithms has size bounded by $O(n^c) \cdot |\mathsf{tr}(\mathcal{H})|$, for some universal constant c, *i.e.*, the amortised time complexity of the algorithm is $O(n^c)$. By using the same technique as in [28], which consists in outputting a solution at the beginning when the depth of the recursive call is odd, and at the end when the depth is even, one obtains the desired delay per solution. ◀

4 Related results and Conclusion

Enumerating all the minimal dominating sets of a graph is another interesting task in the area of enumeration algorithms with numerous applications (see for instance [20]). Our algorithm readily can enumerate the minimal dominating sets of graphs of girth at least 9. We refer to [26] and [17] for related works. The result of this paper also, gives an incremental delay algorithm for enumerating all the minimal dominating sets in a bipartite graph without induced cycles of length 6 and 8.

▶ **Corollary 2.**

a. *There is a polynomial delay algorithm that enumerates all the minimal dominating sets of a given input graph G of girth at least 9. More precisely, the result holds if G does not contain induced $(4, 5, 6, 7, 8)$-cycles.*

b. *There is an incremental delay algorithm that enumerates all the minimal dominating sets of a given bipartite graph without chordless 6 and 8-cycles.*

Proof of Corollary 2.

a. Let G be a graph of girth at least 9. Let $\mathcal{N}[G]$ be the hypergraph $\{N[x] \mid x \in V(G)\}$ where $N[x]$ is the set $\{x\} \cup \{y \in V(G) \mid y \text{ a neighbour of } x\}$. It is well-known that D is a minimal dominating set of G if and only if D is a minimal transversal of $\mathcal{N}[G]$. One can easily check that if G does not contain a chordless cycle of length strictly smaller than 9 and greater than 3, then $\mathcal{N}[G]$ cannot contain a chordless 3 or 4-cycle, because a cycle of length 3 in $\mathcal{N}[G]$ can only be obtained by an induced cycle of length at most 6 in G and a cycle of length 4 in $\mathcal{N}[G]$ by an induced cycle of length at most 8 in G. By Theorem 1, one can enumerate with polynomial delay all the minimal transversals of $\mathcal{N}[G]$.

b. Let $G := (R, B, E)$ be a bipartite graph without chordless 6 and 8-cycles. By Theorem 1 one can enumerate all the minimal sets $D \subseteq R$ such that D dominates B (called *red-blue dominating sets* in [16]). By using the flipping method in [17], one reduces the existence of an incremental delay enumeration algorithm for the minimal dominating sets of G to the existence of a polynomial delay enumeration algorithm for the minimal red-blue dominating sets in induced subgraphs of G. This concludes the proof. ◀

As we have already discussed in the introduction, the vertices of the polyhedron $\mathcal{P}(A, \underline{1}) = \{x \in \mathbb{R}^n \mid Ax \geq \underline{1},\ x \geq \underline{0}\}$ are in bijection with the minimal transversals of the corresponding hypergraph $\mathcal{H}[A]$, where the columns of A correspond to the vertices of $\mathcal{H}[A]$ and the rows of A are incident vectors of the hyperedges of $\mathcal{H}[A]$ [27]. If the coefficient matrix A in the polyhedron is balanced, then the corresponding hypergraph does not contain any odd-hole.

▶ **Theorem 3.** *There is a polynomial delay algorithm for listing the vertices of any given 0/1 polyhedron $\mathcal{P}(A, \underline{1})$ whenever A is a balanced matrix without any submatrix that is the incident matrix of a 4-cycle.*

Proof of Theorem 3. Let A be a balanced matrix without any 4-cycle submatrix. Then, $\mathcal{H}[A]$, the hypergraph corresponding to the matrix A as explained above, does not contain chordless cycles of length 3 or 4. By Theorem 1 one can enumerate with polynomial delay the minimal transversals of $\mathcal{H}[A]$, which by [27] correspond to the vertices of $\mathcal{P}(A, \underline{1})$. ◀

We conclude the paper by observing that even though our algorithm in Theorem 1 is a polynomial delay one, it uses exponential space and it should be interesting to know whether one can modify it in order to use polynomial space. However, there are more challenging questions, and in particular, it is still open whether there is an output-polynomial time algorithm for enumerating the vertices of a polyhedron $\mathcal{P}(A, \underline{1})$ when A is balanced. We just notice that one needs another technique to deal with balanced hypergraphs as our technique cannot avoid the requirement of the hypergraph to be without chordless 4-cycles. A more challenging question in this area asks for the existence of an output-polynomial time algorithm for the vertices of bounded polyhedron [13].

In many enumeration algorithms, like ours in this paper or [2, 17], the enumeration is reduced to traverse a graph with vertex set the set of solutions, and the difficulty is usually how to generate the neighbors of a given solution. In our paper, we solve this problem by a rather technical, but nice parent-child relation based on the structure of the hypergraphs.

However, the used techniques in almost all such papers are ad-hoc (despite the nice attempts in [2]) and the area still lacks a general theory on identifying a large family of combinatorial enumeration problems on which such a technique works finely.

References

1. MHG Anthony and Norman Biggs. *Computational learning theory*, volume 30. Cambridge University Press, 1997.
2. David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Appl. Math.*, 65(1-3):21–46, 1996. First International Colloquium on Graphs and Optimization (GOI), 1992 (Grimentz). doi:10.1016/0166-218X(95)00026-N.
3. James Bailey, Thomas Manoukian, and Kotagiri Ramamohanarao. A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 485–488. IEEE, 2003.
4. Claude Berge. *Hypergraphs: combinatorics of finite sets*, volume 45. Elsevier, 1984.
5. Endre Boros, Khaled Elbassioni, Vladimir Gurvich, Leonid Khachiyan, and Kazuhisa Makino. Dual-bounded generating problems: All minimal integer solutions for a monotone system of linear inequalities. *SIAM Journal on Computing*, 31(5):1624–1643, 2002.
6. Endre Boros, Khaled Elbassioni, Vladimir Gurvich, and Hans Raj Tiwary. The negative cycles polyhedron and hardness of checking some polyhedral properties. *Annals of Operations Research*, 188(1):63–76, 2011.
7. Endre Boros, Vladimir Gurvich, and Peter L. Hammer. Dual subimplicants of positive Boolean functions. *Optim. Methods Softw.*, 10(2):147–156, 1998. Dedicated to Professor Masao Iri on the occasion of his 65th birthday. doi:10.1080/10556789808805708.
8. Endre Boros, Vladimir Gurvich, Leonid Khachiyan, and Kazuhisa Makino. On the complexity of generating maximal frequent and minimal infrequent sets. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 133–141. Springer, 2002.
9. Michael R Bussieck and Marco E Lübbecke. The vertex set of a 0/1-polytope is strongly p-enumerable. *Computational Geometry*, 11(2):103–109, 1998.
10. Michele Conforti, Gérard Cornuéjols, and MR Rao. Decomposition of balanced matrices. *Journal of Combinatorial Theory, Series B*, 77(2):292–406, 1999.
11. T. Eiter, G. Gottlob, and K. Makino. New results on monotone dualization and generating hypergraph transversals. *SIAM J. Comput.*, 32(2):514–537, 2003. doi:10.1137/S009753970240639X.
12. Thomas Eiter and Georg Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.
13. Khaled Elbassioni and Kazuhisa Makino. Enumerating Vertices of 0/1-Polyhedra associated with 0/1-Totally Unimodular Matrices. In David Eppstein, editor, *16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*, volume 101 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.SWAT.2018.18.
14. Michael L Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628, 1996.
15. Komei Fukuda. Lecture: polyhedral computation. Technical report, Research Report, Department of Mathematics, and Institute of Theoretical Computer Science ETH Zurich, available online, 2004.
16. Petr A. Golovach, Pinar Heggernes, Mamadou Moustapha Kanté, Dieter Kratsch, Sigve Hortemo Sæther, and Yngve Villanger. Output-polynomial enumeration on graphs of bounded (local) linear mim-width. *Algorithmica*, 80(2):714–741, 2018. doi:10.1007/s00453-017-0289-1.

17 Petr A Golovach, Pinar Heggernes, Dieter Kratsch, and Yngve Villanger. An incremental polynomial time algorithm to enumerate all minimal edge dominating sets. *Algorithmica*, 72(3):836–859, 2015.

18 Dimitrios Gunopulos, Heikki Mannila, Roni Khardon, and Hannu Toivonen. Data mining, hypergraph transversals, and machine learning. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 209–216. ACM, 1997.

19 VA Gurvich. On theory of multistep games. *USSR Computational Mathematics and Mathematical Physics*, 13(6):143–161, 1973.

20 M. M. Kanté, V. Limouzy, A. Mary, and L. Nourine. On the enumeration of minimal dominating sets and related notions. *SIAM J. Discrete Math.*, 28(4):1916–1929, 2014. doi:10.1137/120862612.

21 Dimitris J. Kavvadias and Elias C. Stavropoulos. An efficient algorithm for the transversal hypergraph generation. *J. Graph Algorithms Appl.*, 9(2):239–264 (electronic), 2005. doi:10.7155/jgaa.00107.

22 L. Khachiyan, E. Boros, K. Borys, K. M. Elbassioni, V. Gurvich, and K. Makino. Generating cut conjunctions in graphs and related problems. *Algorithmica*, 51(3):239–263, 2008. doi:10.1007/s00453-007-9111-9.

23 Leonid Khachiyan, Endre Boros, Konrad Borys, Vladimir Gurvich, and Khaled Elbassioni. Generating all vertices of a polyhedron is hard. In *Twentieth Anniversary Volume:*, pages 1–17. Springer, 2009.

24 Leonid Khachiyan, Endre Boros, Khaled Elbassioni, and Vladimir Gurvich. An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals and its application in joint generation. *Discrete Applied Mathematics*, 154(16):2350–2372, 2006.

25 Leonid Khachiyan, Endre Boros, Khaled M. Elbassioni, and Vladimir Gurvich. On the dualization of hypergraphs with bounded edge-intersections and other related classes of hypergraphs. *Theor. Comput. Sci.*, 382(2):139–150, 2007. doi:10.1016/j.tcs.2007.03.005.

26 Kazuhiro Kurita, Kunihiro Wasa, Hiroki Arimura, and Takeaki Uno. Efficient enumeration of dominating sets for sparse graphs. *arXiv preprint arXiv:1802.07863*, 2018.

27 Alfred Lehman. On the width-length inequality. *Mathematical Programming*, 16(1):245–259, 1979.

28 Kazuhisa Makino and Takeaki Uno. New algorithms for enumerating all maximal cliques. In Torben Hagerup and Jyrki Katajainen, editors, *Algorithm Theory - SWAT 2004, 9th Scandinavian Workshop on Algorithm Theory, Humlebaek, Denmark, July 8-10, 2004, Proceedings*, volume 3111 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 2004. doi:10.1007/978-3-540-27810-8_23.

29 Ronald C Read. Every one a winner or how to avoid isomorphism search when cataloguing combinatorial configurations. In *Annals of Discrete Mathematics*, volume 2, pages 107–120. Elsevier, 1978.

30 Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

31 Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pages 488–495, 2009.

32 Kunihiro Wasa. Enumeration of enumeration algorithms. *arXiv preprint arXiv:1605.05102*, 2016.

Collective Fast Delivery by Energy-Efficient Agents

Andreas Bärtschi
Department of Computer Science, ETH Zürich, Switzerland
andreas.baertschi@inf.ethz.ch

Daniel Graf
Department of Computer Science, ETH Zürich, Switzerland
daniel.graf@inf.ethz.ch

Matúš Mihalák
Department of Data Science and Knowledge Engineering, Maastricht University, Netherlands
matus.mihalak@maastrichtuniversity.nl

Abstract

We consider k mobile agents initially located at distinct nodes of an undirected graph (on n nodes, with edge lengths). The agents have to deliver a single item from a given source node s to a given target node t. The agents can move along the edges of the graph, starting at time 0, with respect to the following: Each agent i has a *weight* ω_i that defines the rate of energy consumption while travelling a distance in the graph, and a *velocity* v_i with which it can move.

We are interested in schedules (operating the k agents) that result in a small *delivery time* $\mathcal{T}$ (time when the item arrives at t), and small *total energy consumption* $\mathcal{E}$. Concretely, we ask for a schedule that: either (i) Minimizes $\mathcal{T}$, (ii) Minimizes lexicographically $(\mathcal{T}, \mathcal{E})$ (prioritizing fast delivery), or (iii) Minimizes $\epsilon \cdot \mathcal{T} + (1-\epsilon) \cdot \mathcal{E}$, for a given $\epsilon \in (0, 1)$.

We show that (i) is solvable in polynomial time, and show that (ii) is polynomial-time solvable for uniform velocities and solvable in time $\mathcal{O}(n + k \log k)$ for arbitrary velocities on paths, but in general is NP-hard even on planar graphs. As a corollary of our hardness result, (iii) is NP-hard, too. We show that there is a 2-approximation algorithm for (iii) using a single agent.

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

Keywords and phrases delivery, mobile agents, time/energy optimization, complexity, algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.56

Funding This work was partially supported by the SNF (project 200021L_156620, Algorithm Design for Microrobots with Energy Constraints).

1 Introduction

Technological development has allowed for low-cost mass production of small and simple mobile robots. Autonomous vacuum cleaners, mowers, or drones are some of the best known examples. There are attempts to deploy such autonomous *agents* to deliver physical goods – *packages* [24, 26]. In the future, for delivering over longer distances, a *swarm* of such autonomous agents is a likely option to be adapted, since the energy supply of the agents is limited, or the agents are simply required to operate locally, or simply because the usage of some agents is more costly than others. A careful cooperation and planning of the agents is thus necessary to provide energy, time, and cost efficient delivery. This leads to plentiful optimization problems regarding the operation of the agents.

Here we consider the problem of delivering a single package as quickly as possible from a source node s to a target node t in a graph $G = (V, E)$ with edge lengths by a team of k agents. The agents have individual velocities, with which they can move along the edges of the graph, and also an energy-consumption rate for a travelled unit distance. The goal is to design *centralized algorithms* to coordinate the agents such that the package is delivered from s to t in an efficient way. In the literature, delivery problems focusing solely on energy efficiency have been studied. One research direction considers every agent to have an initial amount of energy (battery) that restricts the agents' movements [1, 11]. The decision problem of whether the agents can deliver the package has been shown to be strongly NP-hard on planar graphs [6, 7] and weakly NP-hard on paths [12], and it remains NP-hard on general graphs even if the agents can exchange energy [13]. The second research direction considers every agent to have unlimited energy supply, and an individual energy-consumption rate per travelled distance [8, 9]. The problem of delivering the package and minimizing the total energy consumption can be solved in time $\mathcal{O}(k + n^3)$ [8].

In this paper, we primarily focus on delivering the package in a quickest possible way, and only secondarily on the total energy that is consumed by the agents. This has not been, to the best of our knowledge, studied before. Specifically, we consider the algorithmic problem of finding a delivery schedule that: (i) minimizes the delivery time, (ii) minimizes the delivery time using the least amount of energy, and (iii) minimizes a linear combination of delivery time and energy consumption.

Our model. We are given an undirected graph $G = (V, E)$ on $n = |V|$ nodes. Each edge $e \in E$ has a positive *length* l_e. The length of a path is the sum of the lengths of its edges. We consider every edge $e = \{u, v\}$ to consist of infinitely many points, where every point is uniquely characterized by its distance from u, which is between 0 and l_e. We consider every such point to subdivide the edge $\{u, v\}$ into two edges of lengths proportional to the position of the point on the edge. The distance $d_G(p, q)$ between two points p and q (nodes or points inside edges) of the graph is the length of a shortest path from p to q in G. There are k mobile agents initially placed on nodes $p_1, \ldots, p_k$ of G. Every agent $i = 1, \ldots, k$ has a *weight* $0 \leq \omega_i < \infty$ and a *velocity* $0 < \upsilon_i \leq \infty$. Agents can traverse the edges of the graph. To traverse an edge e (in either direction), agent i needs time l_e/υ_i and $\omega_i \cdot l_e$ units of energy.

Furthermore there is a single package, initially (at time 0) placed on a source node s, which has to be delivered to a given target node t. Each agent can walk from its current location to the current location of the package (along a path in the graph), pick the package up, carry it to another location (a point of the graph), and drop it there. From this moment, another agent can pick up the package again. Only the moving in the graph takes time – picking up the package and dropping it off is done instantaneously. (The time spent by the package being dropped at a point until picked up again is, however, taken into account.)

We call a schedule that operates the agents such that the package is delivered a *solution*. In such a schedule S, we denote by $d_i(S)$ the total distance travelled by agent i, and by $d_i^*(S)$ the distance travelled by agent i while carrying the package. The total energy consumption of the solution is thus $\mathcal{E}(S) = \sum_{i=1}^{k} \omega_i \cdot d_i(S)$ and the time needed to deliver the package is given by $\mathcal{T}(S) = \sum_{i=1}^{k} d_i^*(S)/\upsilon_i +$ (the overall time the package is not carried). *Fast and energy-efficient* Delivery is the optimization problem of finding a solution that has small delivery time $\mathcal{T}$ as well as total energy consumption $\mathcal{E}$. In particular, we study the following three objectives (see Figure 1 for illustration):

(i) Minimize the delivery time $\mathcal{T}$.

(ii) Lexicographically minimize the tuple $(\mathcal{T}, \mathcal{E})$, i.e. among all solutions with minimum $\mathcal{T}$, find a solution that has minimum energy consumption $\mathcal{E}$.

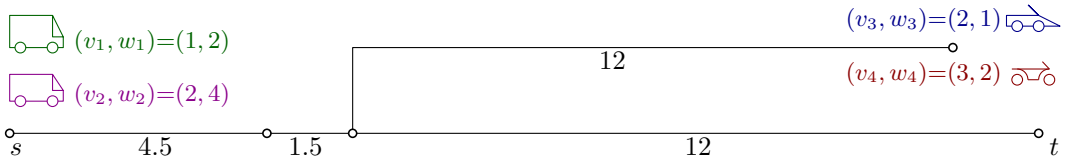


Figure 1 Example for optima of variants of fast *and* energy-efficient DELIVERY:
(ii) Using agents 2 and 4, we get $(\mathcal{T}, \mathcal{E}) = (\max\{6/2, 12/3\} + 12/3,\ 4 \cdot 6 + 2 \cdot 12 + 2 \cdot 12) = (8, 72)$.
(iii) For $\epsilon = \frac{4}{5}$, using agents 1 and 4, we get $\frac{4}{5}\mathcal{T} = \frac{4}{5}(\max\{4.5/1, (12+1.5)/3\} + (1.5+12)/3)$ and $\frac{1}{5}\mathcal{E} = \frac{1}{5}(4.5 \cdot 2 + (12+1.5) \cdot 2 + (1.5+12) \cdot 2)$ for a combined total of $\frac{4}{5}(4.5+4.5) + \frac{1}{5}(9+27+27) = 19.8$.
(iv) Using agents 1 and 3, we get $(\mathcal{E}, \mathcal{T}) = (2 \cdot 6 + 1 \cdot 12 + 1 \cdot 12,\ \max\{6/1, 12/2\} + 12/2) = (36, 12)$.

(iii) Minimize a convex combination $\epsilon \cdot \mathcal{T} + (1-\epsilon) \cdot \mathcal{E}$, for some given value $\epsilon \in (0,1)$.
Recent parallel work studied the following complementary – energy focused – variants:
(iv) Lexicographically minimize the tuple $(\mathcal{E}, \mathcal{T})$, i.e. prioritize the minimization of $\mathcal{E}$ [10].
(v) Minimize the energy consumption $\mathcal{E}$ [8, 9].
In all variants it is natural to (without loss of generality) only consider simple paths as the trajectory of the package, i.e., if at times t_1, t_2 $(0 \le t_1 \le t_2 \le \mathcal{T})$ the package is at the same position p, then it remains at position p for the time in-between $(\forall t \in [t_1, t_2])$. We will make this assumption throughout this paper.

Our contribution. First, in Section 2, we prove for the first time that optimum solutions exist for all mentioned variants of DELIVERY (while previous work on (iv) and (v) implicitly assumed this). Then, in Section 3, we investigate the problem of minimizing the delivery time $\mathcal{T}$ only. We call this optimization problem FASTDELIVERY and show that there is a polynomial-time dynamic program of time complexity $\mathcal{O}(k^2|E| + k|V|^2 + \text{APSP}) \subseteq \mathcal{O}(k^2n^2 + n^3)$, where $\mathcal{O}(\text{APSP})$ is the running time of an all-pair shortest path algorithm for undirected graphs.

In Section 4, we study FASTEFFICIENTDELIVERY, prioritizing the delivery time $\mathcal{T}$ over the energy consumption $\mathcal{E}$. We first show that the problem can be solved in polynomial time for uniform velocities. However, we prove the problem to be NP-hard for general velocities even on planar graphs. We therefore consider the restricted graph class of *paths*, in which we can decompose the problem into uniform velocity instances. For each such instance, we establish a characterization of handover points. Using geometric point-line duality [18] and dynamic planar convex hull techniques [4], we give an $\mathcal{O}(n + k \log k)$ algorithm for paths.

In Section 5, we show that for arbitrary given weights $\epsilon \in (0,1)$, the minimum convex combination $\epsilon \cdot \mathcal{T} + (1-\epsilon) \cdot \mathcal{E}$ can be 2-approximated by a single agent, while NP-hardness follows from an adaptation of the hardness proof in the preceding section. We call the task of minimizing the convex combination COMBINEDDELIVERY. Finally, in Section 6 we discuss several extended models to which our approach can be generalized. Due to the limited space, some proofs are omitted, but are provided in a thesis on several variants of DELIVERY [5].

Comparison to related work. Among the earliest problems related to DELIVERY are the Chinese Postman Problem [19] and the Traveling Salesman Problem [2], in which a single agent has to visit multiple destinations located in edges or nodes of the graph, respectively. The latter has given rise to a class of problems known as Vehicle Routing Problems [25], which are concerned with the distribution of goods by a fleet of (homogeneous) vehicles under additional hard constraints such as time windows. Minimizing the total or the maximum travel distance of a group of agents for several tasks such as the formation of configurations [17] or the visit of designated arcs [20] have been studied for identical agents

as well. Energy-efficient Delivery (without optimization of delivery time) has been recently introduced [8] for an arbitrary number of packages, with handovers restricted to take place at nodes of the graph only. This setting turns out to be NP-hard, but can be solved in polynomial-time for a single package, in which case the restriction of handovers to nodes becomes irrelevant (there is always an optimal solution which does not use any in-edge handovers). To the best of our knowledge, this present paper and a parallel work [10] on variant (iv) are the only ones studying the Delivery problem with agents which have different velocities. Similar to our approach, the latter studies a uniform weight setting first. The uniform weight result is then used as a subroutine in a dynamic program for general weights. Our hardness result shows that such an approach (combination of uniform velocities) is not possible for FastEfficientDelivery, even on planar graphs. Finally, mobile agents with distinct maximal velocities have been getting attention in areas such as searching [3], walking [14] and patrolling [15].

2 Preliminaries

We first formally establish that optimum solutions for all variants of efficient Delivery exist. To this end, each solution which operates agents $i_1, i_2, \ldots, i_\ell$ in this order can be represented by the drop-off locations of these agents only (note that for two consecutive agents i, j, the drop-off location of agent i, denoted by q_i^-, corresponds to the pick-up location of agent j, denoted by q_j^+). Since we allow in-edge handovers, there are infinitely many solutions – however, these can be divided into *finitely* many topologically compact sets. As $\mathcal{E}, \mathcal{T}$ act as continuous functions on these sets, we have in each set a minimum solution.

▶ **Theorem 1** (Existence of optimum solutions)**.** *There exists an optimum solution minimizing the delivery time $\mathcal{T}$ (the energy consumption $\mathcal{E}$, or $\epsilon \cdot \mathcal{T} + (1-\epsilon) \cdot \mathcal{E}$, $(\mathcal{T}, \mathcal{E})$, $(\mathcal{E}, \mathcal{T})$, respectively).*

3 Optimizing delivery time only

Throughout this section, we assume that all agents have weight $\omega_i = 0$. Hence in all three variants of fast energy-efficient Delivery, $\mathcal{E} = 0$ and we are after a solution for delivery with earliest-possible delivery time. We show that FastDelivery is polynomial-time solvable, due to the following characterization of optimum solutions (which exist by Theorem 1):

▶ **Lemma 2.** *For every instance of* FastDelivery*, there is an optimum solution in which (i) the velocities of the involved agents are strictly increasing, (ii) no involved agent arrives at its pick-up location earlier than the package (carried by the preceding agent), and (iii) if more than one agent is involved in transporting the package over an edge $\{u, v\}$ in direction from u to v, then only the first involved agent will ever visit u.*

Proof. All three properties can be shown by exchange arguments. Taking any optimum solution, we turn it into an optimum solution that adheres to the three properties as follows:

(i) Label the agents $1, 2, \ldots, i, \ldots$ in the order in which they transport the package. Let i be the first agent such that $v_i \geq v_{i+1}$. Now we can simply replace agent $i+1$ by letting agent i travel on the same trajectory on which $i+1$ transported the package; and by doing so, we don't increase the delivery time.

(ii) Let i be the first agent that has to wait at its pick-up location for the package to arrive. Instead of waiting, we let i proceed on the original trajectory of the package towards s until it meets the preceding agent $i-1$. Handing over the package at this new spot cannot

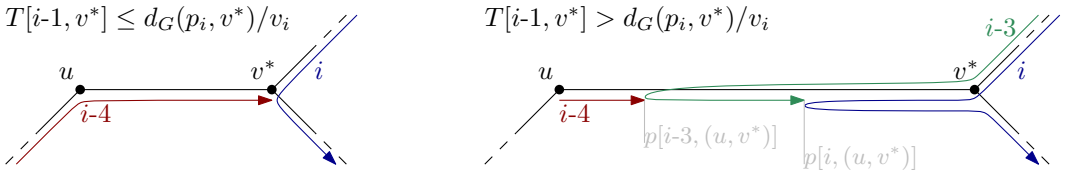


Figure 2 Examples for cases a) and b): (left) Agent i picks up the package at node v^*. (right) Agent i picks up the package inside the edge (u, v^*) at the earliest possible time.

increase the delivery time $\mathcal{T}$, as $v_{i-1} < v_i$ (we only increase velocities along the trajectory). However, $\mathcal{T}$ might remain constant if this increase in velocity is countered by a longer waiting time of the package at the handover to agent $i + 1$.

(iii) Assume that multiple agents bring the package from u to v over the edge $\{u, v\}$, by visiting u first. By assumption (i) the last such agent i has the highest velocity and thus agent i can just as well pick up the package at u without the help of the other agents. ◀

▶ **Corollary 3.** *After a preprocessing step of time $\mathcal{O}(k + |V|)$ – in which we remove in each node all but the agent with maximum velocity v_i – we may assume that $k \leq |V|$.*

Towards a dynamic program. Making use of characterization (i) of Lemma 2, we relabel the agents such that $v_1 \leq v_2 \leq \ldots \leq v_k$. We can then look at subproblems where we only use the first $i - 1$ among all k agents. Assume node v^* is the first node that the new agent i (starting at p_i) passes while actually carrying the package. According to characterizations (ii) and (iii), when defining the recursion, we have to take care of these two cases, see Figure 2:

a) Agent i might arrive at node v^* 'late', the package has already been dropped off there before by one of the agents $1, 2, \ldots, i - 1$ and had been waiting.

b) Agent i might arrive at node v^* 'early', in which case it should walk towards the package to receive it earlier and bring it back to v^* faster (having larger velocity than the currently carrying agent, after all). In this case, agent i picks up the package at a point p which is strictly in the interior of the edge $\{u, v^*\}$ and which is as close to node v^* as possible, i.e., p must be reachable by both agent i *and* the package – carried by only the first $i - 1$ agents – at the *earliest* possible time: $(d(p_i, v^*) + d(v^*, p))/v_i$.

Dynamic Program. First we are interested in the distance between any two nodes in the graph, which we can find with an *all-pair-shortest-paths* algorithm APSP. We denote the time needed for this precomputation by $\mathcal{O}(\text{APSP})$. Then, given the agents in ascending order of their velocities v_i, for each prefix $1, 2, \ldots i$ of the agent order and each node v we define the following subproblem:

$$
\begin{aligned}
S\,[i, v] &= \text{A fastest schedule to bring the package to node } v \text{ using agents } \{1, \ldots, i\}.\\
\mathcal{T}\,[i, v] &= \text{The time needed in } S\,[i, v] \text{ to deliver the package to } v.\\
A\,[i, v] &= \text{Index of the last agent to carry the package in } S\,[i, v].\\
p\,[i, (u, v)] &= \text{The pick-up point } p \text{ strictly inside edge } \{u, v\} \text{ and closest to } v, \text{ reachable}\\
&\quad \text{by } \textit{both} \text{ the package (coming from } u, \text{ delivered by agents } 1, \ldots, i-1)\\
&\quad \textit{and} \text{ agent } i \text{ (coming via } v) \text{ in time } (d(p_i, v) + d(v, p))/v_i \text{ (if applicable).}
\end{aligned}
$$

Note that although our graph only has undirected edges, $p[i, (u, v)]$ considers an *ordered* tuple of nodes (u, v), denoting that the package is transported from u to v. Thus $p[i, (v, u)]$

has the analogous meaning of the package crossing edge $\{u, v\}$ from v towards u. Both $p[i, (u, v)]$ and $p[i, (v, u)]$ might be undefined, as can be seen below.

We compute the optimum delivery times $\mathcal{T}[i, v]$ (together with $A[i, v]$) without explicitly maintaining the schedules $S[i, v]$. A concrete final schedule S can then be retraced from $A[,]$, see Theorem 4. For computing $\mathcal{T}[i, v]$ and $A[i, v]$ we 'guess' the first node v^* of cases a) and b) above by trying each node v as a candidate. We then can compute $\mathcal{T}[i, v]$ and $A[i, v]$ for all other nodes using the pre-computed distances between all pair of nodes:

1. *Initialization:* For all nodes v, we initialize $S[i, v] := S[i-1, v], A[i, v] := A[i-1, v]$ and $\mathcal{T}[i, v] := \mathcal{T}[i-1, v]$. This automatically takes care of case a), where the package arrives at v before agent i can reach v.
2. *In-edge pick-ups:* We go over all node pairs (u, v) such that $\{u, v\} \in E$ and check whether agent i can pick up the package inside $\{u, v\}$ to advance it to node v faster than in schedule $S[i-1, v]$. We do so by checking whether we have $d(p_i, v)/v_i < \mathcal{T}[i-1, v]$ *and* $d(p_i, u)/v_i > \mathcal{T}[i-1, u]$. In this case, agent i receives the package from a previous agent j that brought it from u or from $p[j, (u, v)]$. Thus we get a set P of candidates for $p[i, (u, v)] := \arg\min_{p \in P} \{d(p, v)\}$. The candidate set P consists of all points p strictly inside the edge $\{u, v\}$ such that there exists an agent of index j, $A[i-1, u] \leq j < i$, for which we have

$$\max\left\{\mathcal{T}[i-1, u], \frac{d(p_j, u)}{v_j}\right\} + \frac{d(u, p)}{v_j} = \frac{d(p_i, v) \ + \ d(v, p)}{v_i}$$

   if j is coming from u, *or* – if $p[j, (u, v)]$ is defined –

$$\frac{d(p_j, v) \ + \ d(v, p[j, (u, v)]) \ + \ d(p[j, (u, v)], p)}{v_j} = \frac{d(p_i, v) \ + \ d(v, p)}{v_i}.$$

   Having computed $p[i, (u, v)]$ as the point in P closest to v, we update node v accordingly: Set $\mathcal{T}[i, v] := \min\left\{\mathcal{T}[i, v], \frac{d(p_i, v) + 2d(p[i,(u,v)], v)}{v_i}\right\}$, where using 'min' takes care of cases in which we have multiple incident edges to v that all potentially have in-edge pick-ups by i, and set $A[i, v] = i$ (valid since we consider the case where $d(p_i, v) < \mathcal{T}[i-1, v]$).
3. *Updates:* So far we have computed the subproblems $S[i, v]$ correctly, if node v corresponds to the first node v^* of cases a) and b) (in particular we checked whether the faster agent i can help to advance the package over only one edge). Now we also consider all cases where agent i transports the package over arbitrary distances, by updating all other schedules $S[i, u]$ accordingly: For each node v, for each node u, if $\mathcal{T}[i, u] > \max\{\mathcal{T}[i, v], d(p_i, v)/v_i\} + d(v, u)/v_i$ we set $A[i, u] := i$ and $\mathcal{T}[i, u] := \max\{\mathcal{T}[i, v], d(p_i, v)/v_i\} + d(v, u)/v_i$.

▶ **Theorem 4.** *An optimum schedule for* FastDelivery *of a single package can be computed in time* $\mathcal{O}(k^2|E| + k|V|^2 + \text{APSP}) \subseteq \mathcal{O}(k^2n^2 + n^3)$.

Proof. For each i from 1 to k we can compute all values $A[i, v]$, $\mathcal{T}[i, v]$ in time $\mathcal{O}(|V|)$ for the initialization, $\mathcal{O}(|E|k)$ to check for in-edge pick-ups and $\mathcal{O}(|V|^2)$ for the updates (for which we need precomputed all-pair shortest paths). Overall we get a running time of $\mathcal{O}(\text{APSP} + k^2|E| + k|V|^2)$. The delivery time is then given in $\mathcal{T}[k, t]$. Correctness of the algorithm follows from the definition of the subproblems and the case distinction stemming from Lemma 2. Since we did not explicitly maintain the schedules $S[i, v]$, we retrace the concrete schedule S from $A[,]$ by backtracking: Let i denote the last used agent $A[k, t]$. We can find i's 'first node' v^* in time $\mathcal{O}(|V|)$ by searching for the smallest value $\mathcal{T}[i, u]$ such that

$$\max\{\mathcal{T}[i, u], d(p_i, u)/v_i\} = \mathcal{T}[k, t] - d(u, t)/v_i.$$

If $A[i, v^*] \neq i$, we recurse, otherwise we find the correct adjacent node and all in-edge handovers by looking – for each of the $\mathcal{O}(\deg(v^*))$ many neighbors u of v^* – at the overall $\mathcal{O}(k \deg(v^*))$ many values $p[j, (u, v^*)]$ (where $j \leq i$) and $\mathcal{T}[j, u]$ (where $j < i$). ◀

4 Prioritizing delivery time over energy consumption

In this Section, we want to find the most efficient among all fastest delivery schedules. We call this problem FastEfficientDelivery and will first show that it can be solved in polynomial time for uniform velocities ($\forall i, j \colon v_i = v_j$), due to a characterization of optimum schedules. In contrast, we prove NP-hardness for arbitrary speeds, even on planar graphs. However, for paths we show how one can subdivide general instances into phases of concecutive agents having the same velocity, and achieve an efficient $\mathcal{O}(n + k \log k)$-time algorithm.

4.1 A polynomial-time algorithm for uniform velocities

▶ **Lemma 5.** *Consider* FastEfficientDelivery *on instances with uniform agent velocities and let δ denote the offset of the closest agent's starting position to s. Then there exists an optimum schedule such that the pick-up position q_i^+ of each involved agent i satisfies:*

- $d(s, q_i^+) + d(q_i^+, t) = d(s, t)$*, i.e., q_i^+ lies on a shortest s-t-path, and*
- $d(p_i, q_i^+) \leq \delta + d(s, q_i^+)$*, with equality if q_i^+ lies strictly inside an edge.*

Proof. Since all agents have the same velocity $\overline{v}$, any fastest delivery of the package must follow a shortest path from s to t. Furthermore, since the closest agent could deliver the package on its own in time $(\delta + d(s, t))/\overline{v}$, each involved agent i has to arrive at q_i^+ no later than the package itself, giving $d(p_i, q_i^+) \leq \delta + d(s, q_i^+)$. It remains to show that we can modify every optimum solution into an optimum solution in which we have $d(p_i, q_i^+) = \delta + d(s, q_i^+)$ whenever q_i^+ lies strictly inside an edge $e = \{u, v\}$. Denote by i the first agent for which this is not the case and by $i - 1$ its preceding agent. Assume that the package enters e via u (i.e. $d(s, u) < d(s, v)$). Note that i must have entered e via v, since otherwise the energy consumption could be improved by letting i pick up the package already at u (without increasing the delivery time), contradicting the optimality of our solution. Now we distinguish two cases relating the weights ω_i and ω_{i-1}, yielding either a decrease of the energy consumption, or a possibility to move q_i^+ to a position satisfying the characterization.

- $2\omega_i > \omega_{i-1}$: Moving q_i^+ by $\epsilon > 0$ towards v decreases $\mathcal{E}$ by an amount of $(2\omega_i - \omega_{i-1}) \cdot \epsilon > 0$.
- $2\omega_i \leq \omega_{i-1}$: We move q_i^+ towards u (without increasing neither delivery time nor energy consumption) until we reach $q_i^+ = u$, or q_i^+ inside the edge $\{u, v\}$ such that $d(p_i, q_i^+) = \delta + d(s, q_i^+)$, or $q_i^+ = q_{i-1}^+$. In the last case, discarding agent $i - 1$ from our solution results in an energy consumption decrease of at least $\omega_{i-1} \cdot d(p_{i-1}, q_{i-1}^+) > 0$. ◀

Polynomial-time algorithm. We use the characterization in Lemma 5 to find an optimum solution for FastEfficientDelivery of delivery time $\mathcal{T} = (\delta + d(s, t))/\overline{v}$: For each agent i, we compute the set Q_i of all potential pick-up locations, i.e., the set of points q_i that satisfy Lemma 5. The number of potential locations is $|Q_i| \in \mathcal{O}(|V| + |E|) \subseteq \mathcal{O}(n^2)$. Then we build an auxiliary directed acyclic multi-graph H on a node set $V(H) = \bigcup_{i=1}^{k} Q_i$, of size $|V(H)| \in \mathcal{O}(|V| + k|E|) \subseteq \mathcal{O}(kn^2)$. Each directed edge in $E(H)$ describes how agent i can contribute to the delivery by bringing the package from its starting position q_i to another agent's starting position q_j along a shortest s-t-path: For each pair of nodes $q_i \in Q_i$ and $q_j \in V(H)$ such that $q_i \neq q_j$ and $d(s, q_i) + d(q_i, q_j) + d(q_j, t) = d(s, t)$, we add an arc (q_i, q_j) of

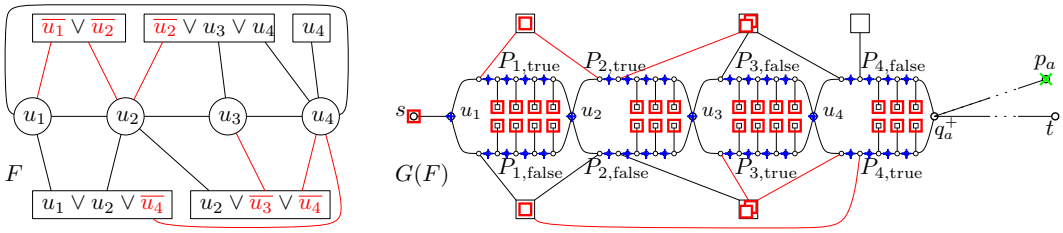


Figure 3 (left) A planar 3CNF formula F, satisfiable by $(u_1, u_2, u_3, u_4) = (\text{true}, \text{false}, \text{false}, \text{true})$. (right) Its transformation into a corresponding delivery graph $G(F)$. The satisfiable assignment of F corresponds to a low-cost delivery in $G(F)$ via paths $P_{1,\text{true}}, P_{2,\text{false}}, P_{3,\text{false}}, P_{4,\text{true}}$, and vice versa. We have slow agents for clauses (□), fast agents for variables/literals (+) and a very fast agent (×).

weight $\omega_i \cdot (d(p_i, q_i) + d(q_i, q_j))$ to $E(H)$. Overall, we have at most $|E(H)| \in \mathcal{O}(k \cdot n^2 \cdot kn^2)$ many arcs. By construction of H, running Dijkstra's shortest path algorithm on the multi-graph H finds a shortest path from s to t corresponding to an optimal solution.

▶ **Theorem 6.** *An optimum solution for* FastEfficientDelivery *can be found in time* $\mathcal{O}(k^2 n^4)$*, assuming all agents have the same velocity.*

4.2 NP-hardness on planar graphs

Contrary to FastDelivery (where we had non-decreasing velocities v_i), when prioritizing delivery time *but still regarding energy consumption*, we can't characterize the order of the agents by their coefficients (v_i, ω_i): Consider an instance in which both the starting position p_a of the absolutely fastest agent a as well as the package destination t are separated from the rest of the graph by two very long edges q_a^+—p_a, q_a^+—t. Then in *every* fastest solution, agent a (with v_a large, e.g. 8) must deliver the package from q_a^+ to t, see Figure 3 (right).

In FastEfficientDelivery, the task is thus to balance slow but efficient agents (with, e.g., $v = 1, \omega = 0$) and fast inefficient agents (with, e.g., $v = 2, \omega = 1$) to collectively deliver the package to a's pick-up location q_a^+ *just-in-time* – i.e., in time $d(p_a, q_a^+)/v_a$ – *without using too much energy*. We can construct suitable instances by a reduction from Planar3SAT [23] *(Sketch)*: Starting from a planar formula F in three-conjunctive normal form, as in Figure 3 (left), we build a delivery graph $G(F)$. This can be done such that the instance is guaranteed to have schedules with minimum delivery time, i.e. with $\mathcal{T} = d(p_a, t)/v_a$. However, there should only be such a minimum-time schedule which simultaneously has low energy consumption $\mathcal{E}$ *if and only if* the formula F has a satisfiable variable assignment.

To this end, we place the fast agents on nodes corresponding to variables and literals. Intuitively, these agents decide on the routing of the package, thus setting the assignment of each variable. The slow agents, on the other hand, are placed on clause nodes, each clause receiving just one agent short of the number of its literals. Intuitively, for a just-in-time delivery to q_a^+ with small energy consumption, each clause has to spend one of its agents for each of its unsatisfied literals. By construction, this is only possible if each clause is satisfied:

▶ **Theorem 7.** FastEfficientDelivery *is* NP*-hard, even on planar graphs.*

4.3 An efficient algorithm for paths

The preceding hardness result raises the question for which restricted graph classes we can expect an efficient algorithm for arbitrary velocity instances. To contribute to this question it is natural to study paths – on paths, the V-shaped p_a—q_a—t component attached to the

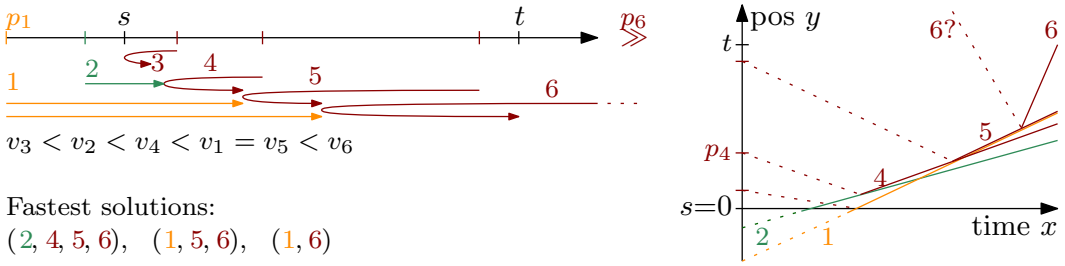


Figure 4 (left) Possible optima: If agent 4 is involved, it must take over the package from agent 2, since agent 3 is too slow. Using agents 2 and 4 to bring the package to agent 5's pick-up position takes the same time as using agent 1 on its own. Agents 1 and 5 have the same velocity, so in terms of delivery time we could use either or even both of them, but agent 1 only if agents 2 and 4 are both not used (otherwise they all consume energy). (right) Fastest solutions correspond to at most 1 agent with $p_i < s$ and a number of agents corresponding to a suffix of the upper envelope.

rest of the graph, as used in the hardness proof, 'collapses' to a line. We show that this allows us to decompose the problem into linearly many uniform velocity instances in time $\mathcal{O}(n + k \log k)$. Theorem 6 then implies that FastEfficientDelivery can be solved in polynomial-time. Improving on this by a careful analysis of paths, we show how to solve each uniform velocity instance in time $\mathcal{O}(n + k \log k)$ as well, and that these instances can be combined in time $\mathcal{O}(k)$, giving an overall $\mathcal{O}(n + k \log k)$-time algorithm.

Decomposition into uniform velocity instances

In the following, we look at the path graph G as the real line, and assume (after performing a depth-first search from s and ordering the starting positions in time $\mathcal{O}(n + k \log k)$) without loss of generality that $s = 0 < t$, that $p_1 \leq p_2 \leq \ldots \leq p_k$ and that $n = k + 2$, as the only relevant nodes on the line are s, t and the starting positions p_i. Note that in an optimum solution of FastEfficientDelivery, no agent i will ever take over the package from another agent j which i overtakes from the left. In particular, this means that we will need at most one agent with starting position $p_i < s$, and that after the package is picked up at s, it will never have to wait between a drop-off by an agent j and a pick-up by the next agent i, since j could continue carrying the package towards i, thus decreasing the overall delivery time. Hence in an optimum schedule we also have for consecutive agents i, j with $s < p_j < p_i$, that $v_j \leq v_i$ (otherwise we can discard i, by this decreasing the delivery time).

Decomposition. Assume that agent i is the agent that delivers the package to t. We represent the trajectory of the package while being carried by i as a ray giving the position y on the real line as a function $f_i(x)$ of the time x passed so far, see Figure 4 (right). We now inductively compute a set containing all functions $f_0, f_1, \ldots, f_k$, where $f_0(x) = s = 0$.

If we have $p_i < 0$, then by the reasoning above, i is the only involved agent, and the function is simply $f_i \colon y = v_i \cdot x + p_i$. For $p_i > s$, the slope v_i of the ray is set, but not its pick-up position. In order to minimize the earliest possible delivery time x (i.e. $f_i(x) = t$), by the non-decreasing velocity property i must pick up the package as early as possible – e.g. in Figure 4 (left), the fastest agent 6 would not get the package from agent 4, but from agent 5 who is able to speed up the transport between agents 4 and 6, thus advancing the last handover position and allowing agent 6 to pick up the package earlier.

Formally, the pick-up position is given by the time-wise first (or in other words leftmost) intersection of a query line $y = p_i - v_i \cdot x$ (modelling the agent moving towards s) with any preceding ray $f_0, \ldots, f_{i-1}$. Let $q_i := (x_i, y_i)$ denote the intersection point of the query

line with the upper envelope of the preceding rays, and denote by f_j a ray of steepest slope v_j among all rays $f_0, \ldots, f_{i-1}$ that contain q_i – e.g. the query line "6?" in Figure 4 (right) intersects both f_1 and f_5 on the upper envelope, and since both have the same slope, we can consider either.

In case $v_j > v_i$, agent i will not be used in an optimal schedule and we set $f_i = 0$. If, however, $v_j \leq v_i$, then f_i is given by the line equality $f_i \colon y = v_i \cdot x + (y_i - v_i \cdot x_i)$. After completion, an optimum schedule corresponds to a path along the rays of our diagram from $(0, 0)$ to the ray reaching $y = t$ at the earliest possible time.

Fast computation and recombination. To quickly compute the equation of each ray f_i, we need to find the intersection of a query line with the upper envelope of $\mathcal{O}(k)$ many rays. Precomputing this envelope as an ordered list of its segments would allow us to speed up the intersection queries from a linear to a binary search (*convex hull trick for dynamic programming* [21]). However, the set of functions that we query here is not known up front. Instead, we apply the classic geometric point-line duality [18]. In this dual setting, the task of finding the leftmost intersection point of a query line with a set of lines turns into finding a right tangent from a query point (the dual of the query line) onto the convex hull of a point set (the dual of the rays f_i). The dynamic planar convex hull data structure by Brodal and Jacob [4, 22] allows point insertions and tangent queries all in $\mathcal{O}(\log k)$ amortized time, giving an overall running time of $\mathcal{O}(k \log k)$. Assuming that we know the optimum schedule for each of the uniform velocity intervals, it remains to recombine these subschedules:

▶ **Lemma 8.** *Arbitrary velocity instances of* FastEfficientDelivery *on paths can be decomposed into and recombined from uniform velocity instances in time* $\mathcal{O}(n + k \log k)$.

A fast algorithm for uniform velocity instances on the line

We are left to solve the case where all agents have the same uniform velocity $\overline{v}$. As before, we denote by δ the offset of the closest agent's starting position to s, and let a denote the corresponding agent. No agent i with $p_i < s$ other than maybe agent a is involved in an optimum schedule (all others would only slow down delivery). Also note that if $p_a < s$, the setting is equivalent to one where a starts at $s + (s - p_a)$, so we can assume (after relabelling the agents) $a = 1$, $\delta = p_1$. This also implies $\mathcal{T} = (\delta + (t - s))/\overline{v}$ and we can ignore agents i that are dominated by earlier, cheaper agents j with $p_j < p_i$ and $\omega_j < \omega_i$.

Towards a dynamic program. We define the point q_i as the leftmost point on the line where agent i can pick up the package without causing a delay, i.e., we have $q_i := \frac{p_i + s - \delta}{2}$ since $p_i - q_i = \delta + (q_i - s)$. Note that $q_1 = s$ and $q_j < q_i$ for $j < i$. Similarly as – but more specific than – in the characterization of uniform instances on general graphs (Lemma 5) we get a limited set of possible pick-up locations:

▶ **Lemma 9.** *There is an optimum solution where each agent i that is involved in advancing the package picks it up at $q_i^+ = q_i$ or at $q_i^+ = p_i$.*

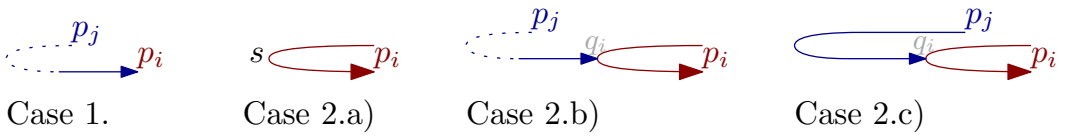


Figure 5 Case distinction in the dynamic program for FastEfficientDelivery on the line. Either agent i is not involved at all, does all on its own or is subsequent to some agent j, where we distinguish between $p_j \leq q_i$ and $p_j > q_i$.

Dynamic program. Lemma 9 suggests that in an inductive approach from left to right it suffices to consider only finitely many handover options. We define the following subproblems:

$$\begin{aligned} S[i] &= \text{An energy-optimal schedule to deliver the package to } p_i \\ &\quad \text{in time } (\delta + (p_i - s))/\overline{v}, \text{ using only the agents } \{1, 2, \ldots, i\}. \\ \mathcal{E}[i] &= \text{Energy consumption of } S[i]. \\ A[i] &= \text{Index of the last package-carrying agent in } S[i]. \\ A'[i] &= \text{Index of the second to last carrying agent in } S[i] \text{ (if any)}. \end{aligned}$$

We will argue how to compute the optimum energy costs $\mathcal{E}[i]$ (and with it $A[i]$ and $A'[i]$) without explicitly maintaining the schedules $S[i]$ ($S[i]$ can later be retraced from $A[i]$ and $A'[i]$). For computing $\mathcal{E}[i]$, $A[i]$ and $A'[i]$, we distinguish four cases (also shown in Figure 5):

1. Agent i is not involved in $S[i]$.
2. Agent i is involved in $S[i]$. Hence by Lemma 9, agent i has pick-up location $q_i^+ = q_i$; and we get the following three variations:
   a. $i = 1$ and agent 1 picks up the package at s itself.
   b. Agent i picks up the package from some other agent j with $p_j \leq q_i$.
   c. Agent i picks up the package from some other agent j with $p_j > q_i$.

In cases 1, 2b) and 2c), we can determine $\mathcal{E}[i]$ in constant time using a single prior entry of the dynamic programming table:

Case 1. If i is not involved in $S[i]$, the best choice for the agent who transports the package to p_i is agent $i-1$, as it is the cheapest one on the last segment $[p_{i-1}, p_i]$ and we have $\mathcal{E}[i-1] \leq \mathcal{E}[j] + (p_{i-1} - p_j) \cdot \omega_j$ for all $j < i-1$ by induction. Hence we can optimize in constant time:

$$\mathcal{E}[i] = \min_{j<i}\{\mathcal{E}[j] + (p_i - p_j) \cdot \omega_j\} = \mathcal{E}[i-1] + (p_i - p_{i-1}) \cdot \omega_{i-1},$$
$$A[i] = i - 1 \text{ and } A'[i] = A[i-1].$$

Case 2.a) This is the base case where the first agent is on its own:

$$\mathcal{E}[1] = 2 \cdot |p_1 - s| \cdot \omega_1 = 2 \cdot \delta \cdot \omega_1, \quad A[1] = \text{none}, \ A'[1] = \text{none}.$$

Case 2.b) If agent i is involved in $S[i]$ and takes over at q_i from an agent j with $p_j \leq q_i$, we want j to minimize $\mathcal{E}[j] + (q_i - p_j) \cdot \omega_j$, the cost of bringing the package to q_i. Now let $i' = \max\{j \mid p_j \leq q_i\}$ be the agent starting closest to the left of q_i. As in Case 1, we argue that i' is the optimum choice for j, as it minimizes the cost on $[p_{i'}, q_i]$ and does not constrain the schedule up to $p_{i'}$ further. Hence, we again get in amortized constant time, i.e., we update i' by incrementing it lazily when going from i to $i+1$:

$$\mathcal{E}[i] = \mathcal{E}[i'] + (q_i - p_{i'}) \cdot \omega_{i'} + 2 \cdot (p_i - q_i) \cdot \omega_i, \quad A[i] = i, \ A'[i] = i'.$$

The most interesting case is the remaining case (2.*c*), where the agent j handing over to i starts in between q_i and p_i. Where can we look up the energy consumption c of an optimum schedule that ends with j bringing the package q_i – the dynamic program being only defined for points p_j? For some j, we might have $A[j] = j$, so $S[j]$ ends by j walking to q_j and back. In that case, we can exploit $q_j < q_i$ and use $\mathcal{E}[j] - (p_j - q_i) \cdot \omega_j$ as a candidate for the energy consumption c. But what if $A[j] \neq j$? As we saw, this implies $A[j] = j - 1$, but in that case we cannot just subtract $(p_j - q_i)\omega_j$: We do not know how $S[j]$ looks like between q_i and p_j. We argue that we do not need to consider these agents j as candidates at all!

▶ **Lemma 10.** *If in some optimal schedule $S[i]$ the agent j preceding i is of type 2.c), then in the schedule $S[j]$ we have $A[j] = j$.*

Proof. Under the assumption of Case 2.c), the cost of agent i is fixed to $2 \cdot (p_i - q_i) \cdot \omega_i$. Agents 1 to $i - 1$ will collaborate in the most efficient way to bring the package up to q_i. By definition of j, j is the last agent bringing the package to q_i. From the decreasing weight property, we know that none of the agents $j + 1$ to $i - 1$ were involved in $S[i]$. So if we take the partial schedule of $S[i]$ up to q_i and extend it by letting j bring the package to p_j, we obtain a feasible candidate schedule S' for $S[j]$ as none of the agents $j + 1$ to i are involved. We now argue that S' is an optimum schedule for $S[j]$. The segment $[q_i, p_j]$ is covered with the minimum possible energy, as j is the *unique* most efficient agent available for $S[j]$. The segment $[s, q_i]$ is also covered cheapest possible as its part of $S[i]$ was optimized over all agents 1 to i, so a superset of the agents available for $S[j]$. Moreover, the uniqueness implies that *all* optimum schedules for $S[j]$ need to end with agent j on $[q_i, p_j]$, hence $A[j] = j$. ◀

Case 2.c) Lemma 10 leaves us with only those agents j whose schedules $S[j]$ we understand sufficiently to modify them into candidates for $S[i]$ under Case 2.c):

$$\mathcal{E}[i] = \min_j \{\mathcal{E}[j] - (p_j - q_i)\omega_j \mid q_i < p_j < p_i \wedge A[j] = j\} + 2(p_i - q_i)\omega_i,$$
$$A[i] = i, \quad A'[i] = \arg\min_j \{\mathcal{E}[j] - (p_j - q_i)\omega_j \mid q_i < p_j < p_i \wedge A[j] = j\}.$$

We can now take $\mathcal{E}[i]$ as the minimum over the four cases 1–2.c) and compute all schedules $S[i]$ by proceeding over all subproblems in increasing order, giving us the energy-optimal schedules for delivering the package to the points p_i in time $(\delta + (p_i - s))/\overline{v}$. How can we use the solutions to the subproblems $\mathcal{E}[i]$ to find the energy $\mathcal{E}$ of an energy-optimal schedule delivering the package to the target t in optimum time $(\delta + (t - s))/\overline{v}$? Let k' be the closest agent on the left of t, i.e., $k' := \arg\max_i p_i \leq t$. Clearly, if in an optimum schedule the package is delivered to t by an agent starting to the left of t, then by the decreasing weight property this agent must be agent k', giving us $\mathcal{E} = \mathcal{E}[k'] + (t - p_{k'}) \cdot \omega_k$.

Delivery to t and agents with $p_i > t$. It remains to take care of agents with starting positions $p_i > t$: As illustrated in Figure 6, multiple agents with $p_i > t$ might be involved in the most efficient delivery. Note that our dynamic programming problem $\mathcal{E}[i]$ is defined independent of t and so we can also easily compute $\mathcal{E}[i]$ for $p_i > t$. Agents i with $q_i > t$ are not useful, however, for a delivery to t, as they arrive in $[s, t]$ only after the package has been delivered. Similar to Lemma 10, we claim that among the remaining agents i only those with $A[i] = i$ need to be considered:

▶ **Lemma 11.** *If an agent i with $p_i > t$ is the last agent in any optimal schedule S from s to t, then $A[i] = i$.*

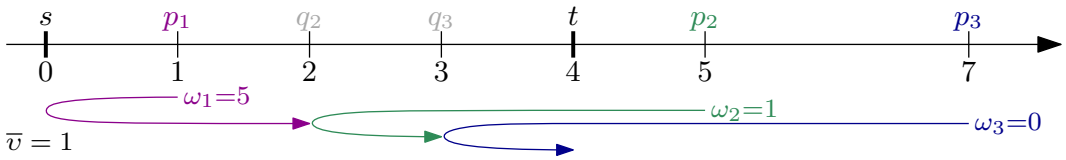


Figure 6 An example, where the only optimum schedule uses both agents on the right of t. The optimum schedule has delivery time $\mathcal{T} = (p_1 + t - s)/\overline{v} = 5/1 = 5$ and energy consumption $\mathcal{E} = p_1 \cdot \omega_1 + (p_2 - q_2 + q_3 - q_2) \cdot \omega_2 + (p_3 - q_3 + t - q_3) \cdot \omega_3 = 3 \cdot 5 + 4 \cdot 1 + 5 \cdot 0 = 19$.

Proof. We have $q_i < t < p_i$. By the decreasing weight property, no agent $j > i$ will be used in S. We extend S to a schedule S' by letting agent i walk from t to p_i. Then S' is a candidate for $S[i]$. Similar to Lemma 10, S' consists of an optimal solution for $[s, t]$ and the strictly cheapest agent on $[t, p_i]$ and hence S' is optimal for $S[i]$ and all optimum schedules for $S[i]$ have $A[i] = i$. The optimum s-t-delivery is thus given by:

$$\mathcal{E} = \min_j \{\mathcal{E}[j] - (p_j - t)\omega_j \mid (q_j < t < p_j \text{ and } A[j] = j) \text{ or } j = k'\},$$

which takes linear time once at the very end. ◀

Details of the dynamic program. The computational bottleneck of our dynamic program is (for each subproblem $\mathcal{E}[i]$) the minimization over the set of options in Case 2.c). Each option evaluates a linear function $f_j(q_i) := \omega_j \cdot q_i + (\mathcal{E}[j] - p_j \cdot \omega_j)$ at position q_i, which can be seen as a lower envelope intersection query. Similarly to before, we use point-line duality and a dynamic convex hull data structure to avoid considering all agents explicitly as predecessors and instead quickly search the best one.

▶ **Lemma 12.** *An optimum schedule for* FastEfficientDelivery *with uniform velocity* $\overline{v}$ *on the line can be computed in* $\mathcal{O}(n + k \log k)$ *time.*

Combining this with Lemma 8 gives the full solution on paths. Note that strictly speaking, in the uniform velocity instances, the package is not available at s at time zero, but is brought there by agents of preceding instances at exactly the time when the first agent can reach it.

▶ **Theorem 13.** *An optimum solution for* FastEfficientDelivery *on paths can be computed in* $\mathcal{O}(n + k \log k)$ *time.*

5 Optimizing convex combinations of objectives

In this section, we look at a convex combination of the two objectives: minimizing both the delivery time $\mathcal{T}$ and the energy consumption $\mathcal{E}$ by minimizing the term $\epsilon \cdot \mathcal{T} + (1 - \epsilon) \cdot \mathcal{E}$, for a given value ϵ, $0 < \epsilon < 1$. We call the problem of minimizing this combined objective CombinedDelivery. As an application of the NP-hardness proof for FastEfficient-Delivery, we get NP-hardness of CombinedDelivery as well: The main idea is to counter small values of ϵ by scaling the weights of the agents by a small factor $\delta(\epsilon)$, thus decreasing the importance of $\mathcal{E}$ alongside $\mathcal{T}$ as well.

▶ **Theorem 14.** CombinedDelivery *is* NP*-hard for all* $\epsilon \in (0, 1)$*, even on planar graphs.*

A 2-approximation for CombinedDelivery using a single agent

Recall that for FastDelivery, the agents involved in an optimum delivery were characterized by increasing velocities v_i, while for FastEfficientDelivery on path graphs, the agents of an optimum solution were characterized by decreasing tuples (v_i^{-1}, ω_i).

Although it is *not* possible to characterize the *order* of the agents in an optimum CombinedDelivery schedule by their velocities and weights alone, we can at least characterize the position of a *minimal* agent, leading to a 2-approximation using a single agent:

▶ **Lemma 15.** *Let without loss of generality $1, 2, \ldots, i$ denote the indices of all involved agents appearing in that order in an optimum* CombinedDelivery *schedule. Then the last agent i is* minimal *in the following sense: $i \in \arg\min_j \left\{\epsilon \cdot v_j^{-1} + (1-\epsilon) \cdot \omega_j\right\}$.*

Proof. Recall that we denote by d_j the total distance travelled by agent j and by d_j^* the distance travelled by agent j while carrying the package. Thus agent j contributes at least $\epsilon \cdot d_j^* \cdot v_j^{-1} + (1-\epsilon) \cdot (d_j \cdot \omega_j) \geq d_j^* \cdot \left(\epsilon v_j^{-1} + (1-\epsilon)\omega_j\right)$ towards $\epsilon\mathcal{T} + (1-\epsilon)\mathcal{E}$. Assume for the sake of contradiction that the minimum value $\epsilon v_j^{-1} + (1-\epsilon)\omega_j$ is not obtained by agent i but by an agent $m < i$. Then we can replace the agents $m+1, \ldots, i$ by agent m, resulting in a decrease in the objective function of at least

$$\begin{aligned}
&\sum_{j=m}^{i} \epsilon d_j^* v_j^{-1} + (1-\epsilon) d_j \omega_j - \sum_{j=m}^{i} d_j^* \left(\epsilon v_m^{-1} + (1-\epsilon)\omega_m\right) \\
&\geq \sum_{j=m}^{i} d_j^* \left(\epsilon v_j^{-1} + (1-\epsilon)\omega_j\right) - \sum_{j=m}^{i} d_j^* \left(\epsilon v_m^{-1} + (1-\epsilon)\omega_m\right) \\
&\geq d_i^* \left(\epsilon v_i^{-1} + (1-\epsilon)\omega_i\right) - d_i^* \left(\epsilon v_m^{-1} + (1-\epsilon)\omega_m\right) > 0,
\end{aligned}$$

contradicting the minimality of the optimum CombinedDelivery schedule. ◀

▶ **Theorem 16.** *There is a 2-approximation for* CombinedDelivery *which uses only a single agent (and thus can be found in polynomial time).*

Proof. Note that agent i contributes at most $\epsilon d_i v_i^{-1} + (1-\epsilon) d_i \omega_i$ towards $\epsilon\mathcal{T} + (1-\epsilon)\mathcal{E}$. Starting from an optimum CombinedDelivery schedule we can replace all agents $1, \ldots, i-1$ along their trajectories by the minimal agent i. This prolongs the travel distance of agent i by $2 \cdot \sum_{j=1}^{i-1} d_j^*$. Overall, we increase the objective function by at most

$$\begin{aligned}
&2\sum_{j=1}^{i-1} d_j^* \left(\epsilon v_i^{-1} + (1-\epsilon)\omega_i\right) - \sum_{j=1}^{i-1} d_j^* \left(\epsilon v_j^{-1} + (1-\epsilon)\omega_j\right) \leq \sum_{j=1}^{i-1} d_j^* \left(\epsilon v_i^{-1} + (1-\epsilon)\omega_i\right) \\
&\leq \sum_{j=1}^{i-1} d_j^* \left(\epsilon v_j^{-1} + (1-\epsilon)\omega_j\right) < \sum_{j=1}^{i} d_j^* \left(\epsilon v_j^{-1} + (1-\epsilon)\omega_j\right) \leq \epsilon\mathcal{T} + (1-\epsilon)\mathcal{E}.
\end{aligned}$$

Hence only using agent i to deliver the package is a 2-approximation for CombinedDelivery. We get a polynomial-time approximation algorithm with approximation ratio 2 by choosing among all k agents the one with minimum value $\left(\epsilon v_j^{-1} + (1-\epsilon)\omega_j\right) \cdot (d(p_j, s) + d(s, t))$. ◀

6 Discussion

Our techniques and results extend to a variety of delivery problems and model generalizations. A key ingredient here is that the order of our dynamic programming subproblems depends only on the parameters the agent has *while carrying the package*. Hence it is possible to, e.g.,

incorporate 2-speed agent models (modeling different speeds [16] with/without carrying the package) or topographical features (modeling edge traversals in uphill/downhill direction).

Furthermore, the 2-approximation given for CombinedDelivery is applicable to FastDelivery as well, and a relaxation of FastEfficientDelivery, in which one allows the optimum delivery time to be achieved with a constant-factor approximation of the energy consumption, stays NP-hard. It is unclear whether FastEfficientDelivery can be *solved efficiently on trees* or whether CombinedDelivery *allows a PTAS*. We consider these two problems the major open questions raised by this work.

References

1 Julian Anaya, Jérémie Chalopin, Jurek Czyzowicz, Arnaud Labourel, Andrzej Pelc, and Yann Vaxès. Convergecast and Broadcast by Power-Aware Mobile Agents. *Algorithmica*, 74(1):117–155, 2016. See also DISC'12. doi:10.1007/s00453-014-9939-8.

2 David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, NJ, USA, 2007. ISBN: 978-0-691-12993-8.

3 Evangelos Bampas, Jurek Czyzowicz, Leszek Gasieniec, David Ilcinkas, Ralf Klasing, Tomasz Kociumaka, and Dominik Pająk. Linear Search by a Pair of Distinct-Speed Robots. In *23rd International Colloquium on Structural Information and Communication Complexity SIROCCO'16*, pages 195–211, 2016. doi:10.1007/978-3-319-48314-6_13.

4 Gerth Stølting Brodal and Riko Jacob. Dynamic Planar Convex Hull. In *43rd Symposium on Foundations of Computer Science FOCS'02*, pages 617–626, 2002. doi:10.1109/SFCS.2002.1181985.

5 Andreas Bärtschi. *Efficient Delivery with Mobile Agents*. PhD thesis, ETH Zürich, 2017. doi:10.3929/ethz-b-000232464.

6 Andreas Bärtschi, Jérémie Chalopin, Shantanu Das, Yann Disser, Barbara Geissmann, Daniel Graf, Arnaud Labourel, and Matúš Mihalák. Collaborative Delivery with Energy-Constrained Mobile Robots. In *23rd International Colloquium on Structural Information and Communication Complexity SIROCCO'16*, pages 258–274, 2016. doi:10.1007/978-3-319-48314-6_17.

7 Andreas Bärtschi, Jérémie Chalopin, Shantanu Das, Yann Disser, Barbara Geissmann, Daniel Graf, Arnaud Labourel, and Matúš Mihalák. Collaborative delivery with energy-constrained mobile robots. *Theoretical Computer Science*, 2017. To appear. See also SIROCCO'16. doi:10.1016/j.tcs.2017.04.018.

8 Andreas Bärtschi, Jérémie Chalopin, Shantanu Das, Yann Disser, Daniel Graf, Jan Hackfeld, and Paolo Penna. Energy-efficient Delivery by Heterogeneous Mobile Agents. In *34th International Symposium on Theoretical Aspects of Computer Science STACS'17*, pages 10:1–10:14, 2017. doi:10.4230/LIPIcs.STACS.2017.10.

9 Andreas Bärtschi, Daniel Graf, and Paolo Penna. Truthful Mechanisms for Delivery with Agents. In *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems ATMOS'17*, pages 2:1–2:17, 2017. doi:10.4230/OASIcs.ATMOS.2017.2.

10 Andreas Bärtschi and Thomas Tschager. Energy-Efficient Fast Delivery by Mobile Agents. In *21st International Symposium on Fundamentals of Computation Theory FCT'2017*, pages 82–95, 2017. doi:10.1007/978-3-662-55751-8_8.

11 Jérémie Chalopin, Shantanu Das, Matús Mihalák, Paolo Penna, and Peter Widmayer. Data Delivery by Energy-Constrained Mobile Agents. In *9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics ALGOSENSORS'13*, pages 111–122, 2013. doi:10.1007/978-3-642-45346-5_9.

12 Jérémie Chalopin, Riko Jacob, Matús Mihalák, and Peter Widmayer. Data Delivery by Energy-Constrained Mobile Agents on a Line. In *41st International Colloquium on Automata, Languages, and Programming ICALP'14*, pages 423–434, 2014. doi:10.1007/978-3-662-43951-7_36.

13 Jurek Czyzowicz, Krzysztof Diks, Jean Moussi, and Wojciech Rytter. Communication Problems for Mobile Agents Exchanging Energy. In *23rd International Colloquium on Structural Information and Communication Complexity SIROCCO'16*, 2016. doi:10.1007/978-3-319-48314-6_18.

14 Jurek Czyzowicz, Leszek Gasieniec, Konstantinos Georgiou, Evangelos Kranakis, and Fraser MacQuarrie. The Beachcombers' Problem: Walking and searching with mobile robots. *Theoretical Computer Science*, 608:201–218, 2015. doi:10.1016/j.tcs.2015.09.011.

15 Jurek Czyzowicz, Leszek Gasieniec, Adrian Kosowski, and Evangelos Kranakis. Boundary Patrolling by Mobile Agents with Distinct Maximal Speeds. In *19th European Symposium on Algorithms ESA'11*, pages 701–712, 2011. doi:10.1007/978-3-642-23719-5_59.

16 Jurek Czyzowicz, Konstantinos Georgiou, Evangelos Kranakis, Fraser MacQuarrie, and Dominik Pająk. Fence patrolling with two-speed robots. In *5th International Conference on Operations Research and Enterprise Systems ICORES'16*, pages 229–241, 2016. doi:10.5220/0005687102290241.

17 Erik D. Demaine, Mohammadtaghi Hajiaghayi, Hamid Mahini, Amin S. Sayedi-Roshkhar, Shayan Oveisgharan, and Morteza Zadimoghaddam. Minimizing movement. *ACM Transactions on Algorithms*, 5(3):1–30, 2009. See also SODA'07. doi:10.1145/1541885.1541891.

18 Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer, Heidelberg, Germany, 1987. doi:10.1007/978-3-642-61568-9.

19 Jack Edmonds and Ellis L. Johnson. Matching, euler tours and the chinese postman. *Mathematical Programming*, 5(1):88–124, 1973. doi:10.1007/BF01580113.

20 Greg N. Frederickson, Matthew S. Hecht, and Chul E. Kim. Approximation Algorithms for Some Routing Problems. *SIAM Journal on Computing*, 7(2):178–193, 1978. See also FOCS'76. doi:10.1137/0207017.

21 Woburn Collegiate Institute's Programming Enrichment Group. Convex hull trick. PEG-Wiki, September 2016. URL: http://wcipeg.com/wiki/Convex_hull_trick.

22 Riko Jacob. *Dynamic planar convex hull*. PhD thesis, Department of Computer Science, University of Aarhus, Denmark, 2002. BRICS Dissertation Series DS-02-3. URL: http://www.brics.dk/DS/Ref/BRICS-DS-Ref/BRICS-DS-Ref.html#BRICS-DS-02-3.

23 David Lichtenstein. Planar Formulae and Their Uses. *SIAM Journal on Computing*, 11(2):329–343, 1982. doi:10.1137/0211025.

24 Swiss Post. Swiss Post delivery robots in use by Jelmoli. Press release, August 2017. URL: https://www.post.ch/en/about-us/company/media/press-releases/2017/swiss-post-delivery-robots-in-use-by-jelmoli.

25 Paolo Toth and Daniele Vigo, editors. *The Vehicle Routing Problem*. SIAM Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. ISBN: 0-89871-498-2.

26 Elizabeth Weise. Amazon delivered its first customer package by drone. USA Today, December 2016. URL: http://usat.ly/2hNgf0y.

Parity to Safety in Polynomial Time for Pushdown and Collapsible Pushdown Systems

Matthew Hague[1]
Royal Holloway, University of London, UK
matthew.hague@rhul.ac.uk
https://orcid.org/0000-0003-4913-3800

Roland Meyer
TU Braunschweig, Germany
roland.meyer@tu-braunschweig.de

Sebastian Muskalla
TU Braunschweig, Germany
s.muskalla@tu-braunschweig.de
https://orcid.org/0000-0001-9195-7323

Martin Zimmermann[2]
Universität des Saarlandes, Saarbrücken, Germany
zimmermann@react.uni-saarland.de

Abstract

We give a direct polynomial-time reduction from parity games played over the configuration graphs of collapsible pushdown systems to safety games played over the same class of graphs. That a polynomial-time reduction would exist was known since both problems are complete for the same complexity class. Coming up with a direct reduction, however, has been an open problem. Our solution to the puzzle brings together a number of techniques for pushdown games and adds three new ones. This work contributes to a recent trend of liveness to safety reductions which allow the advanced state-of-the-art in safety checking to be used for more expressive specifications.

2012 ACM Subject Classification Theory of computation → Logic and verification, Theory of computation → Modal and temporal logics, Theory of computation → Verification by model checking, Theory of computation → Grammars and context-free languages

Keywords and phrases Parity Games, Safety Games, Pushdown Systems, Collapsible Pushdown Systems, Higher-Order Recursion Schemes, Model Checking

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.57

Related Version The full version is available as technical report [16], `https://arxiv.org/abs/1805.02963`.

Acknowledgements We thank the anonymous reviewers for their comments.

[1] Supported by the EPSRC under grant [EP/K009907/1].
[2] Supported by the DFG under grant [ZI 1516/1-1].

1 Introduction

Model-checking games ask whether there is a strategy (or implementation) of a system that can satisfy required properties against an adversary (or environment). They give a natural method for reasoning about systems wrt. popular specification logics such as LTL, CTL, and the μ-calculus. The simplest specifications are reachability or safety properties, where the system either needs to reach a given good state or avoid a bad state (such as a null-pointer dereference). The most expressive logic typically studied is the μ-calculus, which subsumes LTL, CTL, and CTL* [22]. One can reduce μ-calculus model checking in polynomial time to the analysis of parity games (op cit.) via a quite natural product of system and formula.

In the finite-state setting, while reachability and safety games can be solved in linear time and space, the best known algorithms for parity games are quasi-polynomial time [8] or quasi-linear space [18, 13]. For infinite-state games described by pushdown systems, or more generally, collapsible pushdown systems, the complexities match: EXPTIME-complete for solving reachability, safety [5, 32], and parity games [32] over pushdown systems, and n-EXPTIME-complete for order-n collapsible pushdown systems [11, 7, 24, 17].

Pushdown systems are an operational model for programs with (recursive) function calls. In such systems, a configuration has a control state from a finite set and a stack of characters from a finite alphabet (modeling the call stack). Collapsible pushdown systems [17] are an operational model for higher-order recursion as found in most languages (incl. Haskell, JavaScript, Python, C++, Java, ...). They have a nested stack-of-stacks (e.g. an order-2 stack is a stack of stacks) and *collapse links* which provide access to calling contexts.

Given that safety and parity games over collapsible pushdown systems are complete for the same complexity classes, the problems must be inter-reducible in polynomial-time. However, a direct (without a detour via Turing machines) polynomial-time reduction from parity to safety has been an open problem [14]. To see why the reduction is difficult to find, note that a safety game is lost based on a finite prefix of a play while determining the winner of a parity game requires access to the infinitely many elements of a play. Complexity theory tells us that this gap can be bridged by access to the stack, with only polynomial overhead.

Our contribution is such a polynomial-time reduction from parity to safety. From a theoretical standpoint, it explains the matching complexities despite the difference in expressible properties. From a practical standpoint, it may help building model-checking tools for μ-calculus specifications. Indeed, competitive and highly optimized tools exist for analysing reachability and safety properties of higher-order recursion schemes (HorSat [6, 31, 20] and Preface [28] being the current state-of-the-art), but implementing efficient tools for parity games remains a problem [15, 23]. Having the reduction at hand can allow the use of safety tools for checking parity conditions, suggest the transfer of techniques and optimizations from safety to parity, and inspire new algorithms for parity games. Still, a complexity-theoretic result should only be considered a first step towards practical developments.

Reductions from parity to safety have been explored for the finite-state case by Bernet et al. [1], and for pushdown systems by Fridman and Zimmermann [14]. We will refer to them as counter reductions, as they use counters to track the occurrences of odd ranks. These existing reductions are not polynomial. Berwanger and Doyen [2] showed that counter reductions can be made polynomial in the case of finite-state imperfect-information games.

Our solution to the puzzle brings together a number of techniques for pushdown games and contributes three new ones. We first show how to lift the existing counter reductions [1, 14] from first order to higher orders. For this we exploit a rank-awareness property of collapsible pushdown systems [17]. Secondly, we prove the correctness of this lifting by showing that it

commutes with a reduction from order-n to order-$(n-1)$ games [32, 17]. The polynomial-time reduction is then a compact encoding of the lifted counter reduction. It uses the ability of higher-order stacks to encode large numbers [7] and the insight that rank counters have a stack-like behavior, even in their compact encoding.

Much recent work verifies liveness properties via reductions to safety [25, 10, 26, 27, 12] or reachability [21, 4] with promising results. For finite-state generalized parity games, Sohail and Somenzi show that pre-processing via a safety property can reduce the state space that a full parity algorithm needs to explore, giving competitive synthesis results for LTL [30]. In the case of infinite-state systems (including pushdowns), reductions from liveness (but not parity games) have been explored by Biere et al. [3] and Schuppan and Biere [29].

2 Preliminaries

We define games over collapsible pushdown systems (CPDS). For a full introduction see [17]. CPDS are an operational model of functional programs that is equivalent to higher-order recursion schemes (HORS) [17]. Without collapse, they correspond to *safe* HORS [19].

In the following, let $\mathbb{N}$ be the set of natural numbers (including 0) and $[i, j]$ denote the set $\{i, i+1, \ldots, j\}$.

2.1 Higher-Order Collapsible Stacks

Higher-order stacks are a nested stack-of-stacks structure whose stack characters are annotated by collapse links that point to a position in the stack. Intuitively, this position is the context in which the character was created. We describe the purpose of collapse links after some basic definitions.

▶ **Definition 2.1** (Order-n Collapsible Stacks)**.** For $n \geq 1$, let Σ be a finite set of stack characters Σ together with a partition function[3] $\lambda : \Sigma \to [1, n]$. An *order-0 stack* with up-to order-n collapse links is an annotated character $a^i \in \Sigma \times \mathbb{N}$. An *order-$k$ stack* with up-to order-n collapse links is a non-empty sequence $w = [w_1 \ldots w_\ell]_k$ (with $\ell > 0$) such that each w_i is an order-$(k-1)$ stack with up-to order-n collapse links. By $Stacks_n$ we denote the set of order-n stacks with up-to order-n links.

In the sequel, we will refer to stacks in $Stacks_n$ as order-n stacks. By *order-k* stack we will mean an order-k stack with up-to order-n links, where n will be clear from the context.

Given an order-k stack with up-to order-n links $w = [w_1 \ldots w_\ell]_k$, we define below the operation $top_{k'}$ to return the topmost element of the topmost order-k' stack. Note that this element is of order-$(k'-1)$. The top of a stack appears leftmost. The operation bot^i_k removes all but the last i elements from the topmost order-k stack. It does not change the order of the stack and requires $i \in [1, \ell]$.

$$\begin{array}{rclcrcl} top_k(w) & = & w_1 & & bot^i_k(w) & = & [w_{\ell-i+1} \ldots w_\ell]_k \\ top_{k'}(w) & = & top_{k'}(w_1)\ (k' < k) & & bot^i_{k'}(w) & = & [bot^i_{k'}(w_1) w_2 \ldots w_\ell]_k\ (k' < k). \end{array}$$

For technical convenience, we will also define

$$top_{n+1}(w) = w$$

which, we note, does not extend to top_{n+2} or beyond.

[3] Readers familiar with CPDS may expect links to be pairs (k, i) and the alphabet Σ not to be partitioned by link order. The partition assumption is oft-used. It is always possible to tag each character with its link order using $\Sigma \times [1, n]$. Such a partition becomes crucial in Section 4.

The destination of a collapse link i on a with $\lambda(a) = k$ in a stack w is $bot^i_k(w)$, when defined. When $i = 0$, the link is considered *null*. We often omit irrelevant collapse links from characters to improve readability.

When u is a $(k-1)$-stack and $v = [v_1 \dots v_\ell]_n$ is an n-stack with $k \in [1, n]$, we define $u :_k v$ as the stack obtained by adding u on top of the topmost k-stack of v. Formally,

$$u :_k v = [uv_1 \dots v_\ell]_n \ (k = n) \qquad \text{and} \qquad u :_k v = [(u :_k v_1)v_2 \dots v_\ell]_n \ (k < n).$$

▶ **Example 2.2.** When $\lambda(a) = 3$ and $\lambda(b) = 2$ let $w = [[[a^1b^1]_1[b^1]_1]_2[[b^0]_1]_2]_3$ be an order-3 collapsible stack. The destination of the topmost link is $bot^1_3(w) = [[[b^0]_1]_2]_3$. Furthermore, $bot^1_2(w) = [[[b^1]_1]_2[[b^0]_1]_2]_3$ and $top_2(w) = [a^1b^1]_1$. Here, $top_2(w) :_2 bot^1_2(w) = w$.

Operations on Order-n Collapsible Stacks

CPDS are programs with a finite control acting on collapsible stacks via the operations:

$$\mathcal{O}_n = \{push_2, \dots, push_n\} \cup \{push_a, rew_a \mid a \in \Sigma\} \cup \{pop_1, \dots, pop_n\} \cup \{collapse\}.$$

Operations $push_k$ of order $k > 1$ copy the topmost element of the topmost order-k stack. Order-1 push operations $push_a$ push a onto the topmost order-1 stack and annotate it with an order-$\lambda(a)$ collapse link. When executed on a stack w, the link destination is $pop_{\lambda(a)}(w)$. A pop_k removes the topmost element from the topmost order-k stack. The rewrite rew_a modifies the topmost stack character while maintaining the link (rewrite must respect the link order). Collapse, when executed on a^i with $\lambda(a) = k$, pops the topmost order-k stack down to the last i elements, captured by bot^i_k. Formally, for an order-n stack w:

1. $push_k(w) = top_k(w) :_k w$.
2. $push_a(w) = a^{\ell-1} :_1 w$ when $top_{k+1}(w) = [w_1 \dots w_\ell]_k$, where $k = \lambda(a)$ is the link order,
3. $pop_k(w) = v$ when $w = u :_k v$,
4. $collapse(w) = bot^i_k(w)$ when $top_1(w) = a^i$ and $\lambda(a) = k$, and
5. $rew_b(w) = b^i :_1 v$ when $w = a^i :_1 v$ and $\lambda(a) = \lambda(b)$.

Note that since our definition of stacks does not permit empty stacks, pop_k is undefined if v is empty and *collapse* is undefined when $i = 0$. Thus, the empty stack cannot be reached using CPDS operations; instead, the offending operation will simply be unavailable. Likewise if a rewrite operation would change the order of the link.

▶ **Example 2.3.** Recall Example 2.2 and that $w = [[[a^1b^1]_1[b^1]_1]_2[[b^0]_1]_2]_3$. Given the order-3 link 1 of the topmost stack character a, a collapse operation yields $u = [[[b^0]_1]_2]_3$. Now $push_3(u) = [[[b^0]_1]_2[[b^0]_1]_2]_3$. A $push_a$ on this stack results in $v = [[[a^1b^0]_1]_2[[b^0]_1]_2]_3$. We have $pop_3(v) = u = collapse(v)$.

There is a subtlety in the interplay of collapse links and higher-order pushes. For a $push_k$, links pointing outside of $u = top_k(w)$ have the same destination in both copies of u, while links pointing within u point to different sub-stacks.

▶ Remark (*nop*). For convenience we use an operation *nop* which has no effect on the stack. We can simulate it by rew_a where a is the topmost character (by the format of rules, below, we will always know the topmost character when applying an operation). Hence, it is not a proof case.

2.2 Collapsible Pushdown Systems and Games

▶ **Definition 2.4** (CPDS). An order-n *collapsible pushdown system* is a tuple $\mathcal{C}$ given by $(\mathcal{P}, \Sigma, \lambda, \mathcal{R}, p_I, a_I, \rho)$ with $\mathcal{P}$ a finite set of control states with initial control state p_I, Σ a finite stack alphabet with initial stack character a_I and order function λ, $\rho : \mathcal{P} \to \mathbb{N}$ a function assigning ranks to $\mathcal{P}$, and $\mathcal{R} \subseteq (\mathcal{P} \times \Sigma \times \mathcal{O}_n \times \mathcal{P})$ a set of rules. The size is $|\mathcal{C}| = |\mathcal{P}| + |\Sigma|$. The remaining entries polynomially depend on $\mathcal{P}$ and Σ (note that n is fixed).

We attach ranks to CPDS instead of games as we later need the notion of *rank-aware* CPDS.

A *configuration* of a CPDS is a pair $c = \langle p, w\rangle$ with $p \in \mathcal{P}$ and a stack $w \in \mathit{Stacks}_n$. We have a transition $\langle p, w\rangle \longrightarrow \langle p', w'\rangle$ if there is a rule $(p, a, o, p') \in \mathcal{R}$ with $\mathit{top}_1(w) = a$ and $w' = o(w)$. The initial configuration is $\langle p_I, w_I\rangle$ where $w_I = [\dots[a_I^0]_1 \dots]_n$. To begin from another configuration, one can adjust the CPDS rules to build the required stack from the initial configuration. A computation is a sequence of configurations $c_0, c_1, \dots$ where $c_0 = \langle p_I, w_I\rangle$ and $c_i \longrightarrow c_{i+1}$ for all $i \in \mathbb{N}$. Recall, transitions cannot empty a stack or rewrite the order of a link.

▶ **Definition 2.5** (Games over CPDS). A *game over a CPDS* is a tuple $\mathcal{G} = (\mathcal{C}, \mathbb{O}, \mathcal{W})$, where $\mathcal{C}$ is a CPDS, $\mathbb{O} : \mathcal{P} \to \{\mathrm{A}, \mathrm{E}\}$ is a division of the control states of $\mathcal{C}$ by owner Elvis (E) or Agnetha (A), and $\mathcal{W} \subseteq \mathbb{N}^\omega$ is a winning condition. The size of the game is $|\mathcal{G}| = |\mathcal{C}|$.

We call $\mathcal{G}$ a *safety game* if $\rho(p) \in \{1, 2\}$ for all $p \in \mathcal{P}$ and $\mathcal{W} = 2^\omega$. It is a *parity game* if $\mathcal{W}$ is the set of all sequences such that the smallest infinitely occurring rank is even.

We refer to computations as plays and require them to be infinite. This means every configuration $\langle p, w\rangle$ has some successor $\langle p', w'\rangle$. This does not lose generality as we can add to the CPDS transitions to a losing (as defined next) sink state (with self-loop) for $\mathbb{O}(p)$ from any configuration $\langle p, w\rangle$. A play $\langle p_0, w_0\rangle, \langle p_1, w_1\rangle, \langle p_2, w_2\rangle, \dots$ is won by Elvis, if its sequence of ranks satisfies the winning condition, i.e. $\rho(p_0)\rho(p_1)\rho(p_2)\dots \in \mathcal{W}$. Otherwise, Agnetha wins. When a play reaches (p, w), then the owner of p chooses the rule to apply. A *strategy* for player $\Upsilon \in \{\mathrm{E}, \mathrm{A}\}$ is a function $\sigma : (\mathcal{P} \times \mathit{Stacks}_n)^* \to \mathcal{R}$ that returns an appropriate rule based on the prefix of the play seen so far. A play $\langle p_0, w_0\rangle, \langle p_1, w_1\rangle, \langle p_2, w_2\rangle, \dots$ is according to σ if for all i with $\mathbb{O}(p_i) = \Upsilon$ we have $\langle p_i, w_i\rangle \longrightarrow \langle p_{i+1}, w_{i+1}\rangle$ via rule $\sigma(\langle p_0, w_0\rangle, \dots, \langle p_i, w_i\rangle)$. The strategy is winning if all plays according to σ are won by Υ. We say a player *wins* a game if they have a winning strategy from the initial configuration.

2.3 Rank-Aware Collapsible Pushdown Systems

We will often need to access the smallest rank that was seen in a play since some stack was created. *Rank-aware* CPDS record precisely this information [17]. We first define k-ancestors which, intuitively, give the position in the play where the top order-$(k-1)$ stack was pushed. Note, in the definition below, the integer j is unrelated to the collapse links.

▶ **Definition 2.6** (k-Ancestor). Let $k \in [2, n]$ (resp. $k = 1$). Given a play $c_0, c_1, \dots$ we attach an integer j to every order-$(k-1)$ stack as follows. In c_0 all order-$(k-1)$ stacks are annotated by 0. Suppose c_{i+1} was obtained from c_i using operation push_k (resp. push_a). Then the new topmost order-$(k-1)$ stack in c_{i+1} is annotated with i. If c_{i+1} is obtained via a $\mathit{push}_{k''}$ with $k'' > k$, then all annotations on the order-$(k-1)$ stacks in the copied stack are also copied.

The *k-ancestor* with $k \in [1, n]$ of c_i is the configuration c_j where j is the annotation of the topmost order-$(k-1)$ stack in c_i. Let $\mathit{top}_1(c_i) = a^\ell$ and $\lambda(a) = k'$. The *link ancestor* of c_i is the k'-ancestor of the 1-ancestor of c_i.

Applying a pop_k operation will expose (a copy of) the topmost $(k-1)$-stack of the k-ancestor. To understand the notion of a link ancestor, remember that *collapse* executed on a stack whose topmost order-0 stack is a^ℓ with $\lambda(a) = k'$ has the effect of executing $pop_{k'}$ several times. The newly exposed topmost $(k'-1)$-stack is the same that would be exposed if $pop_{k'}$ were applied at the moment the a character was pushed. This exposed stack is the same stack as is topmost on the k'-ancestor of the 1-ancestor of a. We illustrate this with an example.

▶ **Example 2.7.** Assume some c_0. Now take some c_1 containing the stack $w_1 = [[[b^0]_1]_2]_3$. Apply a $push_3$ operation to obtain c_2 with stack $w_2 = [[[b^0]_1]_2[[b^0]_1]_2]_3$. Note, the topmost $[[b^0]_1]_2$ has 3-ancestor c_1.

Now, let $\lambda(a) = 3$ and obtain c_3 with $push_a$, which thus contains the stack $w_3 = [[[a^1b^0]_1]_2[[b^0]_1]_2]_3$ where the a^1 has the 1-ancestor c_2.

We can now apply $push_3$ again to reach c_4 with stack $w_4 = [[[a^1b^0]_1]_2[[a^1b^0]_1]_2[[b^0]_1]_2]_3$. Note that both copies of a^1 have the 1-ancestor c_2. Moreover, the link ancestor of both is c_1. That is, the 3-ancestor of the topmost stack of c_2. In particular, applying *collapse* at c_4 results in a configuration with stack $[[[b^0]_1]_2]_3$, which is the same stack contained in c_1.

In the below, intuitively, the level-k rank is the smallest rank seen since the topmost $(k-1)$ stack was created. Similarly for the link-level rank. Our rank-awareness definition is from [17] but includes level-k ranks as well.

▶ **Definition 2.8** (Level Rank)**.** For a given play $c_0, c_1, \ldots$ the *level-k rank* with $k \in [1, n]$ (resp. *link-level rank*) at a configuration c_i is the smallest rank of a control state in the sequence $c_{j+1}, \ldots, c_i$ where j is the k-ancestor (link ancestor) of c_i.

In the following definition, ℓ is a special symbol to be read as *link*.

▶ **Definition 2.9** (Rank-Aware)**.** A *rank-aware CPDS* is a CPDS $\mathcal{C}$ over stack characters (a, Rk), where a is taken from a finite set and function Rk has type $\mathrm{Rk} : [1, n] \uplus \{\ell\} \to [0, m]$ (with m the highest rank of a state in $\mathcal{C}$). The requirement is that in all computations $c_0, c_1, \ldots$ of the CPDS all configurations $c_i = (p, w)$ with top-of-stack character (a, Rk) satisfy

$$\mathrm{Rk}(k) = \text{ the level-}k\text{ rank at } c_i,\ k \in [1, n], \quad \text{and} \quad \mathrm{Rk}(\ell) = \text{ the link-level rank at } c_i.$$

Below, we slightly generalize a lemma from [17] to include safety games and level-k ranks. Intuitively, we can obtain rank-awareness by keeping track of the required information in the stack characters and control states.

▶ **Lemma 2.10** (Rank-Aware)**.** *Given a parity (resp. safety) game over CPDS $\mathcal{C}$ of order-n, one can construct in polynomial time a rank-aware CPDS $\mathcal{C}'$ of the same order and a parity (resp. safety) game over $\mathcal{C}'$ such that Elvis wins the game over $\mathcal{C}$ iff he wins the game over $\mathcal{C}'$.*

Note, the number of functions Rk is exponential in n. However, since n is fixed the construction is polynomial. In the sequel, we will assume that all CPDS are rank-aware.

3 Main Result and Proof Outline

In the following sections, we define a reduction Poly which takes a parity game $\mathcal{G}$ over a CPDS and returns a safety game $\mathrm{Poly}(\mathcal{G})$ of the same order. The main result follows.

▶ **Theorem 3.1** (From Parity to Safety, Efficient)**.** *Given a parity game $\mathcal{G}$, Elvis wins $\mathcal{G}$ iff he wins* $\mathrm{Poly}(\mathcal{G})$*.* $\mathrm{Poly}(\mathcal{G})$ *is polynomially large and computable in time polynomial in the size of $\mathcal{G}$.*

We outline how to define Poly and prove it correct. First, we give a function $\mathrm{Counter}_B$ reducing an order-n parity game to an equivalent order-n safety game. It extends Fridman and Zimmermann's reduction [14] from first order to higher orders. In the finite-state setting, a related reduction appeared already in [1]. The idea is to count in the stack characters the occurrences of odd ranks. Elvis has to keep the counter values below B, a threshold that is a parameter of the reduction. For completeness, this threshold has to be n-fold exponential in the size of $\mathcal{G}$. Let $\mathrm{Exp}_0(f) = f$ and $\mathrm{Exp}_n(f) = 2^{\mathrm{Exp}_{n-1}(f)}$. We have the following lemma.

▶ **Lemma 3.2** (From Parity to Safety, Inefficient). *Given a parity game $\mathcal{G}$ played over an order-n CPDS, there is a bound $B(\mathcal{G}) = Exp_n(f(|\mathcal{G}|))$ for some polynomial f so that for all $B \geq B(\mathcal{G})$ Elvis wins $\mathcal{G}$ iff he wins the safety game* $\mathrm{Counter}_B(\mathcal{G})$.

The size of $\mathrm{Counter}_B(\mathcal{G})$ is not polynomial, even for constant B. The next step is to give an efficient reduction Poly_B producing a safety game equivalent to $\mathrm{Counter}_B(\mathcal{G})$. In particular $\mathrm{Poly}_{B(\mathcal{G})}(\mathcal{G})$ can be computed in time polynomial only in the size of $\mathcal{G}$, not in $B(\mathcal{G})$. Thus, we can define Poly from the main theorem to be $\mathrm{Poly}(\mathcal{G}) = \mathrm{Poly}_{B(\mathcal{G})}(\mathcal{G})$.

Technically, Poly relies on the insight that counter increments as performed by $\mathrm{Counter}_B$ follow a stack discipline. Incrementing the rth counter resets all counters for $r' > r$ to zero. The upper bound combines this with the fact that collapsible pushdown systems can encode large counters [7]. The second step is summarized as follows.

▶ **Lemma 3.3** (From Inefficient to Efficient). *Elvis wins* $\mathrm{Counter}_B(\mathcal{G})$ *iff he wins* $\mathrm{Poly}_B(\mathcal{G})$. *Moreover,* $\mathrm{Poly}(\mathcal{G}) = \mathrm{Poly}_{B(\mathcal{G})}(\mathcal{G})$ *is polynomial-time computable.*

It should be clear that the above lemmas, once proven, yield the main theorem. For the equivalence stated there, note that $\mathrm{Poly}(\mathcal{G}) = \mathrm{Poly}_{B(\mathcal{G})}(\mathcal{G})$ is equivalent to the game $\mathrm{Counter}_{B(\mathcal{G})}(\mathcal{G})$ by Lemma 3.3. This game, in turn, is equivalent to $\mathcal{G}$ by Lemma 3.2.

The proof of Lemma 3.3 will be direct and is given in Section 7. We explain the proof of Lemma 3.2 here, which relies on a third reduction. We define a function called Order that takes an order-n parity or safety game and produces an equivalent order-$(n-1)$ parity or safety game. The reduction already appears in [17], and generalizes the one from [32]. Let

$$\begin{array}{lcllcl} \mathcal{G}_O & = & \mathrm{Order}(\mathcal{G}), & \mathcal{G}_{C_B} & = & \mathrm{Counter}_B(\mathcal{G}), \\ \mathcal{G}_{O,C_B} & = & \mathrm{Counter}_B(\mathrm{Order}(\mathcal{G})), & \mathcal{G}_{C_B,O} & = & \mathrm{Order}(\mathrm{Counter}_B(\mathcal{G}))\ . \end{array}$$

The proof of Lemma 3.2 chases the diagram below. We rely on the observation that the games $\mathrm{Counter}_B(\mathrm{Order}(\mathcal{G}))$ and $\mathrm{Order}(\mathrm{Counter}_B(\mathcal{G}))$ are equivalent, as stated in Lemma 3.4. The proof of Lemma 3.4 needs the reductions and can be found in Section 6. The commutativity argument yields the following proof, almost in category-theoretic style.

$$\begin{array}{ccc} \mathcal{G} & \text{- - - - - - - - -}\ \mathrm{Counter}_B\ \text{- - - - - - - - -}\rightarrow & \mathcal{G}_{C_B} \\ \big\downarrow \mathrm{Order} & & \big\downarrow \mathrm{Order} \\ \mathcal{G}_O & \longrightarrow\ \mathrm{Counter}_B\ \longrightarrow & \mathcal{G}_{O,C_B} \iff \mathcal{G}_{C_B,O} \end{array}$$

▶ **Lemma 3.4** ($\mathcal{G}_{O,C_B}$ vs. $\mathcal{G}_{C_B,O}$). *Given $B \in \mathbb{N}$ and a parity game $\mathcal{G}$ over an order-n CPDS, Elvis wins* $\mathrm{Counter}_B(\mathrm{Order}(\mathcal{G}))$ *iff Elvis wins* $\mathrm{Order}(\mathrm{Counter}_B(\mathcal{G}))$.

Proof of Lemma 3.2. We induct on the order. At order-1, the result is due to Fridman and Zimmerman [14]. For the induction, without the bound, at order-n, take a winning strategy for Elvis in $\mathcal{G}$. By [17], he has a winning strategy in $\mathcal{G}_O$. By induction, Elvis has a winning

strategy in $\mathcal{G}_{O,C_B}$ and by Lemma 3.4 also in $\mathcal{G}_{C_B,O}$ when B is suitably large. Finally, again by [17], Elvis can win $\mathcal{G}_{C_B}$. I.e., we chase the diagram above from $\mathcal{G}$ to $\mathcal{G}_O$ to $\mathcal{G}_{O,C_B}$ to $\mathcal{G}_{C_B,O}$ and then up to $\mathcal{G}_{C_B}$. To prove the opposite direction, simply follow the path in reverse.

To obtain the required bound, we argue as follows: Intuitively, we have an exponential bound at order-1 by Fridman and Zimmerman. Thus, assume by induction we have a $(n-1)$-fold exponential bound for order-$(n-1)$. From an order-n system we obtain an exponentially large order-$(n-1)$ system for which an n-fold exponential bound is therefore needed. ◀

In Sections 4 and 5, we define Order and Counter$_B$, and show Lemma 3.4 in Section 6. The reduction Poly is defined in Section 7, which also sketches the proof of Lemma 3.3.

4 Order Reduction

We recall the reduction of [17] from order-n to order-$(n-1)$ parity games. This reduction also works for safety games. It is a natural extension of Carayol et al. [9] for higher-order pushdown systems without collapse, which extended Walukiewicz's reduction of pushdown parity games to finite-state parity games [32]. Due to space constraints, we only give the intuition here. It is useful when explaining the motivation behind the constructions in our parity to safety reduction.

Given an order-n CPDS $\mathcal{C}$ and a game $\mathcal{G} = (\mathcal{C}, \mathbb{O}, \mathcal{W})$ we define an order-$(n-1)$ game Order$(\mathcal{G})$ over a CPDS $\mathcal{C}'$. The order-$(n-1)$ CPDS $\mathcal{C}'$ simulates $\mathcal{C}$. The key operations are $push_n$, pop_n, $push_a$ with $\lambda(a) = n$, and *collapse* when the link is order-n. We say these operations are order-n. The remaining operations are simulated directly on the stack of $\mathcal{C}'$.

There is no $push_n$ on an order-$(n-1)$ stack. Instead, observe that if the stack is w before the $push_n$ operation, it will return to w after the corresponding pop_n (should it occur). Thus, we simulate $push_n$ by splitting the play into two branches. The first simulates the play between the $push_n$ and corresponding pop_n. The second simulates the play after the pop_n.

Instead of applying a $push_n$ operation, Elvis makes a claim about the control states the play may pop to. Also necessary is information about the smallest rank seen in the play to the pop. This claim is recorded as a vector of sets of control states $\vec{P} = (P_0, \ldots, P_m)$ which is held in the current control state. Each $p \in P_r$ is a potential future of the play, meaning that the pushed stack may be popped to p and the minimum rank seen since the push could be r. Because Elvis does not have full control of the game, he cannot give a single control state and rank: Agnetha may force him to any of a number of situations.

Once this guess has been made, Agnetha chooses whether to simulate the first play (between the push and the pop) or the second (after the pop). In the first case, $\vec{P}$ is stored in the control state. Then, when the pop occurs, Elvis wins if the destination control state is in P_r where r is the minimum rank seen (his claim was correct). In the second case, Agnetha picks a rank r and moves the play directly to some control state in P_r. This move has rank r (as the minimum rank seen needs to contribute to the parity/safety condition). In both cases, the topmost order-$(n-1)$ stack does not change (as it would be the same in both plays).

To simulate a $push_a$ with $\lambda(a) = n$ and a corresponding *collapse* we observe that the stack reached after the collapse is the same as that after a pop_n applied directly. Thus, the simulation is similar. To simulate the play up to the collapse, the current target set $\vec{P}$ is stored with the new stack character a. Then Elvis wins if a move performs a *collapse* to a control state $p \in P_r$, where r is the smallest rank seen since the order-$(n-1)$ stack, that was topmost at the moment of the original $push_a$, was pushed. To simulate the play after the collapse, we can simulate a pop_n as above.

5 Counter Reduction

We reduce parity to safety games, generalizing Fridman and Zimmermann [14] which extended Bernet et al. [1]. This reduction is not polynomial and we show in Section 7 how to achieve the desired complexity. Correctness is Lemma 3.2 (From Parity to Safety, Inefficient) .

We give the intuition here. The reduction maintains a counter for each odd rank, which can take any value between 0 and B. We also detail the counters below as they are needed in Section 7.

The insight of Bernet et al. is that, in a finite-state parity game of ℓ states, if Agnetha can force the play to pass through some odd rank r for $\ell + 1$ times without visiting a state of lower rank in between, then some state p of rank r is visited twice. Since parity games permit positional winning strategies, Agnetha can repeat the play from p ad infinitum. Thus, the smallest infinitely occurring rank must be r, and Agnetha wins the game.

Thus, Elvis plays a safety game: he must avoid visiting an odd rank too many times without a smaller rank being seen. In the safety game, counters

$$\vec{\alpha} = (\alpha_1, \alpha_3, \ldots, \alpha_m)$$

are added to the states, one for each odd rank. When a rank r is seen, then, if it is odd, α_r is incremented. Moreover, whether r is odd or even, all counters $\alpha_{r'}$ for $r' > r$ are reset to 0.

As the number of configurations is infinite, Bernet's insight does not immediately generalize to pushdown games. However, Fridman and Zimmermann observed that, from Walukiewicz [32], a pushdown parity game can be reduced to a finite-state parity game (of exponential size) as described in the previous section. This finite-state parity game can be further reduced to a safety game with the addition of counters. Their contribution is then to transfer back the counters to the pushdown game, with the following reasoning.

Recall, a push move at $(p, [aw]_1)$ is translated into a branch from a corresponding state $(p, a, \vec{P})$ in the finite-state game. There are several moves from $(p, a, \vec{P})$, some of them simulate the push, the remaining moves simulate the play after the corresponding pop. When augmented with counters the states take the form $(p, a, \vec{P}, \vec{\alpha})$. We see that, when simulating the pop in the finite-state game, the counter values are the same as in the moment when the push is simulated. That is, if we lift the counter construction to the pushdown game, after each pop move we need to reset the counters to their values at the corresponding push. Thus we store the counter values on the stack. For example, for a configuration $(p, [(a, \vec{\alpha})(b, \vec{\alpha}')]_1)$ where the current top of stack is a and the current counter values are $\vec{\alpha}$, the counter values at the moment when a was first pushed are stored on the stack as $\vec{\alpha}'$.

This reasoning generalizes to any order n. We store the counter values on the stack so that, when a pop_k operation occurs, we can retrieve the counter values at the corresponding $push_k$, and similarly for *collapse*. Note also that, when reducing from order-n to order-$(n-1)$, any branch corresponding to a play after a pop passes through a rank r which is the smallest rank seen between the push and pop. Thus, in the safety game, after each pop or collapse we need to update the counter values using r. Hence we require a rank-aware CPDS.

Let m be the maximum rank, and, for convenience, assume it is odd. We maintain a vector of counters $\vec{\alpha} = (\alpha_1, \alpha_3, \ldots, \alpha_m)$, one for each odd rank, stored in the stack alphabet as described above. We update these counters with operations $\oplus_r$ that exist for all $r \in [0, m]$ (including the even ranks). Operation $\oplus_r$ resets the counters $\alpha_{r'}$ with $r' > r$ to zero. If r is odd, it moreover increments α_r. If the bound is exceeded, an overflow occurs. Formally, $\oplus_r(\vec{\alpha}) = \texttt{NaN}$ if r is odd and $\alpha_r + 1 > B$. Otherwise, $\oplus_r(\vec{\alpha}) = \vec{\alpha}'$ where for each $\tilde{r}$

$$\alpha'_{\tilde{r}} = \alpha_{\tilde{r}} \text{ (if } \tilde{r} < r), \quad \alpha'_{\tilde{r}} = \alpha_r + 1 \text{ (if } \tilde{r} = r), \text{ and } \quad \alpha'_{\tilde{r}} = 0 \text{ (if } \tilde{r} > r).$$

6 Equivalence Result

We need equivalence of $\mathcal{G}_{O,C_B} = \text{Counter}_B(\text{Order}(\mathcal{G}))$ and $\mathcal{G}_{C_B,O} = \text{Order}(\text{Counter}_B(\mathcal{G}))$ for Lemma 3.4. The argument is that the two CPDS only differ in order of the components of their control states and stack characters. A subtlety is that when Counter_B is applied first, the contents of $\vec{P}$ are not control states of $\mathcal{G}$, but control states of $\mathcal{G}_{C_B}$. However, the additional information in the control states after Counter_B has to be consistent with $\vec{P}$, which means we can directly translate between guesses over states in the original CPDS, and those over states of the CPDS after the counter reduction.

7 Polynomial Reduction

For a game $\mathcal{G}$ over an order-n CPDS, the counters in the game $\text{Counter}_{B(\mathcal{G})}(\mathcal{G})$ blow up $\mathcal{G}$ by an n-fold exponential factor. To avoid this we use the stack-like behaviour of the counters and a result due to Cachat and Walukiewicz [7], showing how to encode large counter values into the stack of a CPDS with only polynomial overhead (in fact, collapse is not needed).

7.1 Counter Encoding

Cachat and Walukiewicz propose a binary encoding that is nested in the sense that a bit is augmented by its position, and the position is (recursively) encoded in the same way. For example, number 5 stored with 16 bits is represented by $(0,1).(1,0).(2,1).(3,0).(4,0)\ldots(15,0)$. Since four bits are required to index 16 bits, we encode position 4 as $(0,0').(1,0').(2,1').(3,0')$. Finally, position 2 of this encoding stored as $(0,0'').(1,1'')$. The players compete to (dis)prove that the indexing is done properly.

Formally we introduce distinct alphabets to encode counters for all odd ranks r:

$$\Gamma_r \;=\; \hat{\Gamma}_r \cup \{0_r, 1_r\} \;.$$

Here, $\hat{\Gamma}_r$ is a polynomially-large set of characters for the indexing. The set $\{0_r, 1_r\}$ are the bits to encode numbers. Let Γ be the union of all Γ_r.

The values of the counters are stored on the order-1 stack, with the least-significant bit topmost. The indices appear before each bit character. E.g., value 16 for counter r stored with five bits yields a sequence from $\hat{\Gamma}_r^* \,.\, 0_r \,.\, \hat{\Gamma}_r^* \,.\, 0_r \,.\, \hat{\Gamma}_r^* \,.\, 0_r \,.\, \hat{\Gamma}_r^* \,.\, 0_r \,.\, \hat{\Gamma}_r^* \,.\, 1_r$. Actually, the encoding will always use all bits, which means its length will be $(n-1)$-fold exponential.

Cachat and Walukiewicz provide game constructions to assert properties of the counter encodings. For this, play moves to a dedicated control state, from which Elvis wins iff the counters have the specified property. In [7], Elvis plays a reachability game from the dedicated state. We need the dual, with inverted state ownership and a safety winning condition, where the target state of the (former) reachability game has rank 1. Elvis's goal will be to prove the encoding wrong (it violates a property) by means of safety, Agnetha tries to build up the counters correctly and, if asked, demonstrate correctness using reachability.

For all properties, the counter to be checked must appear directly on the top of the stack (topmost on the topmost order-1 stack). If any character outside Γ_r is found, Agnetha loses. When two counters are compared, the first counter must appear directly at the top of the stack, while the second may be separated from the first by any sequence of characters from outside Γ_r (these can be popped away). The first character found from Γ_r begins the next encoding. Agnetha loses the game if none is found. The required properties are listed below.

- Encoding Check ($\mathit{encoding}_r$): For each rank r, we have a control state $\mathit{encoding}_r$. Agnetha can win the safety game from $\langle \mathit{encoding}_r, w\rangle$ only if the topmost sequence of characters from Γ_r is a correct encoding of a counter, in that all indices are present and correct.
- Equals Check (equal_r): For each r, we have a control state equal_r, from which Agnetha can win only if the topmost sequence of characters from Γ_r is identical to the next topmost sequence of Γ_r-characters. I.e., the two topmost rth counter encodings are equal.
- Counter Increment: Cachat and Walukiewicz do not define increment but it can be done via the basic rules of binary addition. We force Agnetha to increment the counter by first using pop_1 to remove characters from $\hat{\Gamma}_r \cup \{1_r\}$ until 0_r is found. Then, Agnetha must rewrite the 0_r to 1_r. Agnetha then performs as many push_a operations as she wishes, where $a \in \hat{\Gamma}_r \cup \{0_r\}$. Next, Elvis can accept this rewriting by continuing with the game, or challenge it by moving to $\mathit{encoding}_r$. This ensures that Agnetha has put enough 0_r characters on the stack (with correct indexing) to restore the number to its full length.

In this encoding one can only increment the topmost counter on the stack. That is, to increment a counter, all counters above it must be erased. Fortunately, $\oplus_r$ resets to zero all counters for ranks $r' > r$, meaning the counter updates follow a stack-like discipline. This enables the encoding to work. To store a character with counter values from the counter reduction $(a, \vec{\alpha})$ with $\vec{\alpha} = \alpha_1, \ldots, \alpha_m$ we store the character a on top and beneath we encode α_m, then α_{m-2} and so on down to α_1.

7.2 The Simulation

The following definition is completed in the following sections. Correctness is stated in Lemma 3.3 (From Inefficient to Efficient) .

▶ **Definition 7.1** (Poly_B)**.** Given a parity game $\mathcal{G} = (\mathcal{C}, \mathbb{O}, \mathcal{W})$ over the order-n CPDS $\mathcal{C} = (\mathcal{P}, \Sigma, \lambda, \mathcal{R}, p_I, a_I, \rho)$ and a bound B n-fold exponential in the size of the game, we define the safety game $\mathrm{Poly}_B(\mathcal{G}) = (\mathcal{C}', \mathbb{O}', 2^\omega)$ where $\mathcal{C}' = (\mathcal{Q}, \Sigma', \lambda', \mathcal{R}', p_I', a_I', \rho')$. The missing components are defined below.

We aim to simulate $\mathrm{Counter}_B(\mathcal{G})$ compactly. This simulation is move-by-move, as follows.

A $\mathit{push}_{(a,\vec{\alpha})}$ of a character with counter values $(a, \vec{\alpha})$ with $\vec{\alpha} = \alpha_1, \ldots, \alpha_m$ (where the max-rank is m) is simulated by first pushing a special character ℓ_k to save the link (with push_{ℓ_k}). Then, since the counter values are a copy of the preceding counter values on the stack, Agnetha pushes an encoding for α_1 using Γ_1 after which Elvis can accept the encoding, check that it is a proper encoding using $\mathit{encoding}_1$, or check that it is a faithful copy of the preceding value of α_1 using equal_1. We do this for all odd ranks through to m. Then the only move is to push a with push_a.

Each push_k and pop_k, with $k \in [2, n]$, is simulated directly by the same operation. For a pop_1 we (deterministically) remove all topmost characters (using pop_1) up to and including the first $\ell_{k'}$ (for some k'). We simulate *collapse* like pop_1, but we apply *collapse* to $\ell_{k'}$.

A $\mathit{rew}_{(a,\vec{\alpha})}$ that does not change the counters can be simulated by rewriting the topmost character. If $\oplus_r$ is applied, we force Agnetha to play as follows. If r is even, Agnetha removes the counters for $r' > r$. She replaces them with zero values by pushing characters from $\hat{\Gamma}_{r'} \cup \{0_{r'}\}$. After each counter is rewritten, Elvis can accept the encoding, or challenge it with $\mathit{encoding}_{r'}$. Finally, a is pushed onto the stack. If r is odd, the counters for r' are removed as before. Then we do an increment as described above, with Elvis losing if the increment fails. Note, it fails only if there is no 0_r in the encoding, which means the counter is at its maximum value and there is an overflow (indicating Elvis loses the parity game). If it succeeds, zero values for the counters $r' > r$ and a are pushed to the stack as before.

Control States and Alphabet

We define the control states $\mathcal{Q}$ with $\mathbb{O}'$ and ρ' as well as the alphabet. First,

$$\mathcal{Q} \;=\; \mathcal{P} \cup (\mathcal{P} \times [0,m]) \cup \{\#,\$\} \cup \mathcal{P}_{CW} \;\cup \mathcal{P}_{OP}\ .$$

where m is the maximum rank. The set $\mathcal{P}_{CW}$ is the control states of the Cachat-Walukiewicz games implementing $\mathit{encoding}_r$ and equal_r. The size is polynomial in $\mathcal{G}$. We have $\mathcal{P}_{OP} =$

$$\left\{ \begin{array}{c} (\mathrm{inc}, r, p), (\mathrm{copy}, r, a, p), (\mathrm{zero}, r, a, p), (\mathrm{pop}_1, \Upsilon, r, p), \\ (\mathrm{inc}, r, a, p), (\mathrm{cchk}, r, a, p), (\mathrm{zchk}, r, a, p), (\mathrm{collapse}, \Upsilon, r, p) \end{array} \;\middle|\; \begin{array}{l} r \in [0,m] \wedge a \in \Sigma \;\wedge \\ p \in \mathcal{P} \;\wedge \Upsilon \in \{\mathrm{A}, \mathrm{E}\} \end{array} \right\}$$

to control the simulation of the operations as sketched above. We describe the states below.

The states in $\mathcal{P}_{CW}$ have the same rank and owner as in the Cachat-Walukiewicz games (more precisely the dual, see above). All other states have rank 2 except # which has rank 1. It (resp. \$) is the losing sink for Elvis (resp. Agnetha). The states in $\mathcal{P} \cup (\mathcal{P} \times [0,m]) \cup \{\#\}$ are used as in $\mathrm{Counter}_B(\mathcal{G})$ to directly simulate $\mathcal{G}$. The owners are as in $\mathcal{G}$.

A state (inc, r, p) begins an application of $\oplus_r$. The top-of-stack character is saved by moving to (inc, r, a, p). The owner of these states does not matter, we give them to Agnetha. In (inc, r, a, p), the stack is popped down to the counter for r. If r is odd, the least significant zero is set to one. Then, control moves to (zero, r, a, p). In (zero, r, a, p), zero counters for ranks r and above are pushed to the stack, followed by a push of a and a return to control state p. The state is owned by Agnetha. The state (zchk, r, a, p) is used by Elvis to accept or challenge that the encoding has been re-established completely. It is owned by Elvis.

The controls (copy, r, a, p) copy the counters for ranks r and above (the current values) and push the copies to the stack, followed by a push of a and a return to control state p. The state is owned by Agnetha. After this phase, the play moves to (cchk, r, a, p) where Elvis can accept or test whether the copy has been done correctly. This state is owned by Elvis.

The controls $(\mathrm{pop}_1, \Upsilon, r, p)$ and $(\mathrm{collapse}, \Upsilon, r, p)$ where $\Upsilon \in \{\mathrm{A}, \mathrm{E}\}$ are used to execute a pop_1 or *collapse*. For the latter, we pop to the next ℓ_k character, perform the collapse and record that the rth counter needs to be incremented. In case the collapse is not possible (because it would empty the stack) play may also move to a sink state that is losing for the player Υ who instigated the collapse. The case of pop_1 pops ℓ_k. The owner in each case is Υ as they will avoid moving to their (losing) sink state if the pop_1 or *collapse* is possible.

The alphabet and initial control state and stack character are

$$\Sigma' = \Sigma \cup \Gamma \cup \Delta \quad \text{and} \quad p'_I = (\mathrm{zero}, 1, a_I, p_I) \quad \text{and} \quad a'_I = \ell_k \text{ where } \lambda(a_I) = k\ .$$

The alphabet is extended by the characters required for the counter and link encodings. Recall that Γ is the union of the counter alphabets, which are of polynomial size. We use $\Delta = \{\ell_1, \ldots, \ell_m\}$ for the link characters. We assign $\lambda'(\ell_k) = k$ and $\lambda'(a) = 1$ for all other a.

The task of the initial state and initial stack character is to establish the encoding of $(a_I, (0, \ldots, 0))$ in $\mathrm{Counter}_B(\mathcal{G})$ and then move to the initial state of $\mathcal{G}$. With the above description, $(\mathrm{zero}, 1, a_I, p_I)$ will establish zeros in all counters (from 1 to m), push the initial character a_I of the given game, and move to state p_I. The initial character ℓ_k is the correct bottom element for the encoding of $(a_I, (0, \ldots, 0))$.

Rules

The rules of $\mathcal{C}'$ follow $\mathcal{C}$ and maintain the counters. $\mathcal{R}'$ contains (only) the following rules. First, we have $\mathcal{R}_{CW}$ which are the (dual of the) rules of Cachat and Walukiewicz implementing $\mathit{encoding}_r$ and equal_r. The rules simulating the operations appear below. Note, pop and collapse use rank-awareness. We give the increment and copy rules after the basic operations.

- Order-k push: $(p, a, push_k, (\text{inc}, \rho(p'), p'))$ when $(p, a, push_k, p') \in \mathcal{R}$.
- Character push: $(p, a, push_{\ell_k}, (\text{copy}, 1, b, p'))$ when $(p, a, push_b, p') \in \mathcal{R}$ and $\lambda(b) = k$.
- Rewrite: $(p, a, rew_b, (\text{inc}, \rho(p'), p'))$ when $(p, a, rew_b, p') \in \mathcal{R}$.
- Pop (> 1): $(p, a, pop_k, (\text{inc}, r, p'))$ when $(p, a, pop_k, p') \in \mathcal{R}$ and $r = \min(\rho(p'), \text{Rk}(k))$.
- Pop ($= 1$) and Collapse: $(p, a, pop_1, (o, \Upsilon, r, p'))$ when $(p, a, o, p') \in \mathcal{R}$ with $\mathbb{O}(p) = \Upsilon$, operation o being pop_1 or *collapse*, and $r = \min(\rho(p'), r')$. Here, if $o = pop_1$ then $r' = \text{Rk}(1)$. Otherwise, if $o = collapse$ then $r' = \text{Rk}(\ell)$.
  Then, we have all rules $((o, \Upsilon, r, p'), a', pop_1, (o, \Upsilon, r, p'))$ for $a' \in \Gamma$. We perform the operation with $((o, \Upsilon, r, p'), \ell_{k'}, o, (\text{inc}, r, p'))$. To allow for the case where the pop or collapse cannot be performed (because the stack would empty), we also have the rules $((o, \text{A}, r, p'), \ell_{k'}, nop, \$)$ and $((o, \text{E}, r, p'), \ell_{k'}, nop, \#)$.
- Sink states: $(\$, a, nop, \$)$ and $(\#, a, nop, \#)$.

To copy counters, for each odd r and $b \in \Gamma_r$ we have $((\text{copy}, r, a, p), *, push_b, (\text{copy}, r, a, p))$. We use $*$ to indicate that the transition exists for all stack symbols. When a counter has been pushed, like in the case of pushing zeros, Agnetha hands over the control to Elvis to check the result: $((\text{copy}, r, a, p), *, nop, (\text{cchk}, r, a, p))$. Elvis can challenge the copied counter or accept it was copied correctly. To challenge, we use $((\text{cchk}, r, a, p), *, nop, equal_r)$. To accept, the behavior depends on r. If $r < m$, we move to copying the next counter $((\text{cchk}, r, a, p), *, nop, (\text{copy}, r + 2, a, p))$. When $r = m$, we finish copying and move to incrementing with rules of the form $((\text{cchk}, r, a, p), *, nop, (\text{inc}, \rho(p), a, p))$.

To increment a counter, we first pop and store the topmost stack character with the rule $((\text{inc}, r, p), a, pop_1, (\text{inc}, r, a, p))$. Agnetha then removes all counters for ranks higher than the given r with the following rules, where $b \in \Gamma_{r'}$ with $r' > r$: $((\text{inc}, r, a, p), b, pop_1, (\text{inc}, r, a, p))$.

When r is even we add back 0 counters once enough have been removed using (with $b \in \Gamma_{r-1}$ if $r > 1$ else $b \in \Delta$) the rules $((\text{inc}, r, a, p), b, nop, (\text{zero}, r + 1, a, p))$. If r is odd, we start incrementing the rth counter with $((\text{inc}, r, a, p), b, pop_1, (\text{inc}, r, a, p))$ for all $b \in \hat{\Gamma}_r \cup \{1_r\}$.

When 0_r is found, we use $((\text{inc}, r, a, p), 0_r, rew_{1_r}, (\text{zero}, r, a, p))$. If no zero bit is found, we have an overflow and move to the sink state with $((\text{inc}, r, a, p), b, nop, \#)$ for $b \in \Gamma_{r-2}$ if $r > 2$ and $b \in \Delta$ otherwise. With $((\text{zero}, r, a, p), *, push_b, (\text{zero}, r, a, p))$ for $b \in \hat{\Gamma}_r \cup \{0_r\}$ we add back zeros to the incremented counter and reset all erased counters. To finish the phase that adds zeros for the rth counter, Agnetha hands over the control to Elvis, $((\text{zero}, r, a, p), *, nop, (\text{zchk}, r, a, p))$.

Elvis can now check if all bits of the counter are present or accept the result. To challenge the encoding, he uses $((\text{zchk}, r, a, p), *, nop, encoding_r)$. When accepting it, if $r < m$, more counters need to be reset. We move to the next using $((\text{zchk}, r, a, p), *, nop, (\text{zero}, r + 2, a, p))$. If $r = m$, there are no more counters to handle and with the rule $((\text{zchk}, r, a, p), *, push_a, p)$ Elvis re-establishes the control state and stack character.

8 Conclusion

We gave a polynomial-time reduction from parity games played over order-n CPDS to safety games over order-n CPDS. Such a reduction has been an open problem [14] (related are also [1, 2]). It builds counters into the stack to count occurrences of odd ranks at the current stack level (without seeing a smaller rank). If this number grows large then Elvis would lose the parity game (if play continued). To obtain a polynomial reduction we use the insight that the counters follow a stack discipline. For correctness, we use a commutativity argument for the rank counter and order reductions. As a theoretical interest, the result explains the matching complexities of parity and safety games over CPDS. From a practical standpoint, the reduction may inspire the use of advanced safety checking tools for, and the transfer of technology from safety to, the empirically harder problem of parity game analysis.

References

1 J. Bernet, D. Janin, and I. Walukiewicz. Permissive strategies: from parity games to safety games. *ITA*, 2002.

2 D. Berwanger and L. Doyen. On the Power of Imperfect Information. In *FSTTCS*, 2008.

3 A. Biere, C. Artho, and V. Schuppan. Liveness checking as safety checking. *Electr. Notes Theor. Comput. Sci.*, 2002.

4 R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. Decidability in parameterized verification. *SIGACT News*, 47(2), 2016.

5 A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, 1997.

6 C. H. Broadbent and N. Kobayashi. Saturation-based model checking of higher-order recursion schemes. In *CSL*, 2013.

7 T. Cachat and I. Walukiewicz. The complexity of games on higher order pushdown automata. *CoRR*, abs/0705.0262, 2007.

8 C. S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In *STOC*, 2017.

9 A. Carayol, M. Hague, A. Meyer, C.-H. L. Ong, and O. Serre. Winning Regions of Higher-Order Pushdown Games. In *LICS*, 2008.

10 J. Daniel, A. Cimatti, A. Griggio, S. Tonetta, and S. Mover. Infinite-state liveness-to-safety via implicit abstraction and well-founded relations. In *CAV*, 2016.

11 J. Engelfriet. Iterated stack automata and complexity classes. *Inf. Comput.*, 95(1):21–75, 1991.

12 A. Farzan, Z. Kincaid, and A. Podelski. Proving liveness of parameterized programs. In *LICS*, 2016.

13 J. Fearnley, S. Jain, S. Schewe, F. Stephan, and D. Wojtczak. An ordered approach to solving parity games in quasi polynomial time and quasi linear space. In *SPIN*, 2017.

14 W. Fridman and M. Zimmermann. Playing pushdown parity games in a hurry. In *GandALF*, 2012.

15 K. Fujima, S. Ito, and N. Kobayashi. Practical alternating parity tree automata model checking of higher-order recursion schemes. In *APLAS*, 2013.

16 M. Hague, R. Meyer, S. Muskalla, and M. Zimmermann. Parity to safety in polynomial time for pushdown and collapsible pushdown systems. *CoRR*, abs/1805.02963, 2018. `arXiv:1805.02963`.

17 M. Hague, A. S. Murawski, C.-H. Luke Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, 2008.

18 M. Jurdzinski and R. Lazic. Succinct progress measures for solving parity games. In *LICS*, pages 1–9, 2017.

19 T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS '02: Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures*, pages 205–222, London, UK, 2002. Springer-Verlag.

20 N. Kobayashi. HorSat2: A model checker for HORS based on SATuration. A tool available at `http://www-kb.is.s.u-tokyo.ac.jp/~koba/horsat2/`.

21 I. V. Konnov, M. Lazic, H. Veith, and J. Widder. A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In *POPL*, 2017.

22 G. Lenzi. The modal μ-calculus: a survey. *Task quarterly*, 9(3):293–316, 2005.

23 R. P. Neatherway and C.-H. L. Ong. Travmc2: higher-order model checking for alternating parity tree automata. In *SPIN*, 2014.

24 C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, 2006.

25 O. Padon, J. Hoenicke, G. Losa, A. Podelski, M. Sagiv, and S. Shoham. Reducing liveness to safety in first-order logic. *PACMPL*, 2017.

26 A. Podelski and A. Rybalchenko. Transition invariants. In *LICS*, 2004.

27 A. Podelski and A. Rybalchenko. Transition invariants and transition predicate abstraction for program termination. In *TACAS*, 2011.

28 S. J. Ramsay, R. P. Neatherway, and C.-H. L. Ong. A type-directed abstraction refinement approach to higher-order model checking. In *POPL*, 2014.

29 V. Schuppan and A. Biere. Liveness checking as safety checking for infinite state spaces. *Electr. Notes Theor. Comput. Sci.*, 2005.

30 S. Sohail and F. Somenzi. Safety first: A two-stage algorithm for ltl games. In *FMCAD*, 2009.

31 T. Terao and N. Kobayashi. A zdd-based efficient higher-order model checking algorithm. In *APLAS*, 2014.

32 I. Walukiewicz. Pushdown processes: Games and model checking. In *CAV*, 1996.

Quantum Generalizations of the Polynomial Hierarchy with Applications to QMA(2)

Sevag Gharibian
University of Paderborn, Paderborn, North Rhine-Westphalia, Germany, and Virginia Commonwealth University, Richmond, Virginia, USA

Miklos Santha
CNRS, IRIF, Université Paris Diderot, Paris, France and Centre for Quantum Technologies, National University of Singapore, Singapore

Jamie Sikora
Perimeter Institute for Theoretical Physics, Waterloo, Ontario, Canada

Aarthi Sundaram
Joint Center for Quantum Information and Computer Science, University of Maryland, College Park, Maryland, USA

Justin Yirka
Virginia Commonwealth University, Richmond, Virginia, USA

Abstract

The polynomial-time hierarchy (PH) has proven to be a powerful tool for providing separations in computational complexity theory (modulo standard conjectures such as PH does not collapse). Here, we study whether two quantum generalizations of PH can similarly prove separations in the quantum setting. The first generalization, QCPH, uses classical proofs, and the second, QPH, uses quantum proofs. For the former, we show quantum variants of the Karp-Lipton theorem and Toda's theorem. For the latter, we place its third level, $\mathrm{Q}\Sigma_3$, into NEXP using the Ellipsoid Method for efficiently solving semidefinite programs. These results yield two implications for QMA(2), the variant of Quantum Merlin-Arthur (QMA) with two unentangled proofs, a complexity class whose characterization has proven difficult. First, if QCPH = QPH (i.e., alternating quantifiers are sufficiently powerful so as to make classical and quantum proofs "equivalent"), then QMA(2) is in the Counting Hierarchy (specifically, in $\mathrm{P}^{\mathrm{PP}^{\mathrm{PP}}}$). Second, unless QMA(2) = $\mathrm{Q}\Sigma_3$ (i.e., alternating quantifiers do not help in the presence of "unentanglement"), QMA(2) is strictly contained in NEXP.

2012 ACM Subject Classification Theory of computation → Quantum complexity theory, Theory of computation → Complexity classes, Theory of computation → Semidefinite programming

Keywords and phrases Complexity Theory, Quantum Computing, Polynomial Hierarchy, Semidefinite Programming, QMA(2), Quantum Complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.58

Related Version A full version of this work is available at `https://arxiv.org/abs/1805.11139`.

Acknowledgements SG and AS thank the Center for Quantum Technologies at the National University of Singapore for their support and hospitality, where part of this research was carried out. SG acknowledges support from NSF grants CCF-1526189 and CCF-1617710. AS is supported by the Department of Defense. Research at the Centre for Quantum Technologies is partially funded by the Singapore Ministry of Education and the National Research Foundation under grant R-710-000-012-135. This research was supported in part by the QuantERA ERA-NET

Cofund project QuantAlgo. Research at Perimeter Institute is supported by the Government of Canada through the Department of Innovation, Science and Economic Development Canada and by the Province of Ontario through the Ministry of Research, Innovation and Science.

1 Introduction

The polynomial time hierarchy (PH) [28] is a staple of computational complexity theory, and generalizes P, NP and co-NP with the use of alternating existential ($\exists$) and universal ($\forall$) operators. Roughly, a language $L \subseteq \{0,1\}^*$ is in Σ_i^p, the ith level of PH, if there exists a polynomial-time deterministic Turing machine M that acts as a verifier and accepts i proofs $y_1, \ldots, y_i$ polynomially bounded in size such that:

$$\begin{aligned} x \in L &\Rightarrow \exists y_1 \forall y_2 \exists y_3 \cdots Q_i y_i \text{ such that } M \text{ accepts } (x, y_1, \ldots, y_i), \\ x \notin L &\Rightarrow \forall y_1 \exists y_2 \forall y_3 \cdots \overline{Q}_i y_i \text{ such that } M \text{ rejects } (x, y_1, \ldots, y_i), \end{aligned}$$

where $Q_i = \exists$ if i is odd and $Q_i = \forall$ if i is even, and $\overline{Q}$ denotes the complement of Q. Then, PH is defined as the union over all Σ_i^p for all $i \in \mathbb{N}$. The study of PH has proven remarkably fruitful in the classical setting, from celebrated results such as Toda's Theorem [30], which shows that PH is contained in $\mathrm{P}^{\#\mathrm{P}}$, to the Karp-Lipton Theorem [21], which says that unless PH collapses to its second level, NP does not have polynomial size non-uniform circuits.

As PH has played a role in separating complexity classes (assuming standard conjectures like "PH does not collapse"), it is natural to ask whether *quantum* generalizations of PH can be used to separate *quantum* complexity classes. Here, there is some flexibility in defining "quantum PH", as there is more than one well-defined notion of "quantum NP": The first, Quantum-Classical Merlin Arthur (QCMA) [6], is a quantum analogue of Merlin-Arthur (MA) with a classical proof but quantum verifier. The second, Quantum Merlin Arthur (QMA) [22], is QCMA except with a quantum proof. Generalizing each of these definitions leads to (at least) two possible definitions for "quantum PH", the first using classical proofs (denoted QCPH), and the second using quantum proofs (denoted QPH).

With these definitions in hand, our aim is to separate quantum classes whose complexity characterization has generally been difficult to pin down. A prime example is QMA(2), the variant of QMA with two "unentangled" quantum provers. While the classical analogue of QMA(2) (i.e. an MA proof system with two provers) trivially equals MA, in the quantum regime multiple unentangled provers are conjectured to yield a more powerful proof system (e.g. there exist problems in QMA(2) not known to be in QMA) [24, 10, 9, 1]. For this reason, QMA(2) has received much attention, despite which the strongest bounds known on QMA(2) remain the trivial ones: QMA $\subseteq$ QMA(2) $\subseteq$ NEXP. (Note: QMA $\subseteq$ PP [23, 27].) In this work, we show that, indeed, results about the structure of QCPH or QPH yield implications about the power of QMA(2).

1.1 Results, techniques, and discussion

We begin by informally defining the two quantum generalizations of PH to be studied.

How to define a "quantum PH"? The first definition, QCPH, has its ith level $\mathrm{QC}\Sigma_i$ defined analogously to Σ_i^p, except we replace the Turing machine M with a polynomial-size uniformly generated quantum circuit V such that:

$$\begin{aligned} x \in A_{\text{yes}} &\Rightarrow \exists y_1 \forall y_2 \exists y_3 \cdots Q_i y_i \text{ s.t. } V \text{ accepts } (x, y_1, \ldots, y_i) \text{ with probability} \geq 2/3, \quad (1) \\ x \in A_{\text{no}} &\Rightarrow \forall y_1 \exists y_2 \forall y_3 \cdots \overline{Q}_i y_i \text{ s.t. } V \text{ accepts } (x, y_1, \ldots, y_i) \text{ with probability} \leq 1/3, \quad (2) \end{aligned}$$

where the use of a language L has been replaced with a *promise problem*[1] $A = (A_{\text{yes}}, A_{\text{no}})$ (since $\text{QC}\Sigma_i$ uses a bounded error verifier). The values $2/3$ and $1/3$ are respectively the *completeness* and *soundness* parameters for A and the interval $(1/3, 2/3)$ where no acceptance probabilities are present is termed the *promise gap* for A. Notice that QCPH defined as $\bigcup_{i\in\mathbb{N}} \text{QC}\Sigma_i$, is a generalization of QCMA in that $\text{QC}\Sigma_1 = \text{QCMA}$.

We next define QPH using *quantum* proofs. Here, however, there are various possible definitions one might consider. Can the quantum proofs be *entangled* between alternating quantifiers? (If not, we are enforcing "unentanglement" as in QMA(2). Allowing entanglement, on the other hand, might yield classes similar to QIP; however, note that QIP = QIP(3) (i.e. QIP collapses to a 3-message proof system) [23, 27], and so it is not clear that allowing entanglement leads to an "interesting" hierarchy.) Assuming proofs are unentangled, should the proofs be *pure* or *mixed* quantum states? (For QMA and QMA(2), standard convexity arguments show both classes of proofs are equivalent, but such arguments fail when *alternating* quantifiers are allowed.)

Here, we define QPH to have its ith level, $\text{Q}\Sigma_i$, defined similarly to $\text{QC}\Sigma_i$, except each classical proof y_j is replaced with a mixed quantum state ρ_j on polynomially many qubits. We say a promise problem $A = (A_{\text{yes}}, A_{\text{no}})$ is in $\text{Q}\Sigma_i$ if it satisfies the following conditions:

$$x \in A_{\text{yes}} \Rightarrow \exists\rho_1\forall\rho_2\exists\rho_3\cdots Q_i\rho_i \text{ such that } V \text{ accepts } (x,\rho_1,\ldots,\rho_i) \text{ with probability} \geq 2/3,$$
$$x \in A_{\text{no}} \Rightarrow \forall\rho_1\exists\rho_2\forall\rho_3\cdots \overline{Q}_i\rho_i \text{ such that } V \text{ accepts } (x,\rho_1,\ldots,\rho_i) \text{ with probability} \leq 1/3.$$

Note that $\text{QPH} := \bigcup_{i\in\mathbb{N}} \text{Q}\Sigma_i$, $\text{Q}\Sigma_1 = \text{QMA}$ and $\text{QMA}(2) \subseteq \text{Q}\Sigma_3$ (simply ignore the second proof); where the latter two hold because a lack of alternating quantifiers allows convexity arguments to yield that all proofs can be assumed to be pure. Our results are now stated as follows under three headings.

An analogue of Toda's theorem for QCPH. As previously mentioned, PH is one way to generalize NP using alternations. Another approach is to *count* the number of solutions for an NP-complete problem such as SAT, as captured by #P. Surprisingly, these two notions are related, as shown by the following celebrated theorem of Toda.

▶ **Theorem 1** (Toda's Theorem [30]). $\text{PH} \subseteq \text{P}^{\#\text{P}}$.

In the quantum setting, for QCPH, it can be shown using standard arguments involving enumeration over classical proofs that $\text{QCPH} \subseteq \text{PSPACE}$. However, we are able to provide the following stronger result.

▶ **Theorem 2** (A quantum-classical analogue of Toda's theorem). $\text{QCPH} \subseteq \text{P}^{\text{PP}^{\text{PP}}}$.

Thus, we "almost" recover the original bound of Toda's theorem[2], except we require an oracle for the *second* level of the Counting Hierarchy (CH). CH can be defined with its first level as $\mathbf{C}^p_1 = \text{PP}$ and its kth level for $k \geq 1$ as $\mathbf{C}^p_{k+1} = \text{PP}^{\mathbf{C}^p_k}$.

Why did we move up to the next level of CH? There are two difficulties in dealing with QCPH (see Section 2 for a detailed discussion). The first can be sketched as follows. Classically, many results involving PH, from basic ones implying the collapse of PH to more

[1] Recall that unlike a decision problem, for a promise problem $A = (A_{\text{yes}}, A_{\text{no}})$, it is not necessarily true that for all inputs $x \in \Sigma^*$, either $x \in A_{\text{yes}}$ or $x \in A_{\text{no}}$. In the case of proof systems such as QCPH, when $x \notin A_{\text{yes}} \cup A_{\text{no}}$, V can output an arbitrary answer.

[2] PP is the set of languages decidable in probabilistic polynomial time with unbounded error. Note $\text{P}^{\text{PP}} = \text{P}^{\#\text{P}}$.

advanced statements such as Toda's theorem, use the following recursive idea (demonstrated with Σ_2 for simplicity): By fixing the existentially quantified proof of Σ_2 the remnant reduces to a co-NP problem, i.e. we can recurse to a lower level of PH. In the quantum setting, however, this does not hold – fixing the existentially quantified proof for $\mathrm{QC}\Sigma_2$ does *not* necessarily yield a co-QCMA problem as some acceptance probabilities may fall in the $(1/3, 2/3)$ promise gap which cannot happen for a problem in co-QCMA! (This is due to the same phenomenon that has been an obstacle to resolving whether $\exists \cdot \mathrm{BPP}$ equals MA (see Remark 17).) Thus, we cannot directly generalize recursive arguments from the classical setting to the quantum setting. The second difficulty is trickier to explain briefly (see Section 2.2 for details). Roughly, Toda's proof that $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{PP}}$ crucially uses the Valiant-Vazirani (VV) theorem [31], which has one-sided error (i.e. VV may map YES instances of SAT to NO instances of UNIQUE-SAT, but NO instances of SAT are always mapped to NO instances of UNIQUE-SAT). The VV theorem for QCMA [5] also has this property, but in addition it can output instances which are "invalid". Essentially, they violate the promise of the problem that the QCMA-VV theorem maps to. Combining such *invalid* instances with *alternating* quantifiers, poses problems in extending the parity arguments used in Toda's proof to the QCPH setting.

To circumvent these difficulties, we exploit a high-level idea from [15] where an oracle for SPECTRAL GAP[3] was used to detect "invalid" QMA instances[4]. In our setting, the "correct" choice of oracle turns out to be a Precise-BQP oracle, where Precise-BQP is roughly BQP with an exponentially small promise gap. Using this, we are able to essentially "remove" the promise gap of QCPH altogether, thus recovering a "decision problem" which does not pose the difficulties above. Specifically, this mapping is achieved by Lemma 18 (Cleaning Lemma), which shows that $\forall i \in \mathbb{N},\ \mathrm{QC}\Sigma_i \subseteq \exists \cdot \forall \cdot \cdots \cdot Q_i \cdot \mathrm{P}^{\mathrm{PP}}$.

Notice that although we use a Precise-BQP oracle above, the Cleaning Lemma shows containment using a PP oracle. This is because, Precise-BQP $\subseteq$ PP as shown in Lemma 14 and Corollary 15. One may ask whether our proof technique would also work with an oracle *weaker* than PP. We show, in Theorem 27, that this is unlikely as the problem of detecting proofs in promise gaps of quantum verifiers is PP-complete.

Finally, an immediate corollary of Theorem 2 and the fact that $\mathrm{QMA}(2) \subseteq \mathrm{QPH}$ is:

▶ **Corollary 3.** *If* $\mathrm{QCPH} = \mathrm{QPH}$*, then* $\mathrm{QMA}(2) \subseteq \mathrm{P}^{\mathrm{PP}^{\mathrm{PP}}}$.

In other words, if alternating quantifiers are so powerful so as to make classical and quantum proofs equivalent in power, then it can be shown that QMA(2) is contained in CH (and thus in PSPACE). For comparison, $\mathrm{QMA} \subseteq \mathrm{P}^{\mathrm{QMA[log]}} \subseteq \mathrm{PP}$ [23, 33, 27, 15].

QPH versus NEXP. We next turn to the study of *quantum* proofs, i.e. QPH. As mentioned above, the best known upper bound on QMA(2) is NEXP – a non-deterministic verifier can simply guess an exponential-size description of the proof. When alternating quantifiers are present, however, this strategy seemingly no longer works. In other words, it is not even clear that QPH $\subseteq$ NEXP! This is in stark contrast to the explicit P^{PP} upper bound for PH [30]. In this section, our goal is to use semidefinite programming to give bounds on some levels of QPH. As we will see, this will yield the existence of a complexity class lying "between" QMA(2) and NEXP.

[3] This problem determines whether the spectral gap of a given local Hamiltonian is "small" or "large".
[4] This was used, in turn, to show in conjunction with [8] that SPECTRAL GAP is $\mathrm{P}^{\text{Unique-QMA[log]}}$-hard.

▶ **Theorem 4** (Informal Statement). *It holds that* $\mathrm{Q}\Sigma_2 \subseteq \mathrm{EXP}$ *and* $\mathrm{Q}\Pi_2 \subseteq \mathrm{EXP}$*, even when the completeness-soundness gap is inverse doubly-exponentially small.*

The proof idea is to map alternating quantifiers to an optimization problem with alternating minimizations and maximizations. Namely, to decide if $x \in A_{\mathrm{yes}}$ or $x \in A_{\mathrm{no}}$ for a $\mathrm{Q}\Sigma_i$ promise problem $A = (A_{\mathrm{yes}}, A_{\mathrm{no}})$, where i is even, we can solve for α defined as the optimal value of the optimization problem:

$$\alpha := \max_{\rho_1} \min_{\rho_2} \max_{\rho_3} \cdots \min_{\rho_i} \langle C, \rho_1 \otimes \rho_2 \otimes \cdots \otimes \rho_i \rangle \tag{3}$$

where C is the POVM operator[5] corresponding to the ACCEPT state of the verifier. This is a non-convex problem, and as such is hard to solve in general. Our approach is to cast the case of $i = 2$ as a semidefinite program (SDP), allowing us to *efficiently* approximate α.

The next natural question is whether a similar SDP reformulation might be used to show whether $\mathrm{Q}\Sigma_3$ or $\mathrm{Q}\Pi_3$ is contained in EXP. Unfortunately, this is likely to be difficult – indeed, if there existed a "nice" SDP for the optimal success probability of $\mathrm{Q}\Sigma_3$ protocols, then it would imply $\mathrm{QMA}(2) \subseteq \mathrm{EXP}$, resolving the longstanding open problem of separating QMA(2) from NEXP (recall $\mathrm{QMA}(2) \subseteq \mathrm{Q}\Sigma_3$). Likewise, a "nice" SDP for $\mathrm{Q}\Pi_3$ would place co-QMA(2) $\subseteq$ EXP.

To overcome this, we resort to non-determinism by stepping up to NEXP. Namely, one can non-deterministically guess the first proof of a $\mathrm{Q}\Sigma_3$ protocol, then approximately solve the SDP for the resulting $\mathrm{Q}\Pi_2$-flavoured computation. Hence, we have the following as a corollary of Theorem 28.

▶ **Theorem 5** (Informal Statement). *It holds true that* $\mathrm{QMA}(2) \subseteq \mathrm{Q}\Sigma_3 \subseteq \mathrm{NEXP}$ *and* co-QMA(2) $\subseteq \mathrm{Q}\Pi_3 \subseteq$ co-NEXP*, even when the completeness-soundness gap is inverse doubly-exponentially small. All the containments hold with equality in the inverse exponentially small completeness-soundness gap setting as* $\mathrm{QMA}(2) = \mathrm{NEXP}$ *in this case* [29].

Three remarks are in order. First, note that our results in this section are independent of the gate set. Second, in principle, it remains plausible that the fourth level of QPH already exceeds NEXP in power. Finally, we have the following implication for QMA(2). Assuming PH does not collapse, alternating quantifiers strictly add power to NP proof systems. If alternating quantifiers similarly add power in the quantum setting, then it would separate QMA(2) from NEXP via the following immediate corollary of Theorem 31.

▶ **Corollary 6.** *If* $\mathrm{QMA}(2) \neq \mathrm{Q}\Sigma_3$*, i.e. if the second universally quantified proof of* $\mathrm{Q}\Sigma_3$ *adds proving power, then* $\mathrm{QMA}(2) \neq \mathrm{NEXP}$*. Similarly, if* co-QMA(2) $\neq \mathrm{Q}\Pi_3$*, then* co-QMA(2) $\neq$ co-NEXP*.*

A quantum generalization of the Karp-Lipton Theorem. Finally, our last result studies a topic which is unrelated to QMA(2) – the well-known Karp-Lipton Theorem [21]. The latter shows that if NP-complete problems can be solved by polynomial-size non-uniform Boolean circuits, then $\Sigma_2 = \Pi_2$, which in turn implies that PH collapses to its second level. Here, a "non-uniform" circuit family means that the generation of a circuit for an input depends on the length of the input. The class of decision problems solved by such circuits is $\mathrm{P}_{/\mathrm{poly}}$.

▶ **Theorem 7** (Karp-Lipton [21]). *If* $\mathrm{NP} \subseteq \mathrm{P}_{/\mathrm{poly}}$ *then* $\Pi_2 = \Sigma_2$*.*

[5] A POVM is a set of Hermitian positive semi-definite operators that sums to the identity. In this case, the POVM has two operators – corresponding to the ACCEPT and REJECT states of the verifier.

In this work, we ask: Does $\mathrm{QCMA} \subseteq \mathrm{BQP}_{/\mathrm{mpoly}}$ imply $\mathrm{QC}\Pi_2 = \mathrm{QC}\Sigma_2$? Here, $\mathrm{BQP}_{/\mathrm{mpoly}}$ is the bounded-error analogue of $\mathrm{P}_{/\mathrm{poly}}$ with polynomial-size non-uniform quantum circuits (see Section 4 for formal definition). Unfortunately, generalizing the proof of the Karp-Lipton theorem is problematic for the same "$\exists \cdot \mathrm{BPP}$ versus MA phenomenon" encountered earlier in extending Toda's result. Namely, the proof of Karp-Lipton proceeds by fixing the outer, universally quantified, proof of a Π_2^p machine, and applying the $\mathrm{NP} \subseteq \mathrm{P}_{/\mathrm{poly}}$ hypothesis to the resulting NP computation. However, for $\mathrm{QC}\Pi_2$, it is not clear that fixing the outer, universally quantified, proof yields a QCMA computation; thus, it is not obvious how to use the hypothesis $\mathrm{QCMA} \subseteq \mathrm{BQP}_{/\mathrm{mpoly}}$.

To sidestep this, our approach is to strengthen the hypothesis. Specifically, using the results of [20] on perfect completeness for QCMA, fixing the outer proof of a $\mathrm{QC}\Pi_2$ computation can be seen to yield a Precise-QCMA "decision problem", where by "decision problem", we mean no proofs for the Precise-QCMA verifier are accepted within the promise gap. Here, Precise-QCMA is QCMA with exponentially small promise gap. We hence obtain:

▶ **Theorem 8** (A quantum-classical Karp-Lipton theorem)**.** *If* $\text{Precise-QCMA} \subseteq \mathrm{BQP}_{/\mathrm{mpoly}}$, *then* $\mathrm{QC}\Pi_2 = \mathrm{QC}\Sigma_2$.

To give this result context, we also show that $\text{Precise-QCMA} \subseteq \mathrm{NP}^{\mathrm{PP}}$ (Lemma 38). However, whether $\mathrm{QC}\Pi_2 = \mathrm{QC}\Sigma_2$ collapses QCPH remains open due to the same "$\exists \cdot \mathrm{BPP}$ versus MA phenomenon".

1.2 Related work

The first work we are aware of which considered a quantum version of PH is that of Yamakami [36], which differs from our setting in that it considers quantum Turing machines (we use quantum circuits) and quantum inputs (we use classical inputs, just like QMA). Gharibian and Kempe [14] next introduced and studied cq-Σ_2, defined as our $\mathrm{QC}\Sigma_2$ except with a quantum universally quantified proof. [14] showed completeness and hardness of approximation results for cq-Σ_2 for (roughly) the following problem: What is the smallest number of terms required in a given local Hamiltonian for it to have a frustrated ground space? More recently, Lockhart and González-Guillén [25] considered a hierarchy (denoted QCPH' here) which *a priori* appears identical to our QCPH, but is apparently not so due to the "$\exists \cdot \mathrm{BPP}$ versus MA phenomenon", which we discuss below.

In this work, the "$\exists \cdot \mathrm{BPP}$ versus MA phenomenon", refers to the following discrepancy (see Remark 17 for details) – unlike with MA, *all* proofs in an $\exists \cdot \mathrm{BPP}$ system *must* be accepted with probability at least 2/3 or at most 1/3 (i.e. no proof is accepted with probability in the gap $(1/3, 2/3)$). The quantum analogue of this phenomenon yields the open question: Is $\exists \cdot \mathrm{BQP} = \mathrm{NP}^{\mathrm{BQP}}$ equal to QCMA? For this reason, it is not clear whether QCPH equals QCPH'. The latter is defined as $\mathrm{QC}\Sigma_1' = \exists \cdot \mathrm{BQP}$, $\mathrm{QC}\Pi_1' = \forall \cdot \mathrm{BQP}$, and

$$\forall m \geq 1, \mathrm{QC}\Sigma_m' = \exists \cdot \mathrm{QC}\Pi_{m-1}'; \qquad \mathrm{QC}\Pi_m' = \forall \cdot \mathrm{QC}\Sigma_{m-1}'.$$

Clearly, for us $\mathrm{QC}\Sigma_1 = \mathrm{QCMA}$ but in [25] $\mathrm{QC}\Sigma_1' = \exists \cdot \mathrm{BQP}$. The benefit from the latter definition is that one avoids the recursion problems discussed earlier – e.g., fixing the first existential proof in $\mathrm{QC}\Sigma_2'$ *does* reduce the problem to a co-QCMA computation, unlike the case with $\mathrm{QC}\Sigma_2$. Hence, recursive arguments from the context of PH can be easily extended to show that, for instance, QCPH' collapses to $\mathrm{QC}\Sigma_2'$ when $\mathrm{QC}\Sigma_2' = \mathrm{QC}\Pi_2'$. On the other hand, the advantage of our definition of QCPH is that it generalizes a natural quantum complexity class like QCMA.

Let us also remark on Toda's theorem in the context of QCPH′ (for clarity, Toda's theorem is not studied in [25]). The recursive definition of QCPH′ allows one to obtain Toda's P^{PP} upper bound for QCPH′ with a simple argument:

$$\forall i,\ \mathrm{QC}\Sigma_i' = \mathrm{NP}^{\mathrm{NP}^{\cdot^{\cdot^{\cdot^{\mathrm{BQP}}}}}} = \Sigma_i^{\mathrm{BQP}} \quad \Longrightarrow \quad \forall i, \mathrm{QC}\Sigma_i' \subseteq (\mathrm{P}^{\mathrm{PP}})^{\mathrm{BQP}} = \mathrm{P}^{\mathrm{PP}},$$

where the first equality expressly holds due to the recursive definition of $\mathrm{QC}\Sigma_i'$ but is not known to hold for our $\mathrm{QC}\Sigma_i$; the implication arises by relativizing Toda's theorem; and the last equality holds as BQP is low for PP [13]. In contrast, our Theorem 2 yields $\mathrm{QCPH} \subseteq \mathrm{P}^{\mathrm{PP}^{\mathrm{PP}}}$, raising the question: is QCPH′ = QCPH? A positive answer may help shed light on whether $\exists \cdot \mathrm{BQP}$ equals QCMA; we leave this for future work.

Finally, a quantum version of the Karp-Lipton theorem was covered by Aaronson and Drucker in [3] and further improved by Aaronson, Cojocaru, Gheorghiu, and Kashefi [2], where the consequences of NP-complete problems being solved by small quantum circuits with polynomial sized quantum advice were considered. Their results differ from ours in that different hierarchies are studied, and in their use of quantum advice as opposed to our use of classical advice. Other Karp-Lipton style results for PH involving classes beyond NP show a collapse of PH to MA (usually) if either PP [26, 32], $\mathrm{P}^{\#\mathrm{P}}$ or PSPACE [21] has $\mathrm{P}_{/\mathrm{poly}}$ circuits.

1.3 Open questions

As the study of quantum generalizations of alternating quantifiers is in its infancy, many open questions exist. For example, due to the "$\exists \cdot \mathrm{BPP}$ versus MA phenomenon", we are not able to show "simple" collapse statements such as the following:

▶ **Conjecture 9.** *For $i \geq 1$, if $\mathrm{QC}\Sigma_i = \mathrm{QC}\Pi_i$ for any i, then* QCPH *collapses to the i^{th} level. Moreover, if* QCMA = BQP*, then* QCPH = BQP*.*

Next, can a non-trivial bound on QPH be shown? Here, we have shown that $\mathrm{Q}\Sigma_3 \subseteq \mathrm{NEXP}$; can the complexity of higher levels be bounded? Along these lines, our Theorem 4 shows $\mathrm{Q}\Sigma_2 \subseteq \mathrm{EXP}$; by applying alternative methods for approximating semidefinite programs arising in quantum complexity theory (see, e.g., [19]), we might also conjecture:

▶ **Conjecture 10.** $\mathrm{Q}\Sigma_2 \subseteq \mathrm{PSPACE}$.

Determining where in the complexity zoo QMA(2) lies remains an important open question; assuming alternating quantifiers *do* add proving power to QPH (the analogous assumption for PH is widely believed), our work shows QMA(2) is strictly contained in NEXP. Can this statement be strengthened?

Finally, we remark on defining a hierarchy similar to QCPH, termed MA-PH, where the first level is MA instead of QCMA and the verifier in equations (1) and (2) will be a BPP circuit. Due to the promise nature of the BPP verifier, we conjecture that the same issues faced with QCPH will translate to MA-PH too. Also, as Precise-BPP is equivalent to PP, we can obtain a similar Cleaning Lemma for MA-PH too. Hence, we conjecture that

▶ **Conjecture 11.** $\mathrm{PH} \subseteq \mathrm{MA\text{-}PH} \subseteq \mathrm{QCPH} \subseteq \mathrm{P}^{\mathrm{PP}^{\mathrm{PP}}}$.

Using other techniques that may harness the fact that BPP and MA are contained in PH to obtain a better bound for MA-PH is an interesting open question.

Organization. We begin in Section 2 by showing a quantum-classical analogue of Toda's theorem. Section 3 gives upper bounds on levels of QPH, and Section 4 shows a Karp-Lipton-type theorem. Formal definitions and many proofs are omitted from this version of the paper owing to space constraints.

2 A quantum-classical analogue of Toda's theorem

2.1 Precise-BQP

Our proof of a "quantum-classical Toda's theorem" requires us to define the Precise-BQP class, which we do now. Below, a promise problem is a pair $A = (A_{\text{yes}}, A_{\text{no}})$ such that $A_{\text{yes}}, A_{\text{no}} \subseteq \{0,1\}^*$, $A_{\text{yes}} \cup A_{\text{no}} \subset \{0,1\}^*$ and $A_{\text{yes}} \cap A_{\text{no}} = \emptyset$.

▶ **Definition 12** (Precise-BQP(c,s))**.** A promise problem $A = (A_{\text{yes}}, A_{\text{no}})$ is contained in Precise-BQP(c,s) for polynomial-time computable functions $c, s : \mathbb{N} \mapsto [0,1]$ if there exists a polynomially bounded function $p : \mathbb{N} \mapsto \mathbb{N}$ such that $\forall \ell \in \mathbb{N},\ c(\ell) - s(\ell) \geq 2^{-p(\ell)}$ and a polynomial-time uniform family of quantum circuits $\{V_n\}_{n\in\mathbb{N}}$ whose input is the all zeroes state and output is a single qubit. Furthermore, for an n-bit input x:

- Completeness: If $x \in A_{\text{yes}}$, then V_n accepts with probability at least c.
- Soundness: If $x \in A_{\text{no}}$, then V_n accepts with probability at most s.

In contrast, BQP is defined such that the completeness and soundness parameters are $2/3$ and $1/3$, respectively (alternatively, the gap is least an inverse polynomial in n).

▶ **Observation 13** (Rational acceptance probabilities)**.** *By fixing an appropriate universal gate set (e.g. Hadamard and Toffoli [4]) for the description of V_n in Definition 12, we assume henceforth, without loss of generality, that the acceptance probability of V_n is a rational number that can be represented using at most* $\text{poly}(n)$ *bits (this observation was used in the proof that* QCMA *has perfect completeness i.e., $c = 1$ [20].).*

The following help to characterize the complexity of Precise-BQP.

▶ **Lemma 14.** *For all $c, s \in [0,1]$ and every n-bit input such that $c - s \in \Omega(1/\exp(n))$,* Precise-BQP$(c,s) \subseteq$ PP*.*

▶ **Corollary 15.** *Let $\mathbb{P}$ denote the set of all polynomials $p : \mathbb{N} \mapsto \mathbb{N}$. Then,*

$$\bigcup_{p\in\mathbb{P}} \text{Precise-BQP}\left(\frac{1}{2} + \frac{1}{2^{p(n)}}, \frac{1}{2}\right) = \text{PP}.$$

2.2 Bounding the power of QCPH

Classically, PH can be defined in terms of the existential and universal operators; while it is not clear that one can also define QCPH using these operators, they nevertheless prove useful in bounding the power of QCPH.

▶ **Definition 16** (Existential and universal quantifiers [35, 7])**.** For $\mathcal{C}$ a class of languages, $\exists \cdot \mathcal{C}$ is defined as the set of languages L such that there is a polynomial p and set $A \in \mathcal{C}$ such that for input x, $x \in L \Leftrightarrow [\exists y\,(|y| \leq p(|x|))$ and $\langle x, y\rangle \in A]$. The set $\forall \cdot \mathcal{C}$ is defined similarly with $\exists$ replaced with $\forall$.

▶ **Remark 17** (Languages versus promise problems)**.** *Directly extending Definition 16 to promise problems, gives rise to subtle issues. To demonstrate, recall that* $\exists \cdot \text{P} = \text{NP}$*. Then,*

let (L, A) *for* $L \in \exists \cdot \mathrm{P} = \mathrm{NP}$ *and* $A \in \mathrm{P}$ *be as in Definition 16, such that* T_A *is a polynomial-time Turing machine deciding* A*. If* $x \in L$*, there exists a bounded length witness* y^* *such that* T_A *accepts* $\langle x, y^* \rangle$ *and, for all* $y' \neq y^*$*,* T_A *by definition either accepts or rejects* $\langle x, y' \rangle$*. Now consider instead* $\exists \cdot \mathrm{BPP}$*, which* a priori *seems equal to Merlin-Arthur (*MA*). Applying the same definition of* $\exists$*, we should obtain a* BPP *machine* T_A *such that if* $x \in L$*, then for all* $y' \neq y^*$*,* T_A either accepts or rejects $\langle x, y' \rangle$*. But this means, by definition of* BPP*, that* $\langle x, y' \rangle$ *is either accepted or rejected with probability at least* $2/3$*, respectively. (Equivalently, for any fixed* y*, the machine* $T_{A,y}$ *must be a* BPP *machine, or more generally a machine with the resources available to class* $\mathcal{C}$*.) Unfortunately, the definition of* MA makes no such promise – *any* $y' \neq y^*$ *can be accepted with arbitrary probability when* x *is a* YES *instance. Indeed, whether* $\exists \cdot \mathrm{BPP} = \mathrm{MA}$ *remains an open question [11].*

The following lemma is the main contribution of this section. To set context, adapting the ideas from Toda's proof of $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{PP}}$ to QCPH is problematic for at least two reasons:

1. Remark 17 says that it is not necessarily true that by fixing a proof y to an MA (resp. QCMA) machine, the resulting machine is a BPP (resp. BQP) machine. This prevents the direct extension of recursive arguments, say from [30] to this regime.
2. The "Quantum Valiant Vazirani (QVV)" theorem for QCMA (and MA) [5] is not a many-one reduction, but a *Turing* reduction. Specifically, it produces a set of quantum circuits $\{ Q_i \}$, *at least one of which* is guaranteed to be a YES instance of some Unique-QCMA promise problem Γ if the input Π to the reduction was a YES instance. Unfortunately, some of the Q_i may violate the promise gap of Γ, which implies that when such Q_i are substituted into the Unique-QCMA oracle O, O returns an arbitrary answer. This does not pose a problem in [5], as one-sided error suffices for that reduction – so long as O accepts at least one Q_i, one safely concludes Π was a YES instance. In the setting of Toda's theorem, however, the use of *alternating* quantifiers turns this one-sided error into two-sided error; this renders the output of O useless, as one can no longer determine whether Π was a YES or NO instance.

To sidestep these issues, we adapt a high-level idea from [15]: With the help of an appropriate *oracle*, one can sometimes detect "invalid proofs" (i.e. proofs in promise gaps of bounded error verifiers) and "remove" them. Indeed, we show that using a PP oracle, one can eliminate the promise-gap of QCPH altogether, thus overcoming the limitations given above. This is accomplished by the following "Cleaning Lemma".

▶ **Lemma 18** (Cleaning Lemma). *For all* $i \geq 0$*,* $\mathrm{QC}\Sigma_i \subseteq \exists \cdot \forall \cdot \cdots \cdot Q_i \cdot \mathrm{P}^{\text{Precise-BQP}} \subseteq \exists \cdot \forall \cdot \cdots \cdot Q_i \cdot \mathrm{P}^{\mathrm{PP}}$*, where* $Q_i = \exists$ *(*$Q_i = \forall$*) if* i *is odd (even). An analogous statement holds for* $\mathrm{QC}\Pi_i$*.*

Proof. Let C be a $\mathrm{QC}\Sigma_i$ verification circuit for a promise problem Π. Let $C_{y_1^*,\ldots,y_i^*}$ denote the quantum circuit obtained from C by fixing values $y_1^*, \ldots, y_i^*$ of the i classical proofs. In general, nothing can be said about the acceptance probability $p_{y_1^*,\ldots,y_i^*}$ of $C_{y_1^*,\ldots,y_i^*}$, except that, by Observation 13, $p_{y_1^*,\ldots,y_i^*}$ is a rational number representable using $p(n)$ bits for some fixed polynomial p. Let S denote the set of all rational numbers in $[0, 1]$ representable using $p(n)$ bits of precision. (Note $|S| \in \Theta(2^{p(n)})$.) Then, for any $a, b \in S$ with $a > b$, the triple $(C_{y_1^*,\ldots,y_i^*}, a, b)$ is a valid QCIRCUIT(a, b) instance, i.e. $C_{y_1^*,\ldots,y_i^*}$ accepts with probability at least a or at most b for $a - b$ an inverse exponential. It follows that using binary search (by varying the values $a, b \in S$ with $a > b$) in conjunction with $\mathrm{poly}(n)$ calls to a QCIRCUIT(a, b) oracle, we may exactly and deterministically compute $p_{y_1^*,\ldots,y_i^*}$. Moreover, since for all such $a > b$, QCIRCUIT$(a, b) \in$ Precise-BQP(a, b), Lemma 14 implies a QCIRCUIT(a, b) oracle call can be simulated with a PP oracle. Denote the binary search subroutine using the PP oracle as B.

Using C and B, we now construct an oracle Turing machine C' as follows. Given any proofs $y_1^*, \ldots, y_i^*$ as input, C' uses B to compute $p_{y_1^*,\ldots,y_i^*}$ for $C_{y_1^*,\ldots,y_i^*}$. If $p_{y_1^*,\ldots,y_i^*} \geq c$, C' accepts with certainty, and if $p_{y_1^*,\ldots,y_i^*} < c$, C' rejects with certainty. Suppose that the circuits C and C' return 1 when they accept and 0 when they reject. Two observations: (1) Since by construction, for any fixed $y_1^*, \ldots, y_i^*$, B makes only makes "valid" QCIRCUIT(a, b) queries (i.e. satisfying the promise of QCIRCUIT(a, b)), C' is a P^{PP} machine (cf. Observation 20). (2) Since $C'_{y_1^*,\ldots,y_i^*}$ accepts if $C_{y_1^*,\ldots,y_i^*}$ accepts with probability at least c, and since $C'_{y_1^*,\ldots,y_i^*}$ rejects if $C_{y_1^*,\ldots,y_i^*}$ accepts with probability at most s, we conclude that

$$\exists y_1 \forall y_2 \cdots Q_i y_i \, \mathrm{Prob}[C(y_1, \ldots, y_i) = 1] \geq c \quad \Leftrightarrow \quad \exists y_1 \forall y_2 \cdots Q_i y_i \; C'(y_1, \ldots, y_i) = 1 \quad (4)$$

$$\forall y_1 \exists y_2 \cdots \overline{Q}_i y_i \, \mathrm{Prob}[C(y_1, \ldots, y_i) = 1] \leq s \quad \Leftrightarrow \quad \forall y_1 \exists y_2 \cdots \overline{Q}_i y_i \; C'(y_1, \ldots, y_i) = 0. \quad (5)$$

(4) and (5) imply that we can simulate Π with a $\exists \cdot \forall \cdot \cdots \cdot Q_i \cdot \mathrm{P}^{\mathrm{PP}}$ computation. The proof for $\mathrm{QC}\Pi_i$ is analogous. ◀

▶ **Remark 19** (Possibility of a stronger containment). *A key question is whether one may replace the* Precise-BQP *oracle in the proof of Lemma 18 with a weaker* BQP *oracle. For example, consider the following alternate definition for oracle Turing machine C': Given proofs $y_1^*, \ldots, y_i^*$, C' plugs $C_{y_1^*,\ldots,y_i^*}$ into a BQP oracle and returns the oracle's answers. It is easy to see that in this case, Equations (4) and (5) hold. However, C' is* not *necessarily a $\mathrm{P}^{\mathrm{BQP}}$ machine, since for some settings of $y_1^*, \ldots, y_i^*$, its input to the* BQP *oracle may violate the BQP promise, hence making the output of C' ill-defined. To further illustrate this subtle point, consider Observation 20. Moreover, in Section 2.3 we show that the task the* Precise-BQP *oracle is used for in Lemma 18 is in fact PP-complete; thus, it is highly unlikely that one can substitute a weaker oracle into the proof above.*

▶ **Observation 20** (When a P machine querying a BQP oracle is not a $\mathsf{P}^{\mathsf{BQP}}$ machine). *The proof of the Cleaning Lemma uses a $\mathrm{P}^{\text{Precise-BQP}}$ machine. Let us highlight a subtle reason why using a weaker* BQP *oracle instead might be difficult (indeed, in Section 2.3 we show that the task we use the* Precise-BQP *oracle for is* PP*-complete). Let M denote the trivially* BQP*-complete problem of determining whether a given polynomial-sized quantum circuit Q accepts with probability at least $2/3$, or accepts with probability at most $1/3$, with the promise that one of the two is the case. Now consider the following polynomial time computation, Π, which is given access to an oracle O_M for M: Π inputs the Hadamard gate H into O_M and outputs O_M's answer. Does it hold that $\Pi \in \mathrm{P}^{\mathrm{BQP}}$? No. Since H violates the promise of BQP, i.e. measuring the output of H yields 0 or 1 with equal probability, the oracle O_M can answer 0 or 1 arbitrarily, and so the output of Π is not well-defined. Having a well-defined output, however, is required for a P^{O_K} computation, where K is any promise class [16].*

▶ **Lemma 21.** *For all $i \geq 0$, the following holds true: $\exists \cdot \forall \cdot \cdots \cdot Q_i \cdot \mathrm{P}^{\mathrm{PP}} \subseteq \Sigma_i^{\mathrm{PP}}$ and $\forall \cdot \exists \cdot \cdots \cdot Q_i \cdot \mathrm{P}^{\mathrm{PP}} \subseteq \Pi_i^{\mathrm{PP}}$ where $Q_i = \exists$ (resp. $Q_i = \forall$) when i is odd (resp. even) in the first containment and vica-versa for the second containment.*

We can now show the main theorem of this section.

▶ **Theorem 22.** $\mathrm{QCPH} \subseteq \mathrm{P}^{\mathrm{PP}^{\mathrm{PP}}}$.

Proof. The claim follows by combining the Cleaning Lemma (Lemma 18), Lemma 21, and Toda's theorem ($\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{PP}}$), whose proof relativizes (see, e.g., page 4 of [12])). ◀

2.3 Detecting non-empty promise gaps is PP-complete

The technique behind the Cleaning Lemma (Lemma 18) can essentially be viewed as using a PP oracle to determine whether a given quantum circuit accepts some input with probability within the promise gap (s, c), where $c - s$ is an inverse polynomial. One can ask whether this rather powerful PP oracle can be replaced with a weaker oracle (see Remark 19)? We answer this in the negative unless one deviates from our specific proof approach; specifically, we show that the problem of detecting non-empty promise gaps is PP-complete, even if the gap is *constant* in size.

To begin, we define QCIRCUIT(c, s), which is trivially Precise-BQP(c, s)-complete when $c - s$ is an inverse exponential. (Take note that when the $c - s$ gap is larger, say inverse polynomial, QCIRCUIT(c, s) is still contained in Precise-BQP(c, s).)

▶ **Definition 23** (QCIRCUIT(c, s))**.** Parameters $c, s : \mathbb{N} \mapsto [0, 1]$ are polynomial-time computable functions such that $c > s$.

- (Input) A classical description of quantum circuit V_n (acting on n qubits, consisting of poly(n) 1 and 2-qubit gates), taking in the all-zeroes state, and outputting a single qubit.
- (Output) Decide if Pr$[V_n$ accepts $] \geq c$ or $\leq s$, assuming one of the two is the case.

▶ **Definition 24** (NON-EMPTY GAP(c, s))**.** Let V_n be an input for QCIRCUIT(c, s). Then, output YES if Prob$[V_n$ accepts $] \in (s, c)$, and NO otherwise.

We now show that NON-EMPTY GAP is PP-complete.

▶ **Lemma 25.** *For all c, s with the $c - s$ gap at least an inverse exponential in input size,* NON-EMPTY GAP$(c, s) \in$ PP*.*

▶ **Lemma 26.** *There exist $c, s \in \Theta(1)$ such that* NON-EMPTY GAP(c, s) *is* PP*-hard.*

▶ **Theorem 27.** *There exist $c, s \in \Theta(1)$ such that* NON-EMPTY GAP(c, s) *is* PP*-complete.*

3 Bounding the power of $\mathbf{Q\Sigma_2}$ and $\mathbf{Q\Sigma_3}$

Let $Q\Sigma_2(c, s)$ (resp., $Q\Pi_2(c, s)$) be defined as $Q\Sigma_2$ (resp., $Q\Pi_2$) with completeness and soundness parameters c and s, respectively. We begin by restating Theorem 4 as follows.

▶ **Theorem 28.** *For any polynomial r, if $c - s \geq 1/2^{2^{r(n)}}$, then $Q\Sigma_2(c, s) \subseteq$ EXP and $Q\Pi_2(c, s) \subseteq$ EXP when c and s are computable in exponential time in the size of the input.*

The two containments in Theorem 28 are proven separately in the following two lemmas.

▶ **Lemma 29.** *Let α be the maximum acceptance probability of a $Q\Sigma_2$ protocol (where the optimization is over the first proof ρ_1). Then one can compute γ such that $|\gamma - \alpha| \leq 1/2^{2^r}$, for any polynomial r, in exponential time.*

▶ **Lemma 30.** *Let α be the* minimum *acceptance probability of a $Q\Pi_2$ protocol (where the optimization is again over the first proof ρ_1). Then one can compute γ such that $|\gamma - \alpha| \leq 1/2^{2^r}$, for any polynomial r, in exponential time.*

We now sketch the exponential time protocol that calculates γ in Lemma 29 (we refer the reader to [17] for standard background in convex optimization). The proof of Lemma 30 is similar.

Proof Sketch. Recall from (3) that the maximum acceptance probability of a $\mathrm{QC}\Sigma_2$ protocol can be expressed as $\alpha := \max_{\rho_1} \min_{\rho_2} \langle C, \rho_1 \otimes \rho_2 \rangle$, where C is the POVM that corresponds to the quantum verification circuit in the $\mathrm{Q}\Sigma_2$ protocol accepting. We wish to decide in exponential time whether $\alpha \geq c$ or $\alpha \leq s$. Since the promise gap satisfies $c - s \geq 1/2^{2^{r(n)}}$, it suffices to approximate α within additive error (say) $\frac{1}{4}(c-s)$ by computing $\gamma \in \mathbb{R}$, in exponential time, such that $|\gamma - \alpha| \leq 1/(4 \cdot 2^{2^{r(n)}})$.

We begin by constructing C' as a numerical approximation to C such that each entry in C' is correct up to exponentially many bits. This can be done independent of the gate set used to describe the verification circuit, V_n, used for the $\mathrm{Q}\Sigma_2$ instance[6]. Then, for some polynomial r, $|\alpha - \alpha'| \leq \frac{1}{2} \cdot 2^{-2^{r(n)/2}}$ for

$$\alpha' := \max_{\rho_1} \min_{\rho_2} \{\langle C', \rho_1 \otimes \rho_2 \rangle : \mathrm{Tr}(\rho_1) = \mathrm{Tr}(\rho_2) = 1,\ \rho_1, \rho_2 \succeq 0\}. \tag{6}$$

Suppose we fix a feasible ρ_1 and solve the inner optimization problem in (6). Then:

$$\alpha'(\rho_1) := \min_{\rho_2} \{\langle C', \rho_1 \otimes \rho_2 \rangle : \mathrm{Tr}(\rho_2) = 1,\ \rho_2 \succeq 0\}.$$

We can rewrite $\langle C', \rho_1 \otimes \rho_2 \rangle$ as $\langle \mathrm{Tr}_1[(\rho_1 \otimes I)C'], \rho_2 \rangle$ where Tr_1 is the partial trace over the register that ρ_1 acts on. Additionally, as $\mathrm{Tr}_1[(\rho_1 \otimes I)C'] = \mathrm{Tr}_1[(\rho_1^{1/2} \otimes I)C(\rho_1^{1/2} \otimes I)]$, this term is Hermitian and positive semidefinite. This implies that the best choice for ρ_2 is a rank-1 projector onto the eigenspace corresponding to least eigenvalue. In other words, $\alpha'(\rho_1) = \lambda_{\min}(\mathrm{Tr}_1[(\rho_1 \otimes I)C'])$ where $\lambda_{\min}(X)$ denotes the least eigenvalue of a Hermitian operator X. For fixed ρ_1, this minimum eigenvalue calculation can be rephrased via the dual optimization program for $\alpha'(\rho_1)$,

$$\alpha'(\rho_1) = \max_{t} \{t : tI \preceq \mathrm{Tr}_1[(\rho_1 \otimes I)C']\}.$$

Re-introducing the maximization over ρ_1, we hence obtain

$$\alpha' = \max_{\rho_1, t} \{t : tI \preceq \mathrm{Tr}_1[(\rho_1 \otimes I)C'],\ \mathrm{Tr}(\rho_1) = 1,\ \rho_1 \succeq 0\}, \tag{7}$$

which is a semidefinite program. By using the ellpsoid method, we can hence solve this semidefinite program (see [17] for details) to obtain estimate γ of α'. Using an analysis similar to [34], we find a γ such that $|\gamma - \alpha'| \leq \epsilon$ with $\epsilon = 2^{-2^{r(n)}}$. ◀

Using the power of non-determinism, we can also bound the power of $\mathrm{Q}\Sigma_3$ and $\mathrm{Q}\Pi_3$.

▶ **Theorem 31.** *For any polynomial r and input size n, if $c - s \geq 1/r(n)$, then*

$$\mathrm{QMA}(2) \subseteq \mathrm{Q}\Sigma_3 \subseteq \mathrm{NEXP} \quad \textit{and} \quad \text{co-}\mathrm{QMA}(2) \subseteq \mathrm{Q}\Pi_3 \subseteq \text{co-}\mathrm{NEXP}, \tag{8}$$

where all classes have completeness and soundness c and s, respectively. Moreover, if we allow smaller gaps (in principle, gaps which are at most inverse singly exponential in n suffice for the first claim below), such as $c - s \geq 1/2^{2^{r(n)}}$, then

$$\mathrm{QMA}(2)(c,s) = \mathrm{Q}\Sigma_3(c,s) = \mathrm{NEXP} \quad \textit{and} \quad \text{co-}\mathrm{QMA}(2)(c,s) = \mathrm{Q}\Pi_3(c,s) = \text{co-}\mathrm{NEXP}. \tag{9}$$

Here, we assume c and s are computable in exponential time in the size of the input.

---

[6] This can be accomplished in exponential time as follows: Replace gate set G with G' by approximating each entry of each gate in G using $2^{s(n)}$ bits of precision, for some sufficiently large, fixed polynomial s. Define C' as C, except each use of a gate $U \in G$ is replaced with its approximation $U' \in G'$. Then, via the well-known bound $\|U_m \cdots U_1 - V_m \cdots V_1\|_\infty \leq \sum_{i=1}^m \|U_i - V_i\|_\infty$ (for unitary U_i, V_i), it follows that $\left\|C' - C\right\|_\infty \in O(\mathrm{poly}(n)/(2^{2^{s(n)}}))$, since V_n contains $\mathrm{poly}(n)$ gates. Here, $\|A\|_\infty = \max_{|\psi\rangle} \|A|\psi\rangle\|_2$ for unit vectors $|\psi\rangle$ denotes the spectral or operator norm. Finally, apply the fact that $\max_{i,j} |A(i,j)| \leq \|A\|_\infty$ (p. 314 of [18]).

4 Karp-Lipton type theorems

The Karp-Lipton [21] theorem showed that if NP $\subseteq$ P$_{/\text{poly}}$ (i.e. if NP can be solved by polynomial-size non-uniform circuits), then $\Sigma_2 = \Pi_2$ (which in turn collapses PH collapses to its second level). Then, building on the conjecture that the polynomial hierarchy is infinite, this result implies that NP $\not\subset$ P$_{/\text{poly}}$ (a stronger claim than P $\neq$ NP as P $\subseteq$ P$_{/\text{poly}}$). Some attempts to separate NP from P use this as a basis to try and prove the stronger claim instead. For instance, this has lead to the approach of proving super-polynomial circuit lower bounds for circuits of NP-complete problems. Here, we show that the proof technique of Karp and Lipton carries over easily to the quantum setting, *provided* one uses the stronger hypothesis Precise-QCMA $\subseteq$ BQP$_{/\text{mpoly}}$ (as opposed to QCMA $\subseteq$ BQP$_{/\text{mpoly}}$). Whether this causes QCPH to collapse to its second level, however, remains open (see Remark 37 below). We begin by formally defining the classes BQP$_{/\text{mpoly}}$ and Precise-QCMA.

▶ **Definition 32** (BQP$_{/\text{mpoly}}$)**.** A promise problem $\Pi = (A_{\text{yes}}, A_{\text{no}})$ is in BQP$_{/\text{mpoly}}$ if there exists a polynomial-sized family of quantum circuits $\{C_n\}_{n\in\mathbb{N}}$ and a collection of binary advice strings $\{a_n\}_{n\in\mathbb{N}}$ with $|a_n| = \text{poly}(n)$, such that for all n and all strings x where $|x| = n$, $\Pr[C_n(|x\rangle, |a_n\rangle) = 1] \geq 2/3$ if $x \in A_{\text{yes}}$ and $\Pr[C_n(|x\rangle, |a_n\rangle) = 1] \leq 1/3$ if $x \in A_{\text{no}}$.

Equivalently, BQP$_{/\text{mpoly}}$ is the set of promise problems solvable by a *non-uniform* family of polynomial-sized bounded error quantum circuits. It is used as a quantum analogue for P$_{/\text{poly}}$ in this scenario. Here, we remark on the use of mpoly instead of poly in Definition 32. Note that BQP$_{/\text{poly}}$ accepts Karp-Lipton style advice i.e. it is a BQP circuit that accepts a poly-sized advice string to provide *some answer* with probability at least 2/3 even if the "advice is bad". On the other hand, BQP$_{/\text{mpoly}}$ accepts Merlin style advice i.e. it is a BQP circuit accepting poly-sized classical advice such that the output is correct with probability at least 2/3 if the "advice is good". Note BQP$_{/\text{poly}}$ versus BQP$_{/\text{mpoly}}$ is analogous to the "$\exists \cdot$BPP versus MA" phenomenon. Moreover, as we are concerned with variations of QCMA, and not $\exists \cdot$BQP, BQP$_{/\text{mpoly}}$ is the right candidate for us.

▶ **Definition 33** (Precise-QCMA)**.** A promise problem $A = (A_{\text{yes}}, A_{\text{no}})$ is said to be in Precise-QCMA(c, s) for polynomial-time computable functions $c, s : \mathbb{N} \mapsto [0, 1]$ if there exists polynomially bounded functions $p, q : \mathbb{N} \mapsto \mathbb{N}$ such that $\forall \ell \in \mathbb{N}$, $c(\ell) - s(\ell) \geq 2^{-q(\ell)}$, and there exists a polynomial-time uniform family of quantum circuits $\{V_n\}_{n\in\mathbb{N}}$ that takes a classical proof $y \in \{0, 1\}^{p(n)}$ and outputs a single qubit. Moreover, for an n-bit input x:

- Completeness: If $x \in A_{\text{yes}}$, then $\exists\ y$ such that V_n accepts y with probability at least c.
- Soundness: If $x \in A_{\text{no}}$, then $\forall\ y$, V_x accepts y with probability at most s.

Define Precise-QCMA $= \bigcup_{c,s}$ Precise-QCMA(c, s).

As an aside, note that QCMA is defined with $c - s \in \Omega(1/\text{poly}(n))$. Recall from the discussion in Section 1.1 that the main obstacle to the recursive arguments that work well for NP in [21] is the "promise problem" nature of QCΠ_2 and QCMA. However, exploiting the perfect completeness of Precise-QCMA[7] and the fact that $\forall c < s' \leq s$, Precise-QCMA$(c, s) \subseteq$ Precise-QCMA(c, s'), we "recover" the notion of a decision problem in a rigorous sense by working with Precise-QCMA as demonstrated below.

[7] The perfect completeness proof for QCMA also works in the inverse exponentially small gap setting [20].

▶ **Claim 34.** *For every promise problem* $\Pi' = (A_{\text{yes}}, A_{\text{no}}) \in \text{Precise-QCMA}(c, s)$ *with verifier* V'*, there exists a verifier* V *(a poly-time uniform quantum circuit family), a polynomial* q *and a decision problem* $\Pi = (A_{\text{yes}}, \{0,1\}^* \setminus A_{\text{yes}})$ *such that* $\Pi \in \text{Precise-QCMA}(1, 1-2^{-q(n)})$ *with verifier* V*. Moreover, for all proofs* y*,* V *accepts* y *with probability either* 1 *or at most* $1-2^{-q(n)}$*.*

Building on this "decision problem" flavour of Precise-QCMA, we first show:

▶ **Lemma 35.** *Suppose* $\text{Precise-QCMA} \subseteq \text{BQP}_{/\text{mpoly}}$*. Then, for every promise problem* $\Pi = (A_{\text{yes}}, A_{\text{no}})$ *in* Precise-QCMA *and every* n*-bit input* x*, there exists a polynomially bounded function* $p : \mathbb{N} \mapsto \mathbb{N}$ *and a bounded error polynomial time non-uniform quantum circuit family* $\{C_n\}_{n\in\mathbb{N}}$ *such that:*

- *if* $x \in A_{\text{yes}}$*, then* C_n *outputs valid proof* $y \in \{0,1\}^{p(n)}$ *such that* (x, y) *is accepted by the corresponding* Precise-QCMA *verifier with probability* 1*;*
- *if* $x \in A_{\text{no}}$*, then* C_n *outputs a symbol* $\perp$ *with probability exponentially close to* 1 *signifying that there is no* $y \in \{0,1\}^{p(n)}$*, such that* (x, y) *is accepted by the corresponding* Precise-QCMA *verifier with probability* 1*.*

We next give a quantum-classical analogue of the Karp-Lipton theorem, whose proof is in the appendix.

▶ **Theorem 36** (A Quantum-Classical Karp-Lipton Theorem)**.** *If* $\text{Precise-QCMA} \subseteq \text{BQP}_{/\text{mpoly}}$ *then* $\text{QC}\Pi_2 = \text{QC}\Sigma_2$*.*

▶ **Remark 37** (Collapse of QCPH?)**.** *An appeal of the classical Karp-Lipton theorem is that it implies that if* $\text{NP} \subseteq \text{P}_{/\text{poly}}$*, then PH collapses to its second level; this is because if* $\Pi_2^p = \Sigma_2^p$*, then* PH *collapses to* Σ_2^p*. Does an analogous statement hold for* QCPH *as a result of Theorem 8? Unfortunately, the answer is not clear. The problem is similar to that outlined in Remark 17. Namely, classically* $\Pi_2^p = \Sigma_2^p$ *collapses* PH *since for any* Π_3^p *decision problem, fixing the first (universally) quantified proof yields a* Σ_2^p *computation. But this can be replaced with a* Π_2^p *computation by assumption, yielding a computation with quantifiers* $\forall\forall\exists$*, which trivially collapses to* $\forall\exists$*, i.e.* $\Pi_3^p \subseteq \Pi_2^p$*. In contrast, for (say)* $\text{QC}\Pi_3$*, similar to the phenomenon in Remark 17, fixing the first (universally) quantified proof does* not *necessarily yield a* $\text{QC}\Sigma_2^p$ *computation. Thus, a recursive application of the assumption* $\text{QC}\Sigma_2^p = \text{QC}\Pi_2^p$ *cannot straightforwardly be applied.*

Since Precise-QCMA plays an important role in Theorem 8, we close with an upper bound on Precise-QCMA.

▶ **Lemma 38.** $\text{Precise-QCMA} \subseteq \text{NP}^{\text{PP}}$*.*

Proof. Let V be a Precise-QCMA verifier. Using Claim 34, we may assume that for any proof y, V either accepts y with probability 1 or rejects with probability at most $1-2^{-q(n)}$. Thus, for any fixed y, the resulting computation V_y is a Precise-BQP computation. This implies $\text{Precise-QCMA} \subseteq \exists \cdot \text{Precise-BQP}$ (see also Remark 17). But by Definition 16, $\exists \cdot \text{Precise-BQP} \subseteq \text{NP}^{\text{Precise-BQP}}$. Combining this with Lemma 14, which says that $\text{Precise-BQP} \subseteq \text{PP}$, yields the claim. ◀

References

1 S. Aaronson, S. Beigi, A. Drucker, B. Fefferman, and P. Shor. The power of unentanglement. *Theory of Computing*, 5:1–42, 2009. doi:10.4086/toc.2009.v005a001.

2 S. Aaronson, A. Cojocaru, A. Gheorghiu, and E. Kashefi. On the implausibility of classical client blind quantum computing. Available at arXiv.org e-Print quant-ph/1704.08482, 2017.

3 S. Aaronson and A. Drucker. A full characterization of quantum advice. *SIAM Journal on Computing*, 43(3):1131–1183, 2014.

4 D. Aharonov. A simple proof that Toffoli and Hadamard are quantum universal. Available at arXiv.org e-Print quant-ph/0301040, jan 2003. arXiv:quant-ph/0301040.

5 D. Aharonov, M. Ben-Or, F. Brandão, and O. Sattath. The pursuit for uniqueness: Extending Valiant-Vazirani theorem to the probabilistic and quantum settings. Available at arXiv.org e-Print quant-ph/0810.4840v1, 2008.

6 D. Aharonov and T. Naveh. Quantum NP - A survey. Available at arXiv.org e-Print quant-ph/0210077v1, 2002.

7 E. W. Allender and K. W. Wagner. *Counting hierarchies: Polynomial time and constant depth circuits*, pages 469–483. World Scientific, 1993. doi:10.1142/9789812794499_0035.

8 A. Ambainis. On physical problems that are slightly more difficult than QMA. In *Proceedings of 29th IEEE Conference on Computational Complexity (CCC 2014)*, pages 32–43, 2014.

9 S. Beigi. NP vs $\mathrm{QMA}_{\log}(2)$. *Quantum Information and Computation*, 10:0141–0151, 2010.

10 H. Blier and A. Tapp. All languages in NP have very short quantum proofs. In *Proceedings of the 3rd International Conference on Quantum, Nano and Micro Technologies*, pages 34–37, 2009.

11 S. Fenner, L. Fortnow, S. A. Kurtz, and L. Li. An oracle builder's toolkit. *Information and Computation*, 182(2):95–136, 2003. doi:10.1016/S0890-5401(03)00018-X.

12 L. Fortnow. The role of relativization in complexity theory. *Bulletin of the European Association for Theoretical Computer Science*, 52:52–229, 1994.

13 L. Fortnow and J. Rogers. Complexity limitations on quantum computation. *Journal of Computer and System Sciences*, 59(2):240–252, 1999.

14 S. Gharibian and J. Kempe. Hardness of approximation for quantum problems. In *Proceedings of 39th International Colloquium on Automata, Languages and Programming (ICALP 2012)*, pages 387–398, 2012.

15 S. Gharibian and J. Yirka. The complexity of simulating local measurements on quantum systems. In Mark M. Wilde, editor, *12th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2017)*, volume 73 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.TQC.2017.2.

16 O. Goldreich. On promise problems: A survey. *Theoretical Computer Science*, 3895:254–290, 2006.

17 M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1993.

18 R. A. Horn and C. H. Johnson. *Matrix Analysis*. Cambridge University Press, 1990.

19 R. Jain, Z. Ji, S. Upadhyay, and J. Watrous. QIP = PSPACE. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing*, pages 573–581, 2010.

20 S. P. Jordan, H. Kobayashi, D. Nagaj, and H. Nishimura. Achieving perfect completeness in classical-witness quantum Merlin-Arthur proof systems. *Quantum Information & Computation*, 12(5 & 6):461–471, 2012.

21 R. M. Karp and R. J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC '80, pages 302–309, New York, NY, USA, 1980. ACM. doi:10.1145/800141.804678.

22 A. Kitaev, A. Shen, and M. Vyalyi. *Classical and Quantum Computation*. American Mathematical Society, 2002.

23 A. Kitaev and J. Watrous. Parallelization, amplification, and exponential time simulation of quantum interactive proof systems. In *Proceedings of the 32nd ACM Symposium on Theory of Computing (STOC 2000)*, pages 608–617, 2000.

24 Y.-K. Liu, M. Christandl, and F. Verstraete. Quantum computational complexity of the N-representability problem: QMA complete. *Physical Review Letters*, 98:110503, 2007.

25 J. Lockhart and C. E. González-Guillén. Quantum state isomorphism. *arXiv preprint arXiv:1709.09622*, 2017.

26 C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992. doi:10.1145/146585.146605.

27 C. Marriott and J. Watrous. Quantum Arthur-Merlin games. *Computational Complexity*, 14(2):122–152, 2005.

28 A. Meyer and L. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proceedings of the 13th Symposium on Foundations of Computer Science*, pages 125–129, 1972.

29 A. Pereszlényi. Multi-prover quantum Merlin-Arthur proof systems with small gap. Available at arXiv.org e-Print quant-ph/1205.2761, 2012.

30 S. Toda. PP is as hard as the Polynomial-Time Hierarchy. *SIAM Journal on Computing*, 20:865–877, 1991.

31 L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

32 N. V. Vinodchandran. A note on the circuit complexity of pp. *TCS*, 347(1-2):415–418, 2005. doi:10.1016/j.tcs.2005.07.032.

33 M. Vyalyi. QMA=PP implies that PP contains PH. *Electronic Colloquium on Computational Complexity*, 2003.

34 J. Watrous. Semidefinite programs for completely bounded norms. *Theory of Computing*, 5:217–238, 2009.

35 C. Wrathall. Complete sets and the Polynomial-Time Hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976. doi:10.1016/0304-3975(76)90062-1.

36 T. Yamakami. Quantum NP and a quantum hierarchy. In *Proceedings of the 2nd IFIP International Conference on Theoretical Computer Science*, pages 323–336. Kluwer Academic Publishers, 2002.

A Subquadratic Algorithm for 3XOR

Martin Dietzfelbinger

Technische Universität Ilmenau, Germany
martin.dietzfelbinger@tu-ilmenau.de
https://orcid.org/0000-0001-5484-3474

Philipp Schlag

Technische Universität Ilmenau, Germany
philipp.schlag@tu-ilmenau.de
https://orcid.org/0000-0001-5052-9330

Stefan Walzer

Technische Universität Ilmenau, Germany
stefan.walzer@tu-ilmenau.de
https://orcid.org/0000-0002-6477-0106

Abstract

Given a set X of n binary words of equal length w, the 3XOR problem asks for three elements $a, b, c \in X$ such that $a \oplus b = c$, where $\oplus$ denotes the bitwise XOR operation. The problem can be easily solved on a word RAM with word length w in time $O(n^2 \log n)$. Using Han's fast integer sorting algorithm (STOC/J. Algorithms, 2002/2004) this can be reduced to $O(n^2 \log \log n)$. With randomization or a sophisticated deterministic dictionary construction, creating a hash table for X with constant lookup time leads to an algorithm with (expected) running time $O(n^2)$. At present, seemingly no faster algorithms are known.

We present a surprisingly simple deterministic, quadratic time algorithm for 3XOR. Its core is a version of the PATRICIA tree for X, which makes it possible to traverse the set $a \oplus X$ in ascending order for arbitrary $a \in \{0,1\}^w$ in linear time. Furthermore, we describe a randomized algorithm for 3XOR with expected running time $O(n^2 \cdot \min\{\frac{\log^3 w}{w}, \frac{(\log \log n)^2}{\log^2 n}\})$. The algorithm transfers techniques to our setting that were used by Baran, Demaine, and Pătraşcu (WADS/Algorithmica, 2005/2008) for solving the related int3SUM problem (the same problem with integer addition in place of binary XOR) in expected time $o(n^2)$. As suggested by Jafargholi and Viola (Algorithmica, 2016), linear hash functions are employed.

The latter authors also showed that assuming 3XOR needs expected running time $n^{2-o(1)}$ one can prove conditional lower bounds for triangle enumeration just as with 3SUM. We demonstrate that 3XOR can be reduced to other problems as well, treating the examples offline SetDisjointness and offline SetIntersection, which were studied for 3SUM by Kopelowitz, Pettie, and Porat (SODA, 2016).

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

Keywords and phrases 3SUM, 3XOR, Randomized Algorithms, Reductions, Conditional Lower Time Bounds

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.59

Related Version A full version of the paper is available at [11], `http://arxiv.org/abs/1804.11086`.

1 Introduction

The 3XOR problem [19] is the following: Given a set X of n binary strings of equal length w, are there elements $a, b, c \in X$ such that $a \oplus b = c$, where $\oplus$ is bitwise XOR? We work with the word RAM (Random Access Machine) [13] model with word length $w = \Omega(\log n)$, and we assume as usual that one input string fits into one word. Then, using sorting, the problem can easily be solved in time $O(n^2 \log n)$. Using Han's fast integer sorting algorithm [18] the time can be reduced to $O(n^2 \log \log n)$. In order to achieve quadratic running time, one could utilize a randomized dictionary for X with expected linear construction time and constant lookup time (like in [12]) or (weakly non-uniform, quite complicated) deterministic static dictionaries with construction time $O(n \log n)$ and constant lookup time as provided in [17]. Once such a dictionary is available, one just has to check whether $a \oplus b \in X$, for all pairs $a, b \in X$. No subquadratic algorithms seem to be known.

It is natural to compare the situation with that for the 3SUM problem, which is as follows:[1] Given a set X of n real numbers, are there $a, b, c \in X$ such that $a + b = c$? There is a very simple quadratic time algorithm for this problem (see Section 3 below). After a randomized subquadratic algorithm was suggested by Grønlund Jørgensen and Pettie [20], improvements ensued [14, 16], and recently Chan [8] gave the fastest deterministic algorithm known, with a running time of $n^2(\log \log n)^{O(1)}/\log^2 n$. The restricted version where the input consists of integers whose bit length does not exceed the word length w is called int3SUM. The currently best randomized algorithm for int3SUM was given by Baran, Demaine, and Pătraşcu [2, 3]; it runs in expected time $O(n^2 \cdot \min\{\frac{\log^2 w}{w}, \frac{(\log \log n)^2}{\log^2 n}\})$ for $w = O(n \log n)$. The 3SUM problem has received a lot of attention in recent years, because it can be used as a basis for conditional lower time bounds for problems, for instance, from computational geometry and data structures [15, 22, 26]. Because of this property, 3SUM is in the center of attention of papers dealing with low-level complexity. Chan and Lewenstein [9] give upper bounds for inputs with a certain structure. Kane, Lovett, and Moran [21] prove near-optimal upper bounds for linear decision trees. Wang [28] considers randomized algorithms for subset sum, trying to minimize the space, and Lincoln et al. [23] investigate time-space tradeoffs in deterministic algorithms for k-SUM. Barba et al. [4] examine a generalization of 3SUM in which the sum function is replaced by a constant-degree polynomial in three variables. Chan [7] shows how to adapt the ideas of the subquadratic int3SUM algorithm to the general position problem.

In contrast, 3XOR received relatively little attention, before Jafargholi and Viola [19] studied 3XOR and described techniques for reducing this problem to triangle enumeration. In this way they obtained conditional lower bounds in a way similar to the conditional lower bounds based on int3SUM.

The main results of this paper are the following: (1) We present a surprisingly simple deterministic algorithm for 3XOR that runs in time $O(n^2)$ (Theorem 5). When X is given in sorted order, it constructs in linear time a version of the PATRICIA tree [25] for X, using only word operations and not looking at single bits at all. This tree then makes it possible to traverse the set $a \oplus X$ in ascending order in linear time, for arbitrary $a \in \{0,1\}^w$. This is sufficient for achieving running time $O(n^2)$. (2) The second result is a randomized algorithm for 3XOR that runs in time $O(n^2 \cdot \min\{\frac{\log^3 w}{w}, \frac{(\log \log n)^2}{\log^2 n}\})$ for $w = O(n \log n)$ (Theorem 7),

[1] There are many different, but equivalent versions of 3SUM and 3XOR, differing in the way the input elements are grouped. Often one sees the demand that the three elements a, b, and c with $a \oplus b = c$ or $a + b = c$, resp., come from different sets.

which is almost the same bound as that of [2] for int3SUM. Finding a deterministic algorithm for 3XOR with subquadratic running time remains an open problem. (3) Finally, we reduce 3XOR to offline SetDisjointness (Theorem 10) and offline SetIntersection (Theorem 11), establishing the conditional lower bound $n^{2-o(1)}$ (as in [22] conditioned on the int3SUM conjecture).

Unfortunately, no (non-trivial) relation between the required (expected) time for 3SUM and 3XOR is known. In particular, we cannot exclude the case that one of these problems can be solved in (expected) time $O(n^{2-\varepsilon})$ for some constant $\varepsilon > 0$ whereas the other one requires (expected) time $n^{2-o(1)}$. Actually, this possibility is the background of some conditional statements on the cost of listing triangles in graphs in [19, Cor. 2]. However, due to the similarity of 3XOR to 3SUM, the question arises whether the recent results on 3SUM can be transferred to 3XOR.

In Section 2, we review the word RAM model and examine 1-universal classes of linear hash functions. In particular, we determine the evaluation cost of such hash functions and we restate a hashing lemma [2] on the expected number of elements in "overfull" buckets. Furthermore, we state how fast one can solve the set intersection problem on word-packed arrays. In Section 3, we construct a special enhanced binary search tree T_X to represent a set X of binary strings of fixed length. This representation makes it possible to traverse the set $a \oplus X$ in ascending order for any $a \in \{0,1\}^w$ in linear time, which leads to a simple deterministic algorithm for 3XOR that runs in time $O(n^2)$. Then, we turn to randomized algorithms and show how to solve 3XOR in subquadratic expected time in Section 4: $O(n^2 \cdot \min\{\frac{\log^3 w}{w}, \frac{(\log\log n)^2}{\log^2 n}\})$ for $w = O(n \log n)$, and $O(n \log^2 n)$ for $n \log n \leq w = O(2^{n \log n})$. Our approach uses the ideas of the subquadratic expected time algorithm for int3SUM presented in [2], i. e., computing buckets and fingerprints, word packing, exploiting word-level parallelism, and using lookup tables to solve the set intersection problem on word-packed arrays. Altogether, we get the same expected running time for $w = O(\log^2 n)$ and a word-length-dependent upper bound on the expected running time for $w = \omega(\log^2 n)$ that is worse by a $\log w$ factor in comparison to the int3SUM setting. Based on these results and the similarity of 3XOR to int3SUM, it seems natural to conjecture that 3XOR requires expected time $n^{2-o(1)}$, too, and so 3XOR is a candidate for reductions to other computational problems just as 3SUM. In Section 5, we describe how to reduce 3XOR to offline SetDisjointness and offline SetIntersection, transferring the results of [22] from int3SUM to 3XOR.

Recently, Bouillaguet et al. [6] studied algorithms "for the 3XOR problem". This is related to our setting, but not identical. These authors study a variant of the "generalized birthday problem", well known in cryptography as a problem to which some attacks on cryptosystems can be reduced, see [6]. Translated into our notation, their question is: Given a *random* set $X \subseteq \{0,1\}^w$ of size $3 \cdot 2^{w/3}$, find, if possible, three different strings $a, b, c \in X$ such that $a \oplus b = c$. Adapting the algorithm from [2], these authors achieve a running time of $O(2^{2w/3}(\log^2 w)/w^2)$, which corresponds to the running time of our algorithm for $n = 3 \cdot 2^{w/3}$. The difference to our situation is that their input is random. This means that the issue of 1-universal families of linear hash functions disappears (a projection of the elements in X on some bit positions does the job) and that complications from weak randomness are absent (e. g., one can use projection into relatively small buckets and use Chernoff bounds to prove that the load is very even with high probability). This means that the algorithm described in [6] does not solve our version of the 3XOR problem.

2 Preliminaries

2.1 The Word RAM Model

As is common in the context of fast algorithms for the int3SUM problem [2], we base our discussion on the *word RAM model* [13]. This is characterized by a word length w. Each memory cell can store w bits, interpreted as a bit string or an integer or a packed sequence of subwords, as is convenient. The word length w is assumed to be at least $\log n$ and at least the bit length of a component of the input. It is assumed that the operations of the *multiplicative instruction set*, i. e., arithmetic operations (addition, subtraction, multiplication), word operations (left shift, right shift), bitwise Boolean operations (AND, OR, NOT, XOR), and random memory accesses can be executed in constant time. We will write $\oplus$ to denote the bitwise XOR operation. A *randomized* word RAM also provides an operation that in constant time generates a uniformly random value in $\{0, 1, \ldots, v-1\}$ for any given $v \leq 2^w$.

2.2 Linear Hash Functions

We consider hash functions $h\colon U \to M$, where the domain ("universe") U is $\{0,1\}^\ell$ and the range M is $\{0,1\}^\mu$ with $\mu \leq \ell$. Both universe and range are vector spaces over $\mathbb{Z}_2$. In [2] and in successor papers on int3SUM "almost linear" hash functions based on integer multiplication and truncation were used, as can be found in [10]. As noted in [19], in the 3XOR setting the situation is much simpler. We may use $\mathcal{H}^{\text{lin}}_{\ell,\mu}$, the set of all $\mathbb{Z}_2$-linear functions from U to M. A function h_A from this family is described by a $\mu \times \ell$ matrix A, and given by $h_A(x) = A \cdot x$, where $x = (x_0, \ldots, x_{\ell-1})^\mathsf{T} \in U$ and $h_A(x) \in M$ are written as column vectors. For all hash functions $h \in \mathcal{H}^{\text{lin}}_{\ell,\mu}$ and all $x, y \in U$ we have $h(x \oplus y) = h(x) \oplus h(y)$, by the very definition of *linearity*. Further, this family is 1-*universal*, indeed, we have $\mathbf{Pr}_{A\in\{0,1\}^{\mu\times\ell}}[\,h_A(x) = h_A(y)\,] = \mathbf{Pr}_{A\in\{0,1\}^{\mu\times\ell}}[\,h_A(x\oplus y) = 0\,] = 2^{-\mu} = 1/|M|$, for all pairs x, y of different keys in U. We remark that the convolution class described in [24], a subfamily of $\mathcal{H}^{\text{lin}}_{\ell,\mu}$, can be used as well, as it is also 1-universal, and needs only $\ell + \mu - 1$ random bits.

The universe we consider here is $\{0,1\}^w$. The time for evaluating a hash function $h \in \mathcal{H}^{\text{lin}}_{w,\mu}$ on one or on several inputs depends on the instruction set and on the way $h = h_A$ is stored. In contrast to the int3SUM setting [2], we are not able to calculate hash values in constant time.

▶ **Lemma 1.** *For $h \in \mathcal{H}^{\text{lin}}_{w,\mu}$ and inputs from $\{0,1\}^w$ we have:*
(a) *$h(x)$ can be calculated in time $O(\mu)$, if* PARITY *of w-bit words is a constant time operation.*
(b) *$h(x)$ can always be calculated in time $O(\mu + \log w)$.*
(c) *$h(x_1), \ldots, h(x_n)$ can be evaluated in time $O(n\mu + \log w)$.*

Proof. (Sketch.) Assume $h = h_A$. For (a) we store the rows of A as w-bit strings, and obtain each bit of the hash value by a bitwise $\wedge$ operation followed by PARITY. For (b) we assume the w columns of A are stored as μ-bit blocks, in $O(\mu)$ words. An evaluation is effected by selecting the columns indicated by the 1-bits of x and calculating the $\oplus$ of these vectors in a word-parallel fashion. In $\log w$ rounds, these vectors are added, halving the number of vectors in each round. For (c), we first pack the columns selected for the n input strings into $O(n\mu)$ words and then carry out the calculation indicated in (b), but simultaneously for all x_i and within as few words as possible. This makes it possible to further exploit word-level parallelism, if μ should be much smaller than w. ◀

We shall use linear, 1-universal hashing for splitting the input set into buckets and for replacing keys by fingerprints in Section 4.

▶ Remark. In the following, we will apply Lemma 1(c) to map n binary strings of length w to hash values of length $\mu = O(\log n)$ in time $O(n \log n + \log w)$. Since $\log w$ will dominate the running time only for huge word lengths, we assume in the rest of the paper that $w = 2^{O(n \log n)}$ and that all hash values can be calculated in time $O(n \log n)$.

▶ Remark. When randomization is allowed, we will assume that we have constructed in expected $O(n)$ time a standard hash table for input set X with constant lookup time [12]. (Arbitrary 1-universal classes can be used for this.)

2.3 A Hashing Lemma for 1-Universal Families

A hash family $\mathcal{H}$ of functions from U to M is called *1-universal* if $\mathbf{Pr}_{h\in\mathcal{H}}[\, h(x) = h(y)\,] \leq 1/|M|$ for all $x, y \in U$, $x \neq y$. We map a set $S \subseteq U$ with $|S| = n$ into M with $|M| = m$ by a random element $h \in \mathcal{H}$. In [2, Lemma 4] it was noted that for 1-universal families the expected number of keys that collide with more than $3n/m$ other keys is bounded by $O(m)$. We state a slightly stronger version of that lemma. (The strengthening is not essential for the application in the present paper.)

▶ **Lemma 2** (slight strengthening of Lemma 4 in [2]). *Let $\mathcal{H}$ be a 1-universal class of hash functions from U to M, with $m = |M|$, and let $S \subseteq U$ with $|S| = n$. Choose $h \in \mathcal{H}$ uniformly at random. For $i \in M$ define $B_i = \{y \in S \mid h(y) = i\}$. Then for $2\frac{n}{m} < t \leq n$ we have:*

$$\mathbf{E}_{h\in\mathcal{H}}[\,|\{x \in S \mid |B_{h(x)}| \geq t\}|\,] < \frac{n}{t - 2\frac{n}{m}}$$

(The bound in [2] was about twice as large. The proof is given in the full version [11].)

In our algorithm, we will be interested in the number of elements in buckets with size at least three times the expectation. Choosing $t = 3\frac{n}{m}$ in Lemma 2, we conclude that the expected number of such elements is smaller than the number of buckets.

▶ **Corollary 3.** *In the setting of Lemma 2 we have* $\mathbf{E}_{h\in\mathcal{H}}[\;|\{\, x \in S \mid |B_{h(x)}| \geq 3n/m \,\}|\;] < m$.

2.4 Set Intersection on Unsorted Word-Packed Arrays

We consider the problem "*set intersection on unsorted word-packed arrays*": Assume k and ℓ are such that $k(\ell + \log k) \leq w$, and that two words a and b are given that both contain k many ℓ-bit strings: a contains $a_0, \ldots, a_{k-1}$ and b contains $b_0, \ldots, b_{k-1}$. We wish to determine whether $\{a_0, \ldots, a_{k-1}\} \cap \{b_0, \ldots, b_{k-1}\}$ is empty or not and find an element in the intersection if it is nonempty.

In [3, proof of Lemma 3] a similar problem is considered: It is assumed that a is sorted and b is *bitonic*, meaning that it is a cyclic rotation of a sequence that first grows and then falls. In this case one sorts the second sequence by a word-parallel version of bitonic merge (time $O(\log k)$), and then merges the two sequences into one sorted sequence (again in time $O(\log k)$). Identical elements now stand next to each other, and it is not hard to identify them. We can use a slightly slower modification of the approach of [3]: We sort both sequences by word-packed bitonic sort [1] (simulating Batcher's *bitonic sort* sorting network [5] on a word-packed array), which takes time $O(\log^2 k)$, and then proceed as before.[2] We obtain the following result.

[2] It is this slower version of packed intersection that causes our randomized 3XOR algorithm to be a little slower than the int3SUM algorithm for $w = \omega(\log^2 n)$.

Algorithm 1: A simple quadratic 3SUM algorithm.

```
1  Algorithm 3SUM(X):
2      sort X as x_1 < ··· < x_n
3      for a ∈ X do
4          (i, j) ← (1, 1)
5          while i ≤ n AND j ≤ n do
6              if a + x_i < x_j then
7                  i ← i + 1
8              else if a + x_i > x_j
                then
9                  j ← j + 1
10             else return (a, x_i, x_j)
11     return no solution
```

Algorithm 2: A quadratic 3XOR algorithm.

```
1  Algorithm 3XOR(X):
2      sort X as x_1 < ··· < x_n
3      T_X ← makeTree(X)
4      for a ∈ X do
5          (i, j) ← (1, 1)
6          (y_i)_{1≤i≤n} ← traverse(T_X, a)
7          while i ≤ n AND j ≤ n do
8              if y_i < x_j then
9                  i ← i + 1
10             else if y_i > x_j then
11                 j ← j + 1
12             else return (a, y_i ⊕ a, x_j)
13     return no solution
```

▶ **Lemma 4.** *Assume $k(\ell + \log k) = O(w)$, and assume that two sequences of ℓ-bit strings, each of length k, are given. Then the t entries that occur in both sequences can be listed in time $O(\log^2 k + t)$.*

A more detailed description is given in the full version [11].

3 A Deterministic 3XOR Algorithm in Quadratic Time

A well known deterministic algorithm for solving the 3SUM problem in time $O(n^2)$ is reproduced in Algorithm 1. After sorting the input X as $x_1 < \cdots < x_n$ in time $O(n \log n)$, we consider each $a \in X$ separately and look for triples of the form $a + b = c$. Such triples correspond to elements of the intersection of $a + X = \{a + x_1, \ldots, a + x_n\}$ and X. Since X is sorted, we can iterate over both X and $a + X$ in ascending order and compute the intersection with an interleaved linear scan.

Unfortunately, the $\oplus$-operation is not order preserving, indeed, $x < y$ does not imply $a \oplus x < a \oplus y$ for the lexicographic ordering on bitstrings – or, indeed, any total ordering on bitstrings. We may sort X and each set $a \oplus X = \{a \oplus x \mid x \in X\}$, for $a \in X$, separately to obtain an algorithm with running time $O(n^2 \log n)$. Using fast deterministic integer sorting [18] reduces this to time $O(n^2 \log \log n)$. In order to achieve quadratic running time, one may utilize a randomized dictionary for X with expected linear construction time and constant lookup time (like in [12]) or (weakly non-uniform, rather complex) deterministic static dictionaries with construction time $O(n \log n)$ and constant lookup time as provided in [17]. Once such a dictionary is available, one just has to check whether $a \oplus b \in X$, for all $a, b \in X$.

Here we describe a rather simple deterministic algorithm with quadratic running time. For this, we utilize a special binary search tree[3] T_X that allows, for arbitrary $a \in \{0,1\}^w$, to traverse the set $a \oplus X = \{a \oplus x \mid x \in X\}$ in lexicographically ascending order, in linear time. For $X \neq \emptyset$, the tree T_X is recursively defined as follows.

[3] The structure of the tree is that of the PATRICIA tree [25] for X.

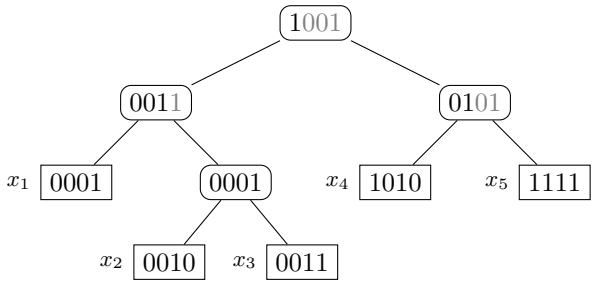


Figure 1 The tree T_X for $X = \{x_1 = 0001, x_2 = 0010, x_3 = 0011, x_4 = 1010, x_5 = 1111\}$. The first 1-bit of the label of an inner node indicates the most significant bit that is not constant among the x-values managed by that subtree (the bits after the first 1-bit are irrelevant). According to the value of this bit, elements are found in the left or right subtree. Apart from the labels of the inner nodes, T_X is essentially the PATRICIA tree [25] for X.

- If $X = \{x\}$, then T_X is $\mathsf{LeafNode}(x)$, a tree consisting of a single leaf with label x.
- If $|X| \geq 2$, let $\mathrm{lcp}(X)$ denote the longest common prefix of the elements of X when viewed as bitstrings. That is, all elements of X coincide on the first $k = |\mathrm{lcp}(X)|$ bits, the elements of some nonempty set $X_0 \subsetneq X$ start with $\mathrm{lcp}(X)0$ and the elements of $X_1 = X - X_0$ start with $\mathrm{lcp}(X)1$. We define $T_X = \mathsf{InnerNode}(T_{X_0}, 0^k1b, T_{X_1})$ for some $b \in \{0,1\}^{w-k-1}$, meaning that T_X consists of a root vertex with label $\ell = 0^k1b$, a left subtree T_{X_0} and a right subtree T_{X_1}. The choice of b is irrelevant, but it is convenient to define the label more concretely as $\ell = (\max X_0) \oplus (\min X_1)$.

Note that along paths of inner nodes down from the root the labels when regarded as integers are strictly decreasing. We give an example in Figure 1 and provide a $O(n \log n)$ time construction of T_X from X in Algorithm 4.

In the context of $T_X = \mathsf{InnerNode}(T_{X_0}, \ell = 0^k1b, T_{X_1})$ as described above, the $(k{+}1)$st bit is the most significant bit where elements of X differ. Crucially, this is also true for the set $a \oplus X$ for any $a \in \{0,1\}^w$. Since the elements of X are partitioned into X_0 and X_1 according to their $(k{+}1)$st bit, either all elements of $a \oplus X_0$ are less than all elements of $a \oplus X_1$, or vice versa, depending on whether the $(k{+}1)$st bit of a is 0 or 1. Using that the $(k{+}1)$st bit of a is 1 iff $a \oplus \ell < a$, this suggests a simple recursive algorithm to produce $a \oplus X$ in sorted order, given as Algorithm 3.

With the data structure T_X in place, the strategy from 3SUM carries over to 3XOR as seen in Algorithm 2. Summing up, we have obtained the following result:

▶ **Theorem 5.** *A deterministic word RAM can solve the* 3XOR *problem in time* $O(n^2)$. ◀

In Algorithm 4 we provide a linear time construction of T_X from a stream containing the sorted array X interleaved with the labels $\ell_i = x_i \oplus x_{i+1}$ (due to sorting the total runtime is $O(n \log n)$). Despite its brevity, the recursive build function is somewhat subtle.

▶ **Claim 6** (Correctness of Algorithm 4). *If build() is called while the stream contains the elements* $(\ell_i, x_{i+1}, \ldots, x_n, \ell_n = \infty)$*, the call consumes a prefix of the stream until* $\mathsf{top}(\mathrm{stream}) = \ell_j$ *where* $j = \min\{j > i \mid \ell_j \geq \ell_i\}$*. It returns* T_X *where* $X = \{x_{i+1}, \ldots, x_j\}$.

Once this is established, the correctness of makeTree immediately follows as for the outer call we have $i = 0$ and $j = n$ (with the understanding that $\infty \geq \infty$).

Proof of Claim 6. By the ℓ-call we mean the (recursive) call to build() with $\mathsf{top}(\mathrm{stream}) = \ell$. In particular the ℓ-call consumes ℓ from the stream and our claim concerns the ℓ_i-call. It is clear from the algorithm that an ℓ-call can only invoke an ℓ'-call if $\ell' < \ell$. Therefore the

Algorithm 3: Given a tree $T = T_X$ and $a \in \{0,1\}^w$, the algorithm yields the elements of $a \oplus X = \{a \oplus x \mid x \in X\}$ in sorted order.

```
1  Algorithm traverse(T, a):
2      if T = LeafNode(x) then
3          yield a ⊕ x
4      else
5          let T = InnerNode(T_0, ℓ, T_1)
6          if a ⊕ ℓ > a then
7              traverse(T_0, a)
8              traverse(T_1, a)
9          else
10             traverse(T_1, a)
11             traverse(T_0, a)
```

Algorithm 4: $O(n \log n)$-time algorithm to construct T_X from X.

```
1  Algorithm makeTree(X):
2      sort X as x_1 < ··· < x_n
3      let ℓ_i = x_i ⊕ x_{i+1},   1 ≤ i < n
4      stream ←
         (∞, x_1, ℓ_1, ..., ℓ_{n−1}, x_n, ∞)
5      return build() where
6      subroutine build():
7          ℓ ← pop(stream)
8          x ← pop(stream)
9          T ← LeafNode(x)
10         while top(stream) < ℓ do
11             ℓ' ← top(stream)
12             T ← InnerNode(T, ℓ', build())
13         return T
```

ℓ_i-call cannot directly or indirectly cause the ℓ_j-call since $\ell_j \geq \ell_i$. At the same time, the ℓ_i-call can only terminate when $\mathsf{top}(\mathit{stream}) \geq \ell_i$. This establishes that $\ell_j = \mathsf{top}(\mathit{stream})$ when the ℓ_i-call ends – the first part of our claim.

Next, note that since X is sorted, there is some m such that we have $X_0 = \{x_{i+1}, \ldots, x_m\}$ and $X_1 = \{x_{m+1}, \ldots, x_j\}$ where $X = X_0 \cup X_1$ is the partition from the definition of T_X. Moreover, ℓ_m is the largest label among $\ell_{i+1}, \ldots, \ell_{j-1}$. This implies that the ℓ_m-call is *directly* invoked from the ℓ_i-call. Just before the ℓ_m-call is made, the ℓ_i-call played out just as though the stream had been $(\ell_i, x_{i+1}, \ldots, x_m, \ell'_m = \infty)$, which would have produced T_{X_0} by induction[4]. However, due to $\ell_m = \mathsf{top}(\text{stream}) < \ell = \ell_i$, instead of returning $T = T_{X_0}$, the while loop is entered (again) and produces $\mathsf{InnerNode}(T = T_{X_0}, \ell = \ell_m, \text{build}())$. The stream for the ℓ_m-call is $(\ell_m, \ldots, x_n, \ell_n)$ and ℓ_j is the first label not smaller than ℓ_m. So, again by induction, the ℓ_m-call produces T_{X_1} and ends with $\mathsf{top}(\text{stream}) = \ell_j$. Given this, it is clear that afterwards the loop condition in the ℓ_i-call is not satisfied (since $\ell_j \geq \ell_i$) and the new $T = T_X$ is returned immediately, establishing the second part of the claim. ◀

4 A Subquadratic Randomized Algorithm

In this section we present a subquadratic expected time algorithm for the 3XOR problem. Its basic structure is the same as in the corresponding algorithm for int3SUM presented in [2]. We use 1-universal families of linear hash functions to split the elements into buckets and to compute short fingerprints. Due to linearity, the bucket of an element c with $c = a \oplus b$ is uniquely determined when knowing the buckets of a and b. Furthermore, if $a \oplus b = c$, then this equation is also true when looking at the fingerprints of these elements. Therefore, packing the fingerprints of all elements of a (not too full) bucket into one word (a word-packed array) allows us to evaluate the latter equation for a lot of triples "in parallel". For this

[4] Formally the induction is on the value of $j - i$. The case of $j - i = 1$ is trivial.

purpose, we exploit word-level parallelism and use lookup tables to solve the set intersection problem on unsorted word-packed arrays. Algorithm 5 illustrates this strategy for large word lengths $w = \Omega((\log^2 n) \log\log n)$.

Changes (in comparison to the int3SUM algorithm) are made where necessary to deal with the different setting. This makes it a little more difficult in some parts of the algorithm (mainly because XOR-ing a sorted sequence with some a will destroy the order) and easier in other parts (in particular where linearity of hash functions is concerned). Altogether, we get an expected running time that is the same as in [2] for $w = O(\log^2 n)$ and slightly worse for larger w. Recall we assume $w = 2^{O(n \log n)}$ throughout.

▶ **Theorem 7.** *A randomized word RAM with word length w can solve the* 3XOR *problem in expected time*

$$O\left(n^2 \cdot \min\left\{\frac{\log^3 w}{w}, \frac{(\log\log n)^2}{\log^2 n}\right\}\right) \quad \text{for } w = O(n \log n),$$

and $O(n \log^2 n)$, otherwise.

The crossover point between the w and the $\log n$ factor is $w = (\log^2 n) \log\log n$. The only difference to the running time of [2] is in an extra factor $\log w$ in the word-length-dependent part.

Proof. We describe the main ideas of the algorithm. For full details, see [11]. If $w = \omega(n \log n)$, we proceed as for $w = \Theta(n \log n)$. We use two levels of hashing.

Good and Bad Buckets. We split X into $R = 2^r = o(n)$ *buckets* $X_u = \{\, x \in X \mid h_1(x) = u \,\}$, $u \in \{0,1\}^r$, using a randomly chosen hash function $h_1 \in \mathcal{H}^{\text{lin}}_{w,r}$. The hash values are calculated once and for all and stored for further use. By linearity, for every solution $a \oplus b = c$ we also have $h_1(a) \oplus h_1(b) = h_1(c)$. Given $a \in X_u$ and $b \in X_v$, we only have to inspect bucket $X_{u \oplus v}$ when looking for a $c \in X$ such that $a \oplus b = c$.

For $a \in X$, the expected size of bucket $X_{h_1(a)}$ is n/R. A bucket of size larger than $3n/R$ is called *bad*, as are elements of bad buckets. All other buckets and elements are called *good*. By Corollary 3, the expected number of bad elements is smaller than R. We can even assume that the total number of bad elements is smaller than $2R$. (By Markov's inequality, we simply have to repeat the choice of h_1 expected $O(1)$ times until this condition is satisfied.)

Fingerprints and Word-Packed Arrays. Furthermore, we use another hash function $h_2 \in \mathcal{H}^{\text{lin}}_{w,p}$ for some appropriately chosen p to calculate p-bit *fingerprints* for all elements in X. If $(3n/R) \cdot p \le w$, we can pack all fingerprints of elements of a good bucket X_u into one word X^*_u. This packed representation is called *word-packed array*. Again by linearity, for every solution $a \oplus b = c$ we have $h_2(a) \oplus h_2(b) = h_2(c)$. On the other hand, the expected number of *colliding triples*, i. e., triples with $a \oplus b \neq c$ but $h_1(a) \oplus h_1(b) = h_1(c)$ and $h_2(a) \oplus h_2(b) = h_2(c)$, is at most $2n^3/(R \cdot 2^p)$. (Since the hash families that we use are 1-universal, the probability that a triple with $a \oplus b \neq c$ is colliding is at most $2/(R \cdot 2^p)$. The additional factor 2 is due to the repeated choice of h_1 until there are fewer than $2R$ bad elements.)

The total time for all the hashing steps described so far is $O(n \cdot (r + p))$, see Section 2.2. We consider two choices of $R = 2^r$ and p, cf. [2, proof of Lemma 3] and [2, proof of Thm. 2]. The first one is better for larger words of length $w = \Omega((\log^2 n) \log\log n)$ whereas the second one yields better results for smaller words. In both cases, we search for triples with a fixed number of bad elements separately. The strategies for finding triples of good elements

correspond to the approach for int3SUM in [2]. However, for triples with at least one bad element we have to rely on a more fine-grained examination than in [2]. For this, we will use hash tables and another lookup table.

Long Words: Exploiting Word-Level Parallelism. If the word length is large enough, i. e., $w = \Omega((\log^2 n) \log\log n)$, we choose $R \approx \lceil 6 \cdot n \cdot (\log w)/w \rceil$ as a power of 2 and $p = \lfloor 2 \cdot \log w \rfloor$ to be able to pack all fingerprints of elements of a good bucket into one word. We examine triples with at most one and at least two bad elements separately, as seen in Algorithm 5.

When looking for triples with at most one bad element, we do the following for every (good or bad) $a \in X$ and $u \in \{0,1\}^r$ where X_u and the corresponding bucket $X_{h_1(a) \oplus u}$ are good (as in [2, proof of Lemma 3] when examining triples of three good elements): We XOR every fingerprint of the word-packed array X_u^* with $h_2(a)$. Then, in time $O(\log^2(n/R) + t) = O(\log^2 w + t)$, we construct a list of t common pairs in this modified word-packed array and $X^*_{h_1(a) \oplus u}$, which is possible by Lemma 4. For each such pair, we only have to check whether it derives from a non-colliding triple. Since we can stop when we find a non-colliding triple and since the expected total number of colliding triples is $O(n^2/(w \log w))$, we are done in expected time $O(n \cdot R \cdot \log^2 w + n^2/(w \log w)) = O(n^2(\log^3 w)/w)$.

In order to examine all triples with at least two bad elements, we provide a hash table for X with expected construction time $O(n)$ and constant lookup time [12]. Now, for each of the at most $4R^2 = O(n^2(\log^2 w)/w^2)$ pairs (a, b) of bad elements we can check if $a \oplus b \in X$ in constant time.[5]

The total expected running time for this parameter choice is $O(n^2(\log^3 w)/w)$.

Short Words: Using Lookup Tables. For word lengths $w = O((\log^2 n) \log\log n)$, we choose $R \approx \lceil 55 \cdot n \cdot (\log\log n)/\log n \rceil$ as a power of 2 and $p = \lfloor 6 \cdot \log\log n \rfloor$ to pack all fingerprints of elements of a good bucket into $(\frac{1}{3} - \varepsilon) \log n$ bits, for some $\varepsilon > 0$.

We start by looking for triples with no bad element. For this, we consider all $\leq R^2$ triples of corresponding good buckets (as in [2, proof of Thm. 2]). We use a lookup table of size $n^{1-\Omega(1)}$ to check whether such a triple of buckets yields a triple of fingerprints (in the word-packed arrays) with $h_2(a) \oplus h_2(b) = h_2(c)$ in constant time. If this is the case, we search for a corresponding triple $a \oplus b = c$ in the buckets of size $O((\log n)/\log\log n)$ in time $O((\log^3 n)/(\log\log n)^3)$. Since one table entry can be computed in time $O((\log^3 n)/(\log\log n)^3)$, setting up the lookup table takes time $n^{1-\Omega(1)}$. Furthermore, the expected $O(n^2/((\log\log n)\log^5 n))$ colliding triples cause additional expected running time $O(n^2/((\log\log n)^4 \log^2 n))$. Since we can stop when we find a non-colliding triple, the total expected time is $O(R^2) = O(n^2(\log\log n)^2/\log^2 n)$.

Searching for triples with exactly one bad element can be done in a similar way. For each bad element $a \in X$ and each good bucket X_u, $u \in \{0,1\}^r$, we XOR all fingerprints in the word-packed array X_u^* with $h_2(a)$ and use a lookup table to check whether it has some fingerprints in common with the word-packed array $X^*_{h_1(a) \oplus u}$ of the corresponding good bucket. If this lookup yields a positive result, we check all pairs in the corresponding buckets in time $O((\log^2 n)/(\log\log n)^2)$. As before, the expected running time is $O(R^2)$, including the expected time $O(n^2/((\log\log n)^3 \log^3 n))$ due to colliding triples.

Examining all triples with at least two bad elements can be done using a hash table as mentioned above in expected time $O(n + R^2)$.

The total expected running time for this parameter choice is $O(n^2(\log\log n)^2/\log^2 n)$. ◀

[5] Note that it would not be possible to derive expected time $O(R^2)$ for checking all pairs of bad elements if we did not start all over if the number of keys in bad buckets is at least $2R$.

Algorithm 5: The randomized subquadratic 3XOR algorithm for the case $w = \Omega((\log^2 n)\log\log n)$. For $w = o((\log^2 n)\log\log n)$ using lookup tables to search for solutions involving at most one bad element yields a faster algorithm.

Algorithm 3XOR(X): // $X \subseteq \{0,1\}^w$, $|X| = n$
  repeat
    // partition X into buckets using h_1:
    pick linear, 1-universal $h_1 : \{0,1\}^w \to \{0,1\}^r$ with $2^r = R \approx \lceil 6n(\log w)/w \rceil$
    $X_u \leftarrow \{x \in X \mid h_1(x) = u\}$ for $u \in \{0,1\}^r$
    $B \leftarrow \{x \in X \mid |X_{h(x)}| > 3\frac{n}{R}\}$ // bad elements in overfull buckets
  until $|B| < 2R$
  // search for solution involving at least two bad elements:
  for $a, b \in B$ **do** // $< 4R^2$ choices
    if $a \oplus b \in X$ **then** // $O(1)$ using appropriate hash table for X
      return $(a, b, a \oplus b)$
  // search for solution involving at most one bad element:
  $X_u \leftarrow \emptyset$ for $u \in \{0,1\}^r$ with $|X_u| > 3\frac{n}{R}$ // empty the bad buckets
  pick linear, 1-universal $h_2 : \{0,1\}^w \to \{0,1\}^p$ with $p = \lfloor 2\log w \rfloor$
  for $u \in \{0,1\}^r$ **do**
    // pack fingerprints of elements of X_u into one word X_u^*
    $X_u^* \leftarrow h_2(X_u) :=$ concatenate $\{h_2(x) \mid x \in X_u\}$
  for $a \in X$ *and* $u \in \{0,1\}^r$ **do** // $n \cdot R$ iterations
    $X_u^{*,a} \leftarrow X_u^* \oplus h_2(a)$ // $h_2(a)$ added to each fingerprint in X_u^*
    for $v \in X_u^{*,a} \cap X^*_{h_1(a)\oplus u}$ **do**
      identify responsible b, c, in particular with $v = h_2(a) \oplus h_2(b) = h_2(c), \quad h_1(b) = u$
      if $a \oplus b = c$ **then**
        return (a, b, c)
  return *no solution*

needs time $O(\log^2(n/R))$ plus size of intersection

5 Conditional Lower Bounds from the 3XOR Conjecture

As already mentioned in Section 1, the best word RAM algorithm for int3SUM currently known [2] can solve this problem in expected time $O(n^2 \cdot \min\{\frac{\log^2 w}{w}, \frac{(\log\log n)^2}{\log^2 n}\})$ for $w = O(n\log n)$. The best deterministic algorithm [8] takes time $n^2(\log\log n)^{O(1)}/\log^2 n$. It is a popular conjecture that every algorithm for 3SUM (deterministic or randomized) needs (expected) time $n^{2-o(1)}$. Therefore, this conjectured lower bound can be used as a basis for conditional lower bounds for a wide range of other problems [15, 19, 22, 26].

Similarly, it seems natural to conjecture that every algorithm for the related 3XOR problem (deterministic or randomized) needs (expected) time $n^{2-o(1)}$. (In Theorem 7, the upper bound for short word lengths is $n^2\frac{(\log\log n)^2}{\log^2 n} = n^{2-(2\log\log n - 2\log\log\log n)/\log n}$ where $(2\log\log n - 2\log\log\log n)/\log n = o(1)$.) Therefore, it is a valid candidate for reductions to other computational problems [19, 27].

The general strategy of the subquadratic int3SUM algorithm [2], already employed in Section 4, is quite similar to the reductions in [22]. Therefore, we are able to reduce 3XOR to offline SetDisjointness and offline SetIntersection, too. Hence, the conditional lower bounds

for the problems mentioned in [22] (and bounds for dynamic problems from [26]) also hold with respect to the 3XOR conjecture. A detailed discussion can be found in [27]. Below, we will outline the general proof strategy.

5.1 Offline SetDisjointness and Offline SetIntersection

We reduce 3XOR to the following two problems.

▶ **Problem 8** (Offline SetDisjointness). ***Input:*** *Finite set C, finite families A and B of subsets of C, $q \in \mathbb{N}$ pairs of subsets $(S, S') \in A \times B$.*
Task: *Find all of the q pairs (S, S') with $S \cap S' \neq \emptyset$.*

▶ **Problem 9** (Offline SetIntersection). ***Input:*** *Finite set C, finite families A and B of subsets of C, $q \in \mathbb{N}$ pairs of subsets $(S, S') \in A \times B$.*
Task: *List all elements of the intersections $S \cap S'$ of the q pairs (S, S').*

5.2 Reductions from 3XOR

By giving an expected time $\leq n^{2-\Omega(1)}$ reduction from 3XOR to offline SetDisjointness and offline SetIntersection, we can prove lower bounds for the latter two problems, conditioned on the 3XOR conjecture.

▶ **Theorem 10.** *Assume* 3XOR *requires expected time $\Omega(n^2/f(n))$ for $f(n) = n^{o(1)}$ on a word RAM. Then for $0 < \gamma < 1$ every algorithm for* offline SetDisjointness *that works on instances with $|C| = \Theta(n^{2-2\gamma})$, $|A| = |B| = \Theta(n \log n)$, $|S| = O(n^{1-\gamma})$ for all $S \in A \cup B$ and $q = \Theta(n^{1+\gamma} \log n)$ requires expected time $\Omega(n^2/f(n))$.*

▶ **Theorem 11.** *Assume* 3XOR *requires expected time $\Omega(n^2/f(n))$ for $f(n) = n^{o(1)}$ on a word RAM. Then for $0 \leq \gamma < 1$ and $\delta > 0$, every algorithm for* offline SetIntersection *which works on instances with $|C| = \Theta(n^{1+\delta-\gamma})$, $|A| = |B| = \Theta(\sqrt{n^{1+\delta+\gamma}})$, $|S| = O(n^{1-\gamma})$ for all $S \in A \cup B$, $q = \Theta(n^{1+\gamma})$ and expected output size $O(n^{2-\delta})$ requires expected time $\Omega(n^2/f(n))$.*

Proof. (For more details, see [27, ch. 6]. Algorithm 6 reduces 3XOR to offline SetDisjointness. The pseudocode implementation for offline SetIntersection is given in [11].) Let $X \subseteq \{0,1\}^w$ be the given 3XOR instance. As in Section 4, we use two levels of hashing.

At first, we hash the elements of X with a randomly chosen hash function $h_1 \in \mathcal{H}^{\text{lin}}_{w,r}$ into $R = 2^r = \Theta(n^\gamma)$ buckets in time $O(n \log n)$. Then, we apply Corollary 3: There are expected $O(R) = O(n^\gamma)$ elements in buckets with more than three times their expected size. For each such bad element, we can naively check in time $O(n \log n)$ whether it is part of a triple (a, b, c) with $a \oplus b = c$ or not. Since $\gamma < 1$, all bad elements can be checked in expected time $\leq n^{2-\Omega(1)}$. Therefore, we can assume that every bucket X_u, $u \in \{0,1\}^r$, has at most $3\frac{n}{R} = O(n^{1-\gamma})$ elements.

The second level of hashing uses two independently and randomly chosen hash functions $h_{21}, h_{22} \in \mathcal{H}^{\text{lin}}_{w,p}$ where $P = 2^{2p} = (5n/R)^2 = O(n^{2-2\gamma})$ for offline SetDisjointness and $P = 2^{2p} = n^{1+\delta}/R = O(n^{1+\delta-\gamma})$ for offline SetIntersection. (The function h_2 with $h_2(x) = h_{21}(x) \circ h_{22}(x)$, where $\circ$ denotes the concatenation of bitstrings, is randomly chosen from a linear and 1-universal class $\mathcal{H}$ of hash functions $\{0,1\}^w \to \{0,1\}^{2p}$.) The hash values can be calculated in time $O(n \log^2 n)$. (The additional $\log n$ factor is only necessary for offline SetDisjointness, since we need to use $\Theta(\log n)$ choices of hash functions h_2 to get an error probability that is small enough.) For each $u \in \{0,1\}^r$ and $v \in \{0,1\}^p$, we create "shifted" buckets $X^{\uparrow}_{u,v} = \{\, h_2(x) \oplus (v \circ 0^p) \mid x \in X_u \,\}$ and $X^{\downarrow}_{u,v} = \{\, h_2(x) \oplus (0^p \circ v) \mid x \in X_u \,\}$. One

such set can be computed in time $O(n^{1-\gamma})$. Therefore, all sets can be computed in time $O(R\sqrt{P}\log n \cdot n^{1-\gamma}) = O(n^{2-\gamma}\log n)$ for offline SetDisjointness and $O(R\sqrt{P}\cdot n^{1-\gamma}) = O(n^{(3+\delta-\gamma)/2})$ for offline SetIntersection.

We can show that for all $u \in \{0,1\}^r$ and $c \in X$, if there are $a, b \in X$ such that $a \oplus b = c$ and $a \in X_u$, then $X^{\uparrow}_{u,h_{21}(c)} \cap X^{\downarrow}_{u\oplus h_1(c),h_{22}(c)} \neq \emptyset$. Therefore, we create the following offline SetDisjointness (offline SetIntersection) instance: $C := \{0,1\}^{2p}$, $A := \{\, X^{\uparrow}_{u,v} \mid u \in \{0,1\}^r, v \in \{0,1\}^p \,\}$, $B := \{\, X^{\downarrow}_{u,v} \mid u \in \{0,1\}^r, v \in \{0,1\}^p \,\}$ and q queries $(X^{\uparrow}_{u,h_{21}(c)}, X^{\downarrow}_{u\oplus h_1(c),h_{22}(c)})$ for all $u \in \{0,1\}^r$ and $c \in X$ in time $\leq n^{2-\Omega(1)}$. (These are $R \cdot n = \Theta(n^{1+\gamma})$ queries for offline SetIntersection. For offline SetDisjointness, we create $R \cdot n$ queries for each of the $\Theta(\log n)$ choices of h_2.)

After the offline SetDisjointness or offline SetIntersection instance has been solved, we can use this answer to compute the answer for X in expected time $\leq n^{2-\Omega(1)}$. We only have to check if a positive answer from offline SetDisjointness (a pair with non-empty intersection) or offline SetIntersection (an element of an intersection) yields a solution triple of X or not.

For offline SetDisjointness, we can show that the probability for a triple to yield a false positive can be made polynomially small if we consider $K = \Theta(\log n)$ choices of h_2 and only examine $(X_u \oplus c) \cap X_{h_1(c)\oplus u}$ if this is suggested by all K corresponding queries. For offline SetIntersection, the expected number of colliding triples is $O(n^{2-\delta})$. By trying to guess a good triple $\Theta(n \log n)$ times before creating the offline SetIntersection instance we can avoid a problem for the expected running time if a 3XOR instance yields an offline SetIntersection instance with output size $\omega(n^{2-\delta})$.

For all relevant values of γ and δ, the total running time is $\leq n^{2-\Omega(1)}$ in addition to the time needed to solve the offline SetDisjointness or offline SetIntersection instance. ◀

6 Conclusions and Remarks

We have presented a simple deterministic algorithm with running time $O(n^2)$. Its core is a version of the PATRICIA tree for $X \subseteq \{0,1\}^w$, which makes it possible to traverse the set $a \oplus X$ in ascending order for arbitrary $a \in \{0,1\}^w$ in linear time. Furthermore, our randomized algorithm solves the 3XOR problem in expected time $O(n^2 \cdot \min\{\frac{\log^3 w}{w}, \frac{(\log\log n)^2}{\log^2 n}\})$ for $w = O(n \log n)$, and $O(n \log^2 n)$ for $n \log n \leq w = O(2^{n \log n})$. The crossover point between the w and the $\log n$ factor is $w = (\log^2 n)\log\log n$. The only difference to the running time of [2] is in an extra factor $\log w$ in the word-length-dependent part. This is due to the necessity to re-sort a word-packed array of size $O(w/\log w)$ in time $O(\log^2 w)$ after we have XOR-ed each of its elements with a (common) element. Finally, we have reduced 3XOR to offline SetDisjointness and offline SetIntersection, establishing conditional lower bounds (as in [22] conditioned on the int3SUM conjecture).

A simple, but important observation, which is used in apparently all deterministic subquadratic time algorithms for 3SUM, is *Fredman's trick*:

$$a + b < c + d \iff a - d < c - b \qquad \text{for all } a, b, c, d \in \mathbb{Z}\,.$$

Unfortunately, such a relation does not exist in our setting, since there is no linear order $\prec$ on $\{0,1\}^w$ such that $a \oplus b \prec c \oplus d \iff a \oplus d \prec c \oplus b$ holds for all $a, b, c, d \in \{0,1\}^w$. Since all elements are self-inverse, for $a = b = c = 0^w$ and any $d \in \{0,1\}^w$, we would get $0^w \prec d \iff d \prec 0^w$. Is there another, "trivial-looking" trick for 3XOR, that establishes a basic approach to solve 3XOR in deterministic subquadratic time?

Algorithm 6: Algorithm reducing 3XOR to offline SetDisjointness, establishing a conditional lower bound on the runtime of offline SetDisjointness.

Reduction 3XOR-to-offlineSetDisjointness(X, γ): // $X \subseteq \{0,1\}^w$, $|X| = n$, $0 < \gamma < 1$
  // partition X into buckets using h_1:
  pick linear, 1-universal $h_1 : \{0,1\}^w \to \{0,1\}^r$ with $2^r = R \approx \lceil n^\gamma \rceil$
  $X_u \leftarrow \{x \in X \mid h_1(x) = u\}$ for $u \in \{0,1\}^r$
  $B \leftarrow \{x \in X \mid |X_{h(x)}| > 3\frac{n}{R}\}$ // bad elements in overfull buckets
  for $b \in B$ **do** // expected $O(R)$ elements
    $X^{\oplus b} \leftarrow \text{sort}\{a \oplus b \mid a \in X\}$
    if $\exists c \in X^{\oplus b} \cap X$ **then**
      return $(c \oplus b, b, c)$
  // create shifted buckets using h^i_{21}, h^i_{22}:
  pick linear, 1-universal $h^i_{21}, h^i_{22} : \{0,1\}^w \to \{0,1\}^p$ with $2^{2p} = P \approx \lceil (5n/R)^2 \rceil = O(n^{2-2\gamma})$ and $0 \le i < \lceil \log n \rceil$
  for $u \in \{0,1\}^r$, $v \in \{0,1\}^p$ *and* $0 \le i < \lceil \log n \rceil$ **do**
    $X^{\uparrow,i}_{u,v} \leftarrow \{(h^i_{21}(a) \oplus v, h^i_{22}(a)) \mid a \in X_u\}$
    $X^{\downarrow,i}_{u,v} \leftarrow \{(h^i_{21}(a), h^i_{22}(a) \oplus v) \mid a \in X_u\}$
  // apply algorithm for offline SetDisjointness:
  $(A, B, C, Q) \leftarrow ((X^{\uparrow,i}_{u,v})_{u,v,i}, (X^{\downarrow,i}_{u,v})_{u,v,i}, \{0,1\}^{2p}, \emptyset)$
  for $c \in X$, $u \in \{0,1\}^r$ *and* $0 \le i < \lceil \log n \rceil$ **do**
    $q \leftarrow (X^{\uparrow,i}_{u,h^i_{21}(c)}, X^{\downarrow,i}_{u \oplus h_1(c), h^i_{22}(c)})$, identified by (c, u, i)
    $Q \leftarrow Q \cup \{q\}$
  $Q' \leftarrow$ offlineSetDisjointness(A,B,C,Q) // $Q' \subseteq Q$
  // calculate solution for the 3XOR instance:
  for $c \in X$ *and* $u \in \{0,1\}^r$ **do**
    if $(c, u, i) \in Q'$ *for all* $0 \le i < \lceil \log n \rceil$ **then**
      $X^{\oplus c}_u \leftarrow \text{sort}\{a \oplus c \mid a \in X_u\}$
      if $\exists b \in X^{\oplus c}_u \cap X_{h_1(c) \oplus u}$ **then**
        return $(b \oplus c, b, c)$
  return *no solution*

Another open question is how the optimal running times for 3SUM and 3XOR are related. At first sight, the two problems seem to be very similar, but the details make the difference. The observations mentioned above (especially the problem of re-sorting slightly modified word-packed arrays and the possible absence of a relation like Fredman's trick) hint at a larger gap than expected. On the other hand, the fact that both problems can be reduced to a wide variety of computational problems in a similar way (e. g. listing triangles in a graph, offline SetDisjointness and offline SetIntersection) increases hope for a more concrete dependance.

References

1 Susanne Albers and Torben Hagerup. Improved Parallel Integer Sorting without Concurrent Writing. *Inf. Comput.*, 136(1):25–51, 1997.
2 Ilya Baran, Erik D. Demaine, and Mihai Pătraşcu. Subquadratic Algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008.
3 Ilya Baran, Erik D. Demaine, and Mihai Pătraşcu. Subquadratic Algorithms for 3SUM. In *WADS*, pages 409–421, 2005.

4 Luis Barba, Jean Cardinal, John Iacono, Stefan Langerman, Aurélien Ooms, and Noam Solomon. Subquadratic Algorithms for Algebraic Generalizations of 3SUM. In *SoCG*, volume 77 of *LIPIcs*, pages 13:1–13:15, 2017.

5 K. E. Batcher. Sorting Networks and Their Applications. In *SJCC*, AFIPS (Spring), pages 307–314, 1968.

6 Charles Bouillaguet, Claire Delaplace, and Pierre-Alain Fouque. Revisiting and Improving Algorithms for the 3XOR Problem. *IACR ToSC*, 2018(1):254–276, 2018.

7 Timothy M. Chan. The Art of Shaving Logs. WADS (Invited Talk), 2013.

8 Timothy M. Chan. More Logarithmic-Factor Speedups for 3SUM, (median, +)-Convolution, and Some Geometric 3SUM-Hard Problems. In *SODA*, pages 881–897, 2018.

9 Timothy M. Chan and Moshe Lewenstein. Clustered Integer 3SUM via Additive Combinatorics. In *STOC*, pages 31–40, 2015.

10 Martin Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *STACS*, pages 569–580, 1996.

11 Martin Dietzfelbinger, Philipp Schlag, and Stefan Walzer. A Subquadratic Algorithm for 3XOR. *CoRR*, abs/1804.11086, 2018. `arXiv:1804.11086`.

12 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a Sparse Table with $O(1)$ Worst Case Access Time. *J. ACM*, 31(3):538–544, 1984.

13 Michael L. Fredman and Dan E. Willard. Surpassing the Information Theoretic Bound with Fusion Trees. *J. Comput. Syst. Sci.*, 47(3):424–436, 1993.

14 Ari Freund. Improved Subquadratic 3SUM. *Algorithmica*, 77(2):440–458, Feb 2017.

15 Anka Gajentaan and Mark H. Overmars. On a Class of $O(n^2)$ Problems in Computational Geometry. *Comput. Geom.*, 5(3):165–185, 1995.

16 Omer Gold and Micha Sharir. Improved Bounds for 3SUM, k-SUM, and Linear Degeneracy. *CoRR*, abs/1512.05279, 2015.

17 Torben Hagerup, Peter Bro Miltersen, and Rasmus Pagh. Deterministic Dictionaries. *J. Algorithms*, 41(1):69–85, 2001. `doi:10.1006/jagm.2001.1171`.

18 Yijie Han. Deterministic sorting in $O(n \log\log n)$ time and linear space. *J. Algorithms*, 50(1):96–105, 2004.

19 Zahra Jafargholi and Emanuele Viola. 3SUM, 3XOR, Triangles. *Algorithmica*, 74(1):326–343, 2016.

20 Allan Grønlund Jørgensen and Seth Pettie. Threesomes, Degenerates, and Love Triangles. In *FOCS*, pages 621–630, 2014.

21 Daniel M. Kane, Shachar Lovett, and Shay Moran. Near-optimal Linear Decision Trees for k-SUM and Related Problems. In *STOC*, pages 554–563, 2018.

22 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In *SODA*, pages 1272–1287, 2016.

23 Andrea Lincoln, Virginia Vassilevska Williams, Joshua R. Wang, and R. Ryan Williams. Deterministic Time-Space Tradeoffs for k-SUM. *CoRR*, abs/1605.07285, 2016.

24 Y. Mansour, N. Nisan, and P. Tiwari. The Computational Complexity of Universal Hashing. *Theor. Comput. Sci.*, 107(1):121–133, 1993.

25 Donald R. Morrison. PATRICIA - Practical Algorithm To Retrieve Information Coded in Alphanumeric. *J. ACM*, 15(4):514–534, 1968. `doi:10.1145/321479.321481`.

26 Mihai Pătraşcu. Towards Polynomial Lower Bounds for Dynamic Problems. In *STOC*, pages 603–610, 2010.

27 Philipp Schlag. Untere Schranken für Berechnungsprobleme auf der Basis der 3SUM-Vermutung. Master's thesis, TU Ilmenau, Germany, 2016.

28 Joshua R. Wang. Space-Efficient Randomized Algorithms for K-SUM. In *ESA*, pages 810–829, 2014.

Treewidth-Two Graphs as a Free Algebra

Christian Doczkal
Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, France
christian.doczkal@ens-lyon.fr

Damien Pous
Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, France
damien.pous@ens-lyon.fr

Abstract

We give a new and elementary proof that the graphs of treewidth at most two can be seen as a free algebra. This result was originally established through an elaborate analysis of the structure of K_4-free graphs, ultimately reproving the well-known fact that the graphs of treewidth at most two are precisely those excluding K_4 as a minor. Our new proof is based on a confluent and terminating rewriting system for term-labeled graphs and does not involve graph minors anymore. The new strategy is simpler and robust in the sense that it can be adapted to subclasses of treewidth-two graphs, e.g., graphs without self-loops.

2012 ACM Subject Classification Mathematics of computing → Graph theory, Theory of computation → Rewrite systems, Computing methodologies → Symbolic and algebraic manipulation

Keywords and phrases Treewidth, Universal Algebra, Rewriting

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.60

Related Version A long version is available at [11], `https://hal.archives-ouvertes.fr/hal-01780844`.

Funding This work has been funded by the European Research Council (ERC) under the European Union's Horizon 2020 programme (CoVeCe, grant agreement No 678157). This work was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

1 Introduction

The notion of *treewidth* [9] is a cornerstone of algorithmic graph theory and parameterised complexity: treewidth measures how close a graph is to a forest, and many problems that can be solved in polynomial time on forests but are NP-complete on arbitrary graphs remain polynomial on classes of graphs of bounded treewidth. This is the case for instance for the graph homomorphism problem (and thus k-coloring) [13, 5, 14].

Similar to trees, graphs of bounded treewidth can be described by a variety of syntaxes [8]. Among the open problems, there is the question, for graphs of a given treewidth, of finding a syntax making it possible to get a finite and equational axiomatisation of graph isomorphism [8, page 118]. This question was recently answered positively for directed multigraphs of treewidth at most two [7].

The syntax used in [7] is comprised of two binary operations: *series* and *parallel* composition [12], their neutral elements, and a unary *converse* operation. In this syntax, several terms may denote the same graph (up-to isomorphism); the key result of [7] is that the cor-

responding equational theory is characterized by twelve equational axioms, defining so-called 2p-algebras.

To get this result, the authors define a function $\mathtt{t}$ from graphs to terms and establish that $\mathtt{t}$ is a isomorphism of 2p-algebras. The function $\mathtt{t}$ is defined using an elaborate analysis of the structure of treewidth-two graphs, which requires complicated graph-theoretical arguments that are not directly related to the proposed axiom system. For instance they ultimately reprove the well-known fact that the graphs of treewidth at most two are precisely those graphs excluding K_4 (the complete graph with four vertices) as a minor [12]. The authors also make $\mathtt{t}$ as canonical as possible in order to facilitate the proof that on isomorphic graphs, $\mathtt{t}$ returns terms that are congruent modulo the axioms. This comes at the price of complicating the proofs that $\mathtt{t}$ is a homomorphism of 2p-algebras.

In the present paper, we reprove the result from [7] using a completely different approach inspired by [2]: instead of using an elaborate top-down analysis, we design a graph rewriting system on term-labeled graphs and use it to reduce graphs, in a bottom-up fashion, to a shape where a term can be read off. This process is highly nondeterministic but can be shown confluent modulo the axioms. This results in big simplifications: tree decompositions are only used to show that all treewidth-two graphs can be reduced to the point where a term can be read off, and minors are not used at all in this new approach.

Another important feature of this new proof is that it makes it possible to discover the required axioms almost automatically, mainly during the confluence proof. It is also more robust: it allows us to solve two problems left open in [7], characterizing connected graphs as a free-algebra, and characterizing self-loop free graphs as a free-algebra, in both cases for graphs of treewidth at most two.

The first problem was solved recently [16] using a purely model-theoretic argument: 2p-algebras form a conservative extension of *2pdom-algebras*, the counterpart of 2p-algebras for connected graphs. Our strategy makes it possible to proceed the other way around: we prove the main result for connected graphs and 2pdom-algebras (Sections 3 to 5), before extending it to potentially disconnected graphs and 2p-algebras using a simple and mainly algebraic argument (Section 6).

The second problem was still open. We solve it using a slight variation of the presented proof, which actually leads us to the discovery of the required axioms (Section 7).

2 Preliminaries: 2p- and 2pdom-algebras

We recall the definitions of 2p- and 2pdom-algebras [7, 16]. We let $a, b \dots$ range over the *letters* of a fixed alphabet $\mathcal{A}$. We consider labeled directed graphs with two designated vertices. We just call them graphs in the sequel.

▶ **Definition 1.** A *graph* is a tuple $G = \langle V, E, s, t, l, \iota, o\rangle$, where V is a finite set of *vertices*, E is a finite set of *edges*, $s, t : E \to V$ are maps indicating the *source* and *target* of each edge, $l : E \to \mathcal{A}$ is a map indicating the *label* of each edge, and $\iota, o \in V$ are the designated vertices, respectively called *input* and *output*.

Note that we allow multiple edges between two vertices, as well as self-loops.

▶ **Definition 2.** A *homomorphism* from $G = \langle V, E, s, t, l, \iota, o\rangle$ to $G' = \langle V', E', s', t', l', \iota', o'\rangle$ is a pair $h = \langle f, g\rangle$ of functions $f : V \to V'$ and $g : E \to E'$ that respect the various components: $s' \circ g = f \circ s$, $t' \circ g = f \circ t$, $l = l' \circ g$, $\iota' = f(\iota)$, and $o' = f(o)$.

A *(graph) isomorphism* is a homomorphism whose two components are bijective functions. We write $G \simeq G'$ when there exists an isomorphism between graphs G and G'.

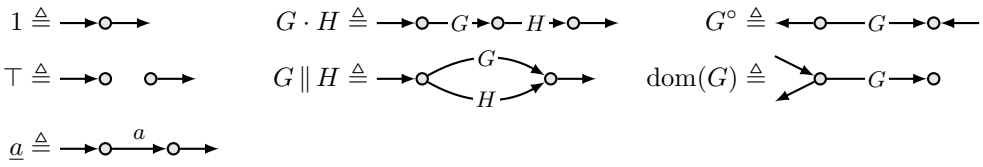


Figure 1 Graph operations.

$$u \parallel (v \parallel w) = (u \parallel v) \parallel w \quad \text{(A1)}$$
$$u \parallel v = v \parallel u \quad \text{(A2)}$$
$$u \cdot (v \cdot w) = (u \cdot v) \cdot w \quad \text{(A4)}$$
$$u \cdot 1 = u \quad \text{(A5)}$$
$$u^{\circ\circ} = u \quad \text{(A6)}$$
$$(u \parallel v)^\circ = u^\circ \parallel v^\circ \quad \text{(A7)}$$
$$(u \cdot v)^\circ = v^\circ \cdot u^\circ \quad \text{(A8)}$$
$$1 \parallel 1 = 1 \quad \text{(A9)}$$
$$\mathrm{dom}(u \parallel v) = 1 \parallel u \cdot v^\circ \quad \text{(A10)}$$
$$u \parallel \top = u \quad \text{(A3)}$$
$$u \cdot \top = \mathrm{dom}(u) \cdot \top \quad \text{(A11)}$$
$$(1 \parallel u) \cdot v = (1 \parallel u) \cdot \top \parallel v \quad \text{(A12)}$$
$$\mathrm{dom}(u \cdot v) = \mathrm{dom}(u \cdot \mathrm{dom}(v)) \quad \text{(A13)}$$
$$\mathrm{dom}(u) \cdot (v \parallel w) = \mathrm{dom}(u) \cdot v \parallel w \quad \text{(A14)}$$

Figure 2 Axioms of 2p-algebras (A1-A12) and 2pdom-algebras (A1,A2,A4-A10,A13,A14).

We consider the following signatures for terms and algebras:

$$\Sigma = \{\cdot_2, \parallel_2, \_^\circ_1, 1_0\} \qquad \Sigma_\top = \Sigma \cup \{\top_0\} \qquad \Sigma_{\mathrm{dom}} = \Sigma \cup \{\mathrm{dom}_1\}$$

We usually omit the $\cdot$ symbol and we assign priorities so that the term $(a \cdot (b^\circ)) \parallel c$ can be written just as $ab^\circ \parallel c$.

Graphs form algebras for those signatures by considering the operations depicted in Figure 1, where input and outputs are represented by unlabelled ingoing and outgoing arrows. The binary operations $(\cdot)$ and $(\parallel)$ respectively correspond to series and parallel composition, converse $(\_^\circ)$ just exchanges input and output, and *domain* $(\mathrm{dom}(\_))$ relocates the output to the input.

A graph is called a *test* if its input and output coincide. The parallel composition of a graph with a test merges the input and output of the former graph. For instance, the graph $\underline{a} \parallel 1$ consists of a single vertex with a self-loop labeled with a. Also note that the graph $\mathrm{dom}(G)$ is isomorphic to the graph $G \cdot \top \parallel 1$. For $\Sigma_\top$-terms, we will therefore consider $\mathrm{dom}(u)$ to be an abbreviation for $u\top \parallel 1$.

▶ **Definition 3.** A 2p-algebra is a $\Sigma_\top$-algebra satisfying axioms A1-A12 from Figure 2. A 2pdom-algebra is a Σ_{dom}-algebra satisfying axioms A1,A2,A4-A10,A13,A14 from Figure 2.

▶ **Lemma 4.** *Every 2p-algebra is a 2pdom-algebra (with* $\mathrm{dom}(u) \triangleq u\top \parallel 1$*).*

Proof. This easy result is implicitly proved in [16]; Coq proofs scripts are available [10]. ◀

▶ **Proposition 5.** *Graphs (up to isomorphism) form a 2p-algebra.*

▶ **Proposition 6.** *Connected graphs form a subalgebra of the* Σ_{dom}*-algebra of graphs.*

Given $\Sigma_\top$-terms u, v with variables in $\mathcal{A}$, we write $2p \vdash u = v$ when the equation is derivable from the axioms of 2p-algebra (equivalently, when the equation universally holds in all 2p-algebras). Similarly for Σ_{dom}-terms and 2pdom-algebras.

By interpreting a letter $a \in \mathcal{A}$ as the graph $\underline{a}$ in Figure 1, we can associate a graph $\mathsf{g}(u)$ to every term over the considered signatures. By Proposition 5, $2p \vdash u = v$ entails $\mathsf{g}(u) \simeq \mathsf{g}(v)$ for all $\Sigma_\top$-terms u, v and similarly for Σ_{dom}-terms and $2pdom$-algebras (using Lemma 4).

▶ **Definition 7.** A $\Sigma_\top$-term u is called a *test* if $2p \vdash u \parallel 1 = u$. A Σ_{dom}-term u is called a *test* if $2pdom \vdash u \parallel 1 = u$. We write $\mathcal{T}$ for the set of tests and $\mathcal{N}$ for the set of non-tests. We let α, β, and γ range over terms that are tests.

Thanks to converse being an involution, there is a notion of duality in 2p-algebras: a valid law remains so when swapping the arguments of products and replacing $\mathrm{dom}(u)$ with $\mathrm{dom}(u^\circ)$.

▶ **Lemma 8.** *The following laws hold in all 2pdom-algebras.*

1. $\mathrm{dom}(u) \parallel 1 = \mathrm{dom}(u)$ *(*$\mathrm{dom}(u)$ *is a test)*
2. $\alpha^\circ = \alpha$
3. $\alpha\beta = \alpha \parallel \beta = \beta\alpha$
4. $(u \parallel v)\alpha = u \parallel v\alpha$

Proof. See long version [11]. ◀

▶ **Lemma 9** ([7, Proposition 1])**.** *The following laws hold in all 2p-algebras*

1. $u\top v \parallel \top w\top = u\top w\top v$
2. $uv \parallel \top w\top = (u \parallel \top w\top)v$
3. $\top u^\circ\top = \top u\top$
4. $\alpha\top\beta \parallel u = \alpha u\beta$

▶ **Lemma 10.** *A* Σ_{dom}*- or* $\Sigma_\top$*-term* u *is a test iff* $\mathsf{g}(u)$ *is a test.*

Proof. The direction from left to right follows with Proposition 5. The converse direction follows by induction on u using the lemmas above. ◀

One useful consequence of the lemma above is that uv is a test iff both u and v are tests and $u \parallel v$ is test if either u or v is a test. Further, $\mathcal{A} \subseteq \mathcal{N}$, i.e., letters are non-tests.

We conclude this preliminary section by defining the subalgebra of treewidth-two graphs.

▶ **Definition 11.** A *simple graph* is a pair $\langle V, R\rangle$ consisting of a finite set V of vertices and an irreflexive and symmetric binary relation R on V. The *skeleton* of a graph G is the simple graph obtained from G by forgetting input, output, labeling, self loops, and edge directions and multiplicities. The *strong skeleton* of a graph is the skeleton of G with an additional edge connecting ι and o.

▶ **Definition 12** ([9])**.** Let G be a simple graph. A *tree decomposition* of G is a tree T where each node $t \in T$ is labeled with a set of vertices B_t such that:

1. For every vertex x of G, the set of nodes t such that $x \in B_t$ is nonempty and connected in T (i.e., forms a subtree)
2. For every xy-edge, there exists some t such that $\{x, y\} \subseteq B_t$.

The *width* of a tree decomposition is the size of its largest set B_t minus one, and the *treewidth* of a graph is the minimal width of a tree decomposition for this graph. The simple graphs of treewidth at most one are the forests. We write TW_2 for the collection of graphs whose strong skeleton has treewidth at most two.

▶ **Proposition 13** ([7])**.** TW_2 *forms a subalgebra of the* $\Sigma_\top$*-algebra of graphs.*

▶ **Corollary 14.** *For every term* u*,* $\mathsf{g}(u) \in \mathsf{TW}_2$*.*

The main results about 2p- and 2pdom-algebras, which we reprove in this paper, are that TW_2 (up to isomorphism) forms the free 2p-algebra (over $\mathcal{A}$) [7] and that the connected graphs in TW_2 form the free 2pdom-algebra [16]. As explained in the introduction, we start with the connected case, which we then extend to deal with disconnected graphs.

3 A Confluent Rewriting System for Term-labeled Graphs

The rewriting system we define to extract terms from graphs works on a generalised form of graphs, whose edges are labeled by terms rather than just letters, and whose vertices are labeled by tests.

We work exclusively with Σ_{dom}-terms and connected graphs in Sections 3 to 5; for these sections we thus abbreviate $2pdom \vdash u = v$ as $u \equiv v$.

▶ **Definition 15.** A *term-labeled graph* is a tuple $G = \langle V, E, s, t, l, \iota, o\rangle$ that is a graph except that l is a function from $V \uplus E$ to Σ_{dom}-terms (we assume V and E to always be disjoint) such that $l(x) \in \mathcal{T}$ for vertices x and $l(e) \in \mathcal{N}$ for edges e. We write $\exp(G)$ for the *expansion* of G obtained by replacing every edge e with $\mathsf{g}(l(e))$ and every vertex x with $\mathsf{g}(l(x))$.

Restricting edge labels to non-tests ensures that replacing edges by the graphs described by their labels does not collapse source and target of the edge. Similarly, replacing vertices by graphs is only meaningful if the replacement is a test.

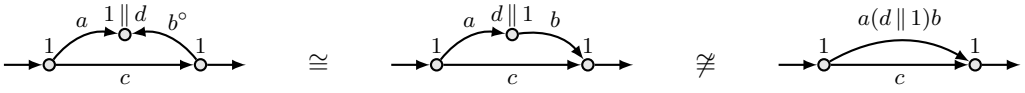
We will compare term-labeled graphs using a notion of isomorphism where labels are compared modulo 2pdom-axioms. A subtlety here is that we should consider as equivalent two graphs where one is obtained from the other by reversing a u-labeled edge and labeling it with u° (this operation preserves the expansion). The following predicate, which we use in the definitions below, captures this idea in a formal way: $L(x, y, e, u)$ means that e can be seen as a u-labeled xy-edge, up to $\equiv$.

$$L(x, y, e, u) \triangleq (s(e) = x \wedge t(e) = y \wedge l(e) \equiv u) \vee (s(e) = y \wedge t(e) = x \wedge l(e) \equiv u^\circ)$$

▶ **Definition 16.** Two term-labeled graphs $G = \langle V, E, s, t, l, \iota, o\rangle$ and $H = \langle V', E', s', t', l', \iota', o'\rangle$ are *weakly isomorphic*, written $G \cong H$, if there is a pair of bijective functions $\langle f, g\rangle$ satisfying

1. $f(\iota) = \iota'$ and $f(o) = o'$.
2. For all vertices $x \in V$, $l(x) \equiv l'(f(x))$.
3. For all edges $e \in E$ and $e' \in E'$ such that $g(e) = e'$, $L(s'(e'), t'(e'), e', l(e))$.

▶ **Example 17.** Weakly isomorphic graphs always have isomorphic expansions. However, the converse is not true: all three graphs below have isomorphic expansions, but only the first two are weakly isomorphic.



We now define the rewriting system on term-labeled graphs, as depicted in Figure 3.

▶ **Definition 18.** Let $G = \langle V, E, s, t, l, \iota, o\rangle$ be a term-labeled graph. We write $G \mapsto G'$ if G' can be obtained from G by applying one of the following rules.

1. If $l(x) = \alpha$, $L(x_1, x, e_1, u)$ and $L(x, x_2, e_2, v)$ where $x \notin \{\iota, o, x_1, x_2\}$ and e_1 and e_2 are the only incident edges of x, then replace e_1 and e_2 with an $u\alpha v$-labeled edge from x_1 to x_2 and remove x.
2. If $l(x) = \alpha$, $l(y) = \beta$ and $L(x, y, e, u)$ where $y \notin \{\iota, o\}$ and e is the only edge incident to y, then change the label of x to $\alpha{\cdot}\mathrm{dom}(u{\cdot}\beta)$ and remove y and e.
3. If $L(x, y, e_1, u)$ and $L(x, y, e_2, v)$ then replace e_1 and e_2 with a $(u \parallel v)$-labeled xy-edge.
4. If $s(e) = t(e) = x$, $l(x) = \alpha$ and $l(e) = u$, then assign label $\alpha(u \parallel 1)$ to x and remove e.

It is straightforward to verify that $\mapsto$ preserves the requirements on edge and vertex labels from Definition 15. We write $\rightsquigarrow$ for the reflexive transitive closure of $\mapsto$ up to $\cong$ (i.e., $G \rightsquigarrow H$ iff either $G \cong H$ or there exists a sequence $G \cong G_1 \mapsto G_2 \cong G_3 \mapsto \ldots \mapsto G_n \cong H$).

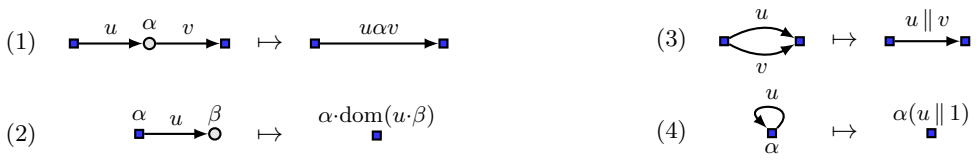


Figure 3 Rewriting system for term-labeled graphs. The square vertices may have additional incident edges. The circular vertices (i.e., those that are removed) must be distinct from input and output and may not have other incident edges.

▶ **Lemma 19.** *The relation* $\mapsto$ *is terminating.*

▶ **Lemma 20.** *If* $G \mapsto G'$*, then* $\exp(G) \simeq \exp(G')$*.*

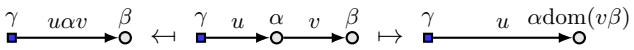
▶ **Lemma 21.** *If* $G \cong H$ *and* $G \mapsto G'$*, then there exists* H' *such that* $H \mapsto H'$ *and* $G' \cong H'$*.*

We now show that the relation $\mapsto$ is locally confluent up to weak isomorphism. The proof is fundamental: while closing the various critical pairs, we rediscover most of the axioms of 2pdom-algebras. Note that for rules (1) and (3) we do *not* assume that the square vertices are distinct. This introduces some critical pairs (e.g, between rules 3 and 4), but ensures that reductions are preserved in contexts that collapse input and output (Lemma 29 below).

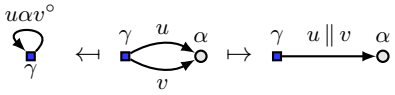
▶ **Lemma 22** (Local Confluence)**.** *If* $G_1 \leftarrow\!\shortmid G \mapsto G_2$*, then there exist* G_1' *and* G_2' *such that* $G_1 \mapsto G_1'$*,* $G_2 \mapsto G_2'$ *and* $G_1' \cong G_2'$*.*

Proof. If the redexes do not overlap, we can reduce G_1 and G_2 to the same graph in one step. It remains to analyze the critical pairs. The nontrivial interactions are as follows:

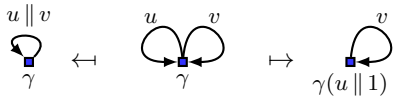
- Rules 1 and 2 can interact as follows:



  After applying rule 2 on both sides, it suffices to show $\mathrm{dom}(u\alpha v\beta) \equiv \mathrm{dom}(u\alpha\mathrm{dom}(v\beta))$, which is an instance of (A13).
- Rules 1 and 3 can interact as follows:



  After applying rule 4 on the left and rule 2 on the right, it suffices to show that we have $u\alpha v^\circ \parallel 1 \equiv \mathrm{dom}((u \parallel v)\alpha)$. We prove it as follows using Lemma 8(4) and (A10): $\mathrm{dom}((u \parallel v)\alpha) \equiv \mathrm{dom}(u \parallel v\alpha) \equiv 1 \parallel u(v\alpha)^\circ \equiv u\alpha v^\circ \parallel 1$.
- Rules 3 and 4 can interact as follows:



  After applying rule 4 on both sides, it suffices to show $u \parallel v \parallel 1 \equiv (u \parallel 1)(v \parallel 1)$. Since $v \parallel 1$ is a test, this follows with Lemma 8(4) and (A5).

There are a number of other critical pairs that can easily be resolved using (A1)-(A8) (e.g., two overlapping instances of rule 1 that differ in the direction the edges are matched, or overlapping instances of rule 3). Similarly, overlapping instances of rules 2 and 4 remain instances after the first rule has been applied and the resulting graphs only differ in the order of the tests being generated; thus they are weakly isomorphic by Lemma 8(3). ◀

▶ **Proposition 23** (Confluence)**.** *If* $G_1 \leftsquigarrow G \rightsquigarrow G_2$*, then there exists* H *such that* $G_1 \rightsquigarrow H \leftsquigarrow G_2$*.*

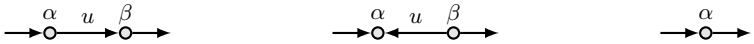


Figure 4 Atomic Graphs.

$$
\begin{aligned}
\mathsf{f}(u^\circ) &= \mathsf{match}\ \mathsf{f}(u)\ \mathsf{with} && \mathsf{f}(1) = (1)\\
&\quad (\alpha_u, u', \beta_u) \Rightarrow (\beta_u, u'^\circ, \alpha_u)\\
&\quad (\gamma) \Rightarrow (\gamma)\\
\mathsf{f}(u \parallel v) &= \mathsf{match}\ \mathsf{f}(u), \mathsf{f}(v)\ \mathsf{with} && \mathsf{f}(a) = (1, a, 1)\\
&\quad (\alpha_u, u', \beta_u), (\alpha_v, v', \beta_v) \Rightarrow (\alpha_u\alpha_v, u' \parallel v', \beta_u\beta_v)\\
&\quad (\alpha_u, u', \beta_u), (\gamma) \Rightarrow (\alpha_u u' \beta_u \parallel \gamma)\\
&\quad (\gamma), (\alpha_v, v', \beta_v) \Rightarrow (\gamma \parallel \alpha_v v' \beta_v)\\
&\quad (\gamma_1), (\gamma_2) \Rightarrow (\gamma_1\gamma_2)\\
\mathsf{f}(u{\cdot}v) &= \mathsf{match}\ \mathsf{f}(u), \mathsf{f}(v)\ \mathsf{with} && \mathsf{f}(\mathrm{dom}(u)) = (\mathrm{dom}(u))\\
&\quad (\alpha_u, u', \beta_u), (\alpha_v, v', \beta_v) \Rightarrow (\alpha_u, u'\beta_u\alpha_v v', \beta_v)\\
&\quad (\alpha_u, u', \beta_u), (\gamma) \Rightarrow (\alpha_u, u', \beta_u\gamma)\\
&\quad (\gamma), (\alpha_v, v', \beta_v) \Rightarrow (\gamma\alpha_v, v', \beta_v)\\
&\quad (\gamma_1), (\gamma_2) \Rightarrow (\gamma_1\gamma_2)
\end{aligned}
$$

Figure 5 Test analysis for Σ_{dom}-terms.

▶ **Definition 24.** We call a term-labeled graph *atomic* if it consists of either a single vertex and no edges or two vertices connected by a single edge as depicted in Figure 4. If A is atomic, we write $\underline{A}$ for the term that can extracted from A, i.e., $\alpha u \beta$, $\alpha u^\circ \beta$, or α for the atoms in Figure 4, from left to right.

▶ **Lemma 25.** *If $A \leftsquigarrow G \rightsquigarrow B$ for some atomic graphs A, B, then $\underline{A} \equiv \underline{B}$.*

Proof. We have $A \cong B$ by Proposition 23, since atomic graphs are irreducible. The claim then follows by case analysis on A and B. ◀

4 Reducibility of Term-Graphs

We now show that the rewriting system from the previous section can be used to reduce graphs of the shape $\mathsf{g}(u)$ to atomic graphs. As a consequence, we obtain that $u \equiv v$ iff $\mathsf{g}(u) \simeq \mathsf{g}(v)$ and, hence, that equivalence of Σ_{dom}-terms is decidable.

To show that $\mathsf{g}(u)$ reduces to an atomic graph, we define a function computing for every u an equivalent term that can be obtained as $\underline{A}$ for some atomic graph A. In particular, if u is not a test, it computes "maximal" tests α and β and a non-test v such that $u \equiv \alpha v \beta$.

▶ **Definition 26.** We define a function f from Σ_{dom}-terms to $\mathcal{T} \cup \mathcal{T} \times \mathcal{N} \times \mathcal{T}$ as depicted in Figure 5, as well as functions $\lceil \_ \rceil$ and $\lfloor \_ \rfloor$ interpreting elements of $\mathcal{T} \cup \mathcal{T} \times \mathcal{N} \times \mathcal{T}$ as atomic graphs and terms, respectively: $\lceil (\alpha, u, \beta) \rceil$ is the graph on the left in Figure 4 and $\lceil (\alpha) \rceil$ is the graph on the right, $\lfloor (\alpha, u, \beta) \rfloor \triangleq \alpha u \beta$, and $\lfloor (\gamma) \rfloor \triangleq \gamma$. Note that $\underline{\lceil (\alpha, u, \beta) \rceil} = \lfloor (\alpha, u, \beta) \rfloor$.

A summary of the functions defined so far is given in Figure 6.

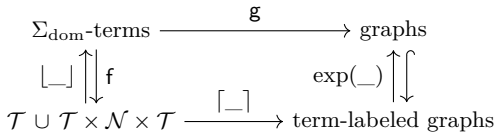


Figure 6 Summary of functions between terms and graphs.

▶ **Lemma 27.** $u \equiv \lfloor \mathsf{f}(u) \rfloor$.

Proof. By induction on u. The cases for a, 1, and $\mathrm{dom}(u)$ are trivial. The case for u° follows with Lemma 8(2). We show the case for $\mathsf{f}(u \parallel v)$ where $\mathsf{f}(u) = (\alpha_u, u', \beta_u)$ and $\mathsf{f}(v) = (\alpha_v, v', \beta_v)$.

$$\begin{aligned}
\lfloor \mathsf{f}(u \parallel v) \rfloor &= \alpha_u \alpha_v (u' \parallel v') \beta_u \beta_v \\
&\equiv (\alpha_u u' \beta_u) \parallel (\alpha_v v' \beta_v) && \text{Lemma 8(4) and commutativity of } \parallel \\
&\equiv \lfloor \mathsf{f}(u) \rfloor \parallel \lfloor \mathsf{f}(v) \rfloor \\
&\equiv u \parallel v && \text{induction hypothesis}
\end{aligned}$$

The remaining cases are straightforward. ◀

We now show that $\mathsf{g}(u)$ (seen as a term-labeled graph) reduces to $\lceil \mathsf{f}(u) \rceil$. To do so, we first extend the graph operations to term-labeled graphs and prove the context lemma below.

▶ **Definition 28.** If a graph G occurs as a term-labeled graph, it is to be read as the graph where every vertex is labeled with 1 (and every edge is labeled with a single letter as before). We extend the operations $\cdot$, $\parallel$ and $\mathrm{dom}(\_)$ to term-labeled graphs. If two vertices x and y are identified by an operation, we label the resulting vertex with $l(x) \cdot l(y)$.

▶ **Lemma 29** (Context Lemma). *If* $G \rightsquigarrow G'$*, then* $G \parallel H \rightsquigarrow G' \parallel H$*,* $G \cdot H \rightsquigarrow G' \cdot H$*,* $H \cdot G \rightsquigarrow H \cdot G'$*, and* $\mathrm{dom}(G) \rightsquigarrow \mathrm{dom}(G')$.

Proof. By induction on $G \rightsquigarrow G'$. First, all operations preserve weak isomorphisms. Second, a redex in G is still a redex in $G \cdot H$, $H \cdot G$, and $\mathrm{dom}(G)$ since G remains unchanged except that one of its nodes may cease to be input or output. Similarly, a redex in G is also a redex in $G \parallel H$ (even if H is a test: we do not require the square vertices in rules 1 and 3 to be distinct so that redexes are preserved under collapsing input and output). ◀

Note that the converse of Lemma 29 does not hold. For instance, $\mathrm{dom}(\underline{a})$ (cf. Figure 1) reduces by rule 2 to a graph G with a single node labeled $1\mathrm{dom}(a1)$. Hence, $\mathrm{dom}(\underline{a}) \rightsquigarrow \mathrm{dom}(G)$ since $G \cong \mathrm{dom}(G)$, but $\underline{a}$ is an atom and thus irreducible.

▶ **Proposition 30** (Reducibility). $\mathsf{g}(u) \rightsquigarrow \lceil \mathsf{f}(u) \rceil$

Proof. By induction on u. The base cases are trivial. For the inductive cases, we use Lemma 29 and the induction hypothesis to reduce the respective subgraphs to atomic graphs. The resulting graphs always reduce to atomic graphs in a single step (cf. [11]). ◀

We can finally characterise the equational theory of 2pdom-algebras:

▶ **Theorem 31.** $2pdom \vdash u = v$ *iff* $\mathsf{g}(u) \simeq \mathsf{g}(v)$.

Proof. The direction from left to right follows with Proposition 5. For the converse direction, assume $\mathsf{g}(u) \simeq \mathsf{g}(v)$. Then $\mathsf{g}(u) \cong \mathsf{g}(v)$ and therefore $\underline{\lceil \mathsf{f}(u) \rceil} \equiv \underline{\lceil \mathsf{f}(v) \rceil}$ by Proposition 30 and Lemma 25. Hence, $u \equiv \lfloor \mathsf{f}(u) \rfloor = \underline{\lceil \mathsf{f}(u) \rceil} \equiv \underline{\lceil \mathsf{f}(v) \rceil} = \lfloor \mathsf{f}(v) \rfloor \equiv v$ using Lemma 27 twice. ◀

As explained in the introduction we did not use minors to obtain this result. Actually, we did not use tree decompositions either: those arise only in the following section, where we need to characterize the image of the function g. This sharply contrasts with the approach from [7], where both tree decompositions and minors are used to obtain the above characterization.

5 The free 2pdom-algebra

In order to show that the connected graphs in TW_2 form the free 2p-algebra, it remains to obtain an inverse to g (up to $\equiv$), i.e., we need to extract terms from such graphs. We again make use of the rewriting system.

In a slight abuse of notation, we also write $G \in \mathsf{TW}_2$ to denote that (the strong skeleton of) a term-labeled graph G has treewidth at most two.

▶ **Lemma 32** (Preservation). *If $G \in \mathsf{TW}_2$ is a connected term-labeled graph and $G \mapsto G'$, then $G' \in \mathsf{TW}_2$ and G' is connected.*

▶ **Lemma 33** (Progress). *If $G \in \mathsf{TW}_2$ is a connected term-labeled graph, then either there exists some G' such that $G \mapsto G'$ or G is atomic.*

Proof. W.l.o.g., we can assume that rules 3 and 4 do not apply. Thus, it suffices to show that either ι and o are the only vertices of G or that there is some vertex distinct from input and output that has at most two neighbors. Let T be a tree decomposition of the strong skeleton of G of width at most two, and remove leafs of T that are included in their unique neighbor (T remains a tree-decomposition). If T has only one node (say t) then $\{\iota, o\} \subseteq B_t$. Hence, if there is another vertex, it has degree at most two. Otherwise, let t be a leaf and let z be a vertex appearing only in B_t. Without loss of generality, we can assume $z \notin \{\iota, o\}$. (If $z = \iota$ then $o \in B_t$, due to the ιo-edge in the strong skeleton; hence, for any other leaf, neither ι nor o can be the vertex unique to that leaf.) Since z appears only on B_t it has at most two neighbors. ◀

▶ **Definition 34.** We define a function t' from connected term-labeled graphs of treewidth at most two to terms as follows: $\mathsf{t}'(G) \triangleq \underline{A}$ for some atomic graph such that $G \mapsto^* A$. A suitable atomic graph A can be computed by blindly applying the rules (Lemmas 32 and 33): all choices lead to equivalent terms (Lemma 25). For connected (standard) graphs G, we write $\mathsf{t}(G)$ for $\mathsf{t}'(G')$ where G' is G seen as a term-labeled graph.

▶ **Lemma 35.** *If $G \in \mathsf{TW}_2$ is a term-labeled graph and $G \rightsquigarrow H$, then $\mathsf{t}'(G) \equiv \mathsf{t}'(H)$.*

Proof. Follows with Lemma 25. ◀

As an immediate consequence of the lemma above we also have:

▶ **Proposition 36.** *If $G, H \in \mathsf{TW}_2$ are are connected and $G \simeq H$, then $\mathsf{t}(G) \equiv \mathsf{t}(H)$.*

We now show that t and g are inverses up to term equivalence and isomorphism respectively.

▶ **Proposition 37.** *For all Σ_{dom}-terms u, $\mathsf{t}(\mathsf{g}(u)) \equiv u$.*

Proof. We have $\mathsf{t}(\mathsf{g}(u)) \equiv \underline{\lceil \mathsf{f}(u) \rceil} = \lfloor \mathsf{f}(u) \rfloor$ by Proposition 30 and Lemma 35. The claim then follows with Lemma 27. ◀

▶ **Proposition 38.** *If $G \in \mathsf{TW}_2$ is connected, then $\mathsf{g}(\mathsf{t}(G)) \simeq G$.*

Proof. We have $\mathsf{t}(G) = \underline{A}$ for some A such that $G \mapsto^* A$. Hence, $\exp(A) \simeq G$ (Lemma 20). The claim follows since $\mathsf{g}(\underline{A}) \simeq \exp(A)$ for all atoms A. ◀

The function g is a Σ_{dom}-homomorphism by definition. By the above results, this is actually an isomorphism between the $2pdom$-algebra of connected graphs in TW_2 and the (canonically) free $2pdom$-algebra of Σ_{dom}-terms quotiented by $\equiv$:

▶ **Theorem 39** ([16]). *The connected graphs in TW_2 (with labels in $\mathcal{A}$) form the free 2pdom-algebra (over $\mathcal{A}$).*

6 The free 2p-algebra

We now extend the results from the previous section to disconnected graphs. That is, we show that the class of all graphs in TW_2 forms the free 2p-algebra [7]. We use for that the previous function t to extract terms from the various connected components of a graph.

In this section, we take $u \equiv v$ to mean $2p \vdash u = v$. Recall that $2p \vdash u = v$ whenever $2pdom \vdash u = v$ (Lemma 4). Hence, all the lemmas from the previous section still apply.

▶ **Definition 40.** Let G be a graph. For vertices x, y of G, we write $G[x, y]$ for the graph G with input set to x and output set to y. We abbreviate $G[x, x]$ as $G[x]$. Further, we write G_x for the connected component of x (as a subgraph of G, with input and output set to x).

▶ **Definition 41.** Let $\mathcal{C}(G)$ be the collection of components G_x obtained by choosing some vertex x for every connected component of G containing neither ι nor o. We define a function $\mathsf{t}^\top$ extracting terms from (possibly disconnected) graphs as follows:

$$c_G \triangleq \mathop{\Big\|}_{H \in \mathcal{C}(G)} \top\cdot\mathsf{t}(H)\cdot\top \qquad \mathsf{t}^\top(G) \triangleq \begin{cases} \mathsf{t}(G_\iota)\cdot\top\cdot\mathsf{t}(G_o) \parallel c_G & \iota \text{ and } o \text{ disconnected} \\ \mathsf{t}(G_\iota[\iota, o]) \parallel c_G & \iota \text{ and } o \text{ connected} \end{cases}$$

Note that the function $\mathsf{t}^\top$ needs to choose shared input/outputs vertices for all disconnected components. For isomorphic arguments, these choices can differ. We begin by showing that this choice does not matter up to term equivalence.

▶ **Lemma 42.** *Let $G \in \mathsf{TW}_2$ be a connected test and let x be a neighbor of ι in G. We have $\mathsf{t}(G)\cdot\top \equiv \mathsf{t}(G[\iota, x])\cdot\top$.*

Proof. Since $G \in \mathsf{TW}_2$, so is $G[\iota, x]$. Hence, $G[\iota, x] \rightsquigarrow \lceil(\alpha, u, \beta)\rceil$ for some terms α,β, and u. Since $G = \mathrm{dom}(G[\iota, x])$, we also have $G \rightsquigarrow \mathrm{dom}(\lceil(\alpha, u, \beta)\rceil)$ by Lemma 29. Moreover, $\mathrm{dom}(\lceil(\alpha, u, \beta)\rceil) \rightsquigarrow \lceil(\alpha\cdot\mathrm{dom}(u\beta))\rceil$ by rule 2. Using Lemma 35 and (A11), we have: $\mathsf{t}(G)\cdot\top \equiv \mathsf{t}'(\lceil(\alpha\cdot\mathrm{dom}(u\beta))\rceil)\cdot\top = (\alpha\cdot\mathrm{dom}(u\beta))\cdot\top \equiv \alpha u\beta\cdot\top = \mathsf{t}'(\lceil(\alpha, u, \beta)\rceil)\cdot\top \equiv \mathsf{t}(G[\iota, x])\cdot\top$ ◀

▶ **Lemma 43.** *Let $G \in \mathsf{TW}_2$ be a connected graph and let x, y be vertices of G. We have $\top\cdot\mathsf{t}(G[x])\cdot\top \equiv \top\cdot\mathsf{t}(G[y])\cdot\top$*

Lemma 43 follows by repeatedly applying Lemma 42 along some xy-path. Both lemmas also appear in [7]. We remark that the proof of Lemma 42 given here, which depends on the definition of t, is considerably simpler than the one in [7]. The proof of Lemma 43 remains essentially unchanged.

▶ **Proposition 44.** *Let $G, H \in \mathsf{TW}_2$. If $G \simeq H$, then $\mathsf{t}^\top(G) \equiv \mathsf{t}^\top(H)$.*

Proof. Follows with Proposition 36 and Lemma 43. ◀

▶ **Proposition 45.** $\mathsf{g}(\mathsf{t}^\top(G)) \simeq G$.

▶ **Lemma 46.** $\mathsf{t}^\top$ *is a homomorphism of 2p-algebras.*

Proof. We already showed that that $\mathsf{t}^\top$ respects graph isomorphisms. It remains to show that $\mathsf{t}^\top$ commutes with all operations.

We show $\mathsf{t}^\top(G{\cdot}H) \equiv \mathsf{t}^\top(G){\cdot}\mathsf{t}^\top(H)$. Let $F \triangleq G{\cdot}H$. We distinguish four cases based on whether ι and o are connected in G and H respectively.

ι and o disconnected in both G and H: In that case, G_o and H_ι are merged into one component of F that is connected neither to the input nor to the output of F. By Lemma 43 and Proposition 36 we therefore have: $c_F \equiv (\top\mathsf{t}(G_o{\cdot}H_\iota)\top) \parallel c_G \parallel c_H$. We reason as follows:

$$\begin{aligned}
\mathsf{t}^\top(F) &\equiv \mathsf{t}(F_\iota){\cdot}\top{\cdot}\mathsf{t}(F_o) \parallel c_F && \iota \text{ and } o \text{ disconnected in F}\\
&\equiv \mathsf{t}(G_\iota){\cdot}\top{\cdot}\mathsf{t}(H_o) \parallel \top{\cdot}\mathsf{t}(G_o{\cdot}H_\iota){\cdot}\top \parallel c_G \parallel c_H && F_\iota \simeq G_\iota,\ F_o \simeq G_o\\
&\equiv \mathsf{t}(G_\iota){\cdot}\top{\cdot}\mathsf{t}(G_o{\cdot}H_\iota){\cdot}\top{\cdot}\mathsf{t}(H_o) \parallel c_G \parallel c_H && \text{Lemma 9(1)}\\
&\equiv \mathsf{t}(G_\iota){\cdot}\top{\cdot}\mathsf{t}(G_o){\cdot}\mathsf{t}(H_\iota){\cdot}\top{\cdot}\mathsf{t}(H_o) \parallel c_G \parallel c_H && \mathsf{t} \text{ is a homomorphim}\\
&\equiv \mathsf{t}^\top(G){\cdot}\mathsf{t}^\top(H) && \text{Lemma 9(2) and its dual}
\end{aligned}$$

ι and o connected in G but not in H: We have $c_F \equiv c_G \parallel c_H$ by Prop. 36 and Lemma 43.

$$\begin{aligned}
\mathsf{t}^\top(F) &\equiv \mathsf{t}(F_\iota){\cdot}\top{\cdot}\mathsf{t}(F_o) \parallel c_F\\
&\equiv \mathsf{t}(\mathrm{dom}(G_\iota[\iota,o]{\cdot}H_\iota)){\cdot}\top{\cdot}\mathsf{t}(H_o) \parallel c_F && F_\iota \simeq \mathrm{dom}(G_\iota[\iota,o]\cdot H_\iota),\ F_o \simeq H_o\\
&\equiv \mathsf{t}(G_\iota[\iota,o]){\cdot}\mathsf{t}(H_\iota){\cdot}\top{\cdot}\mathsf{t}(H_o) \parallel c_F && \text{(A11), } \mathsf{t} \text{ is a homomorphism}\\
&\equiv \mathsf{t}(G_\iota[\iota,o]){\cdot}\mathsf{t}(H_\iota){\cdot}\top{\cdot}\mathsf{t}(H_o) \parallel c_G \parallel c_H\\
&\equiv \mathsf{t}^\top(G){\cdot}\mathsf{t}^\top(H) && \text{Lemma 9(2) and its dual}
\end{aligned}$$

The case where input and output are connected only in H is symmetric and the case where they are connected in both graphs follows from t being a homomorphism.

Proving that $\mathsf{t}^\top$ commutes with the other operations is done in a similar manner. ◀

▶ **Proposition 47.** *For all $\Sigma_\top$-terms u, $\mathsf{t}^\top(\mathsf{g}(u)) \equiv u$.*

Proof. By induction on u, using Lemma 46. ◀

▶ **Theorem 48** ([7])**.** *The graphs in TW_2 (with labels in $\mathcal{A}$) form the free 2p-algebra (over $\mathcal{A}$).*

7 1-free 2p-algebras

We now show that the techniques from the previous sections can be adapted to the setting where 1 (and hence dom(_)) are removed from the signature. We define algebras over the signature $\Sigma_\top^{-1} \triangleq \Sigma_\top \setminus \{1\}$, which we call 1-free 2p-algebras, and show that the graphs of treewidth at most two without self-loops and with distinct input and output form the free 1-free 2p-algebra (over $\mathcal{A}$).

The axioms for 1-free 2p-algebras are A1-A4,A6-A8 plus the following three axioms:

$$(u{\cdot}v \parallel w){\cdot}\top = (u \parallel w{\cdot}v^\circ){\cdot}\top \tag{A15}$$

$$u{\cdot}(v{\cdot}\top \parallel w) = (u \parallel \top{\cdot}v^\circ){\cdot}w \tag{A16}$$

$$u{\cdot}v \parallel \top{\cdot}w = u{\cdot}(v \parallel \top{\cdot}w) \tag{A17}$$

The main complication in adapting our techniques to the 1-free case is that the syntax of 1-free 2p-algebras cannot express tests, even though the algebra of graphs still exhibits tests-like structures. For instance, if G is a test without self-loops, then $G{\cdot}\top$ is a graph of the proposed free 1-free 2p-algebra. To account for this, we distinguish between the type of $\Sigma_{\top}^{-1}$-terms, written Tm, and a type of *(syntactic) tests* defined as follows:

$$\alpha \in \mathsf{Tst} ::= 1 \mid [u] \qquad (u \in \mathsf{Tm})$$

Tests, which are not terms, allow us to describe graphs that are tests. We let $\alpha,\beta,\ldots$ range over tests, and we extend the definition of g to tests by setting $\mathsf{g}(1) = 1$ and $\mathsf{g}([u]) = \mathrm{dom}(\mathsf{g}(u))$. Intuitively, in a test $[u]$, the output of the term u does not matter: u will always be used in contexts where this information disappears, e.g., as in $u\top$; this allows us to treat $[u]$ essentially like $\mathrm{dom}(u)$. It also motivates the following notion of equivalence for tests: $1 \equiv 1$, and $[u] \equiv [v]$ if $u{\cdot}\top \equiv v{\cdot}\top$.

▶ **Lemma 49.** *If $\alpha \equiv \beta$ then $\mathsf{g}(\alpha) \simeq \mathsf{g}(\beta)$.*

We extend the sequential composition to take one or two tests as arguments in a manner that 1 is the neutral element on both sides:

$$\begin{array}{lll} 1{\cdot}v \triangleq v & u{\cdot}1 \triangleq u & 1{\cdot}\alpha \triangleq \alpha \\ [u]{\cdot}v \triangleq u\top \parallel v & u{\cdot}[v] \triangleq u \parallel \top v^\circ & [u]{\cdot}1 \triangleq [u] \\ & & [u]{\cdot}[v] \triangleq [u\top \parallel v] \end{array}$$

Note that $\alpha{\cdot}\beta$ is a test whereas all other variants are terms. The three operations above appropriately preserve test equivalence (e.g., if $\alpha \equiv \beta$, then $u\alpha \equiv u\beta$, $\alpha u \equiv \beta u$, $\gamma\alpha \equiv \gamma\beta$, $\alpha\gamma \equiv \beta\gamma$ for all terms u and tests γ.

The definitions above essentially yield a 2-sorted extension of 1-free 2p-algebras. We prove various laws, including all axioms of 2pdom-algebras that can still be expressed in the 2-sorted setting (using $[\_]$ instead of $\mathrm{dom}(\_)$). Examples of laws that cannot be expressed in the 2-sorted setting are $\mathrm{dom}(\alpha) \equiv \alpha$, and $\mathrm{dom}(u \parallel v) \equiv 1 \parallel uv^\circ$.

▶ **Lemma 50.** *We have the following equivalences:*

1. $u\top \equiv [u]\top$ *and* $\top u \equiv \top[u^\circ]$.
2. $(\alpha u)^\circ \equiv u^\circ\alpha$, $(u\alpha)^\circ \equiv \alpha u^\circ$.
3. $(xy)z \equiv x(yz)$ *(for all x, y, z either test or term).*
4. $[uv] \equiv [u[v]]$
5. $\alpha\beta \equiv \beta\alpha$
6. $\alpha(v \parallel w) \equiv \alpha v \parallel w$.
7. $[uv \parallel w] \equiv [u \parallel wv^\circ]$.

Proof. For all statements involving tests α of unknown shape, we distinguish the cases $\alpha = 1$ (usually trivial) and $\alpha = [w]$ for some w. Claims (1) and (2) are straightforward. By (2) and the laws for converse, we only need to consider 5 of the 8 cases of (3). $(u\alpha)v \equiv u(\alpha v)$ follows with (A16) and $(uv)\alpha \equiv u(\alpha v)$ follows with (A17). For $(\alpha\beta)\gamma \equiv \alpha(\beta\gamma)$ we repeatedly use (A15) with $v = \top$. The remaining cases for associativity are straightforward. Claims (4) and (5) follow with associativity. Claim (6) follows with (A1). Claim (7) follows with (A15). ◀

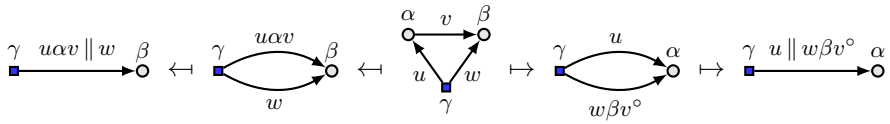
Having recovered most of the laws of 2pdom-algebras, we adapt the rewriting system for 2pdom-algebras (Figure 3) to the 1-free case. We define term-labeled graphs as for 2pdom, with the difference that now vertices are labeled with syntactic tests and edges are labeled with $\Sigma_{\top}^{-1}$-terms (whose graphs are never tests). The rewriting system on term-labeled graphs (Figure 3) is adapted by replacing $\mathrm{dom}(u)$ with $[u]$, removing rule 4, and restricting rules 1

and 3 such that the two outer vertices must be distinct. For rule 1, this is necessary to avoid introducing self loops.

Local confluence adapts, although for one of the pairs we now need two reduction steps to join the two alternatives.

▶ **Lemma 51** (Local Confluence). *If $G_1 \leftarrow\!\shortmid\, G \mapsto G_2$, then there exist G_1' and G_2' such that $G_1 \rightsquigarrow G_1'$, $G_2 \rightsquigarrow G_2'$ and $G_1' \cong G_2'$.*

Proof. The only interesting (new) critical pair is that of overlapping instances of rule 1, where the outer nodes are the same. Due to the restriction that the outer nodes of rule 1 must be distinct, this pair can no longer be joined by applying rule 1. Instead we use rules 3 and 2 as follows:



After applying rule 2 on both sides, it suffices to show $[(u\alpha v \parallel w)\beta] \equiv [(u \parallel w\beta v^\circ)\alpha]$. This follows with Lemma 50(6+7). ◀

In order to adapt Proposition 30, we need to restrict to terms u such that $\mathsf{g}(u)$ is connected. We write Tm' and Tst' for the set of terms and tests respectively, where tests and the extended sequential composition are treated as primitive and $\top$ does not occur. We then employ a function $\mathsf{f} : \mathsf{Tm}' \to \mathsf{Tst}' \times \mathsf{Tm}' \times \mathsf{Tst}'$ that can be seen as a type directed variant of the function in Figure 5.

$$
\begin{aligned}
\mathsf{f}(a) &= (1, a, 1)\\
\mathsf{f}(u^\circ) &= \mathsf{let}\ (\alpha_u, u', \beta_u) := \mathsf{f}(u)\ \mathsf{in}\ (\beta_u, u'^\circ, \alpha_u)\\
\mathsf{f}(u \parallel v) &= \mathsf{let}\ (\alpha_u, u', \beta_u), (\alpha_v, v', \beta_v) := \mathsf{f}(u), \mathsf{f}(v)\ \mathsf{in}\ (\alpha_u\alpha_v, u' \parallel v', \beta_u\beta_v)\\
\mathsf{f}(u{\cdot}v) &= \mathsf{let}\ (\alpha_u, u', \beta_u), (\alpha_v, v', \beta_v) := \mathsf{f}(u), \mathsf{f}(v)\ \mathsf{in}\ (\alpha_u, u'\beta_u\alpha_v v', \beta_v)\\
\mathsf{f}(\gamma{\cdot}u) &= \mathsf{let}\ (\alpha_u, u', \beta_u) := \mathsf{f}(u)\ \mathsf{in}\ (\gamma\alpha_u, u', \beta_u)\\
\mathsf{f}(u{\cdot}\gamma) &= \mathsf{let}\ (\alpha_u, u', \beta_u) := \mathsf{f}(u)\ \mathsf{in}\ (\alpha_u, u', \beta_u\gamma)
\end{aligned}
$$

▶ **Lemma 52.** *If $u \in \mathsf{Tm}'$ and $\alpha \in \mathsf{Tst}'$, then $\mathsf{g}(u) \rightsquigarrow \lceil \mathsf{f}u \rceil$ and $\mathsf{g}(\alpha) \rightsquigarrow \lceil (\alpha) \rceil$.*

Proof. We have a context lemma similar to Lemma 29. The proof then proceeds by mutual induction on u and α. The cases correspond to those of Proposition 30. ◀

We define two extraction functions t_1 and t_2, where t_1 extracts syntactic tests (in Tst') from graphs that are tests and t_2 extracts terms (in Tm') from non-tests. Both functions are defined just like t (Definition 34), exploiting the fact that the rewriting system does not merge or delete input and output. Propositions 37 and 38 then adapt without conceptual changes.

▶ **Proposition 53.**

1. $\mathsf{t}_1(\mathsf{g}(\alpha)) \equiv \alpha$ *for all* $\alpha \in \mathsf{Tst}'$ *and* $\mathsf{t}_2(\mathsf{g}(u)) \equiv u$ *for all* $u \in \mathsf{Tm}'$.
2. *If* $G \in \mathsf{TW}_2$ *is a connected test without self loops, then* $\mathsf{g}(\mathsf{t}_1(G)) \simeq G$.
3. *If* $G \in \mathsf{TW}_2$ *is a connected non-test without self loops, then* $\mathsf{g}(\mathsf{t}_2(G)) \simeq G$.

Using t_1 and t_2, we define a variant of $\mathsf{t}^\top$ extracting $\Sigma_\top^{-1}$-terms from non-tests without self-loops.

▶ **Definition 54.** Let $\mathcal{C}(G)$ as in Definition 41. We define

$$c_G \triangleq \mathop{\Big\|}_{H \in \mathcal{C}(G)} \top \cdot \mathsf{t}_1(H) \cdot \top \qquad \mathsf{t}^\top(G) \triangleq \begin{cases} \mathsf{t}_1(G_\iota) \cdot \top \cdot \mathsf{t}_1(G_o) \parallel c_G & \iota \text{ and } o \text{ disconnected} \\ \mathsf{t}_2(G_\iota[\iota, o]) \parallel c_G & \iota \text{ and } o \text{ connected} \end{cases}$$

That $\mathsf{t}^\top$ respects graph isomorphisms is immediate with Proposition 53. To show $\mathsf{t}^\top$ is a homomorphism of 1-free 2p-algebras, we require a 2-sorted analog to Lemma 9.

▶ **Lemma 55.** *We have the following equivalences:*

1. $\top u^\circ \top \equiv \top u \top$.
2. $\alpha v \equiv \alpha \top \parallel v$.
3. $\alpha \top \beta \equiv \alpha \top \parallel \top \beta$
4. $uv \parallel \top \alpha \top \equiv (u \parallel \top \alpha \top) v$
5. $\alpha \top \beta \parallel \top \gamma \top \equiv \alpha \top \gamma \top \beta$

Note that, due to Lemma 50(1), any equivalence where a test α appears either only as $\alpha\top$ or only as $\top\alpha$ (i.e., in α in Lemma 55(4)), also holds if α is replaced by a term.

The remaining proofs of Section 6 adapt to the 1-free setting by carefully distinguishing between terms and tests, but without any conceptual changes. For instance, we have $\mathsf{t}_1(G)\top \equiv \mathsf{t}_2(G[\iota, x])\top$ for neighbors x of ι.

▶ **Theorem 56.** *The graphs (with labels in $\mathcal{A}$) of treewidth at most two, with distinct input and output, and without self-loops form the free 1-free 2p-algebra (over $\mathcal{A}$).*

The axioms we listed for 1-free 2p-algebras are precisely those needed to prove the 2-sorted 2pdom- and 2p-laws. The proofs of Lemmas 50 and 55 have been verified in the Coq proof assistant [6]. We also used the model generator Mace4 [15] to verify that the axioms of 1-free 2p-algebras are independent. The corresponding scripts can be downloaded from [10].

8 Conclusion and directions for future work

We have proved that graphs in TW_2, connected graphs in TW_2, and self-loop free graphs in TW_2 with distinct input and output respectively form the free 2p-algebra, the free 2pdom-algebra, and the free 1-free 2p-algebra.

To do so, we used a graph rewriting system that makes it possible to extract terms from connected graphs in TW_2, in a bottom-up fashion. This technique is much easier than the one used in [7] in that it is more local and does not require us to study the precise structure of graphs in TW_2 (i.e., through excluded minors).

As explained in the introduction, the result about connected graphs can be reduced to the one about arbitrary graphs by model-theoretic means: one can easily embed a 2pdom-algebra into a 2p-algebra [16], so that 2p-algebras form a conservative extension of 2pdom-algebras. As a corollary of Theorem 56, we get that 2p-algebras also form a conservative extension of 1-free 2p-algebras. It is however unclear how to prove this result directly, by model-theoretic means: terms which are missing in 1-free 2p-algebras (self-loops) can occur deep inside terms of 2p-algebras, unlike terms which are missing in 2pdom-algebras (disconnected components).

As a natural follow-up to this work, we would like to study whether one can characterize the classes of graphs of higher treewidth as free algebras. The present approach seems promising for treewidth at most three: a reasonable rewriting system is known for recognising such graphs [3]. In contrast, trying to exploit the four excluded minors known to characterize treewidth three [4, 3] seems extremely difficult. For larger treewidth, rewriting systems recognizing graphs of a given treewidth can be shown to exist [1]. However, the result is nonconstructive in the same way as the existence of a finite set of excluded minors for each treewidth [17]).

References

1 S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. *Journal of the ACM*, 40(5):1134–1164, 1993. doi:10.1145/174147.169807.

2 S. Arnborg and A. Proskurowski. Canonical representations of partial 2- and 3-trees. In *SAWT*, pages 310–319. Springer, 1990. URL: http://dl.acm.org/citation.cfm?id=88723.88787.

3 Stefan Arnborg and Andrzej Proskurowski. Characterization and recognition of partial 3-trees. *SIAM J. Algebraic Discrete Methods*, 7(2):305–314, 1986. doi:10.1137/0607033.

4 Stefan Arnborg, Andrzej Proskurowski, and Derek G. Corneil. Forbidden minors characterization of partial 3-trees. *Discrete Mathematics*, 80(1):1–19, 1990. doi:10.1016/0012-365X(90)90292-P.

5 Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theoretical Computer Science*, 239(2):211–229, 2000. doi:10.1016/S0304-3975(99)00220-0.

6 Coq team. The Coq proof assistant. URL: https://coq.inria.fr/.

7 Enric Cosme-Llópez and Damien Pous. K_4-free graphs as a free algebra. In *MFCS*, volume 83 of *LIPIcs*. Schloss Dagstuhl, 2017. doi:10.4230/LIPIcs.MFCS.2017.76.

8 B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge Univ. Press, 2012.

9 R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2005.

10 Christian Doczkal and Damien Pous. Supplementary material accompanying this paper. URL: https://perso.ens-lyon.fr/damien.pous/covece/tw2rw.

11 Christian Doczkal and Damien Pous. Treewidth-two graphs as a free algebra. Full version of this extended abstract, with all proofs, 2018. URL: https://hal.archives-ouvertes.fr/hal-01780844.

12 R.J Duffin. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, 10(2):303–318, 1965. doi:10.1016/0022-247X(65)90125-3.

13 Eugene C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *NCAI*, pages 4–9. AAAI Press / The MIT Press, 1990. URL: http://www.aaai.org/Library/AAAI/1990/aaai90-001.php.

14 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1):1:1–1:24, 2007. doi:10.1145/1206035.1206036.

15 W. McCune. Prover9 and Mace4, 2005–2010. URL: http://www.cs.unm.edu/~mccune/prover9/.

16 Damien Pous and Valeria Vignudelli. Allegories: decidability and graph homomorphisms, 2018. to appear in Proc. LiCS 2018. URL: https://hal.archives-ouvertes.fr/hal-01703906/.

17 Neil Robertson and P.D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. doi:10.1016/j.jctb.2004.08.001.

On Pseudodeterministic Approximation Algorithms

Peter Dixon[1]
Iowa State University, Ames, USA
tooplark@iastate.edu

A. Pavan[2]
Iowa State University, Ames, USA
pavan@cs.iastate.edu

N. V. Vinodchandran[3]
University of Nebraska, Lincoln, USA
vinod@cse.unl.edu

Abstract

We investigate the notion of *pseudodeterminstic approximation algorithms*. A randomized approximation algorithm A for a function f is *pseudodeterministic* if for every input x there is a *unique value* v so that $A(x)$ outputs v with high probability, and v is a good approximation of $f(x)$. We show that designing a pseudodeterministic version of Stockmeyer's well known approximation algorithm for the NP-membership counting problem will yield a new circuit lower bound: if such an approximation algorithm exists, then for every k, there is a language in the complexity class $\mathrm{ZPP}^{\mathrm{NP}}_{tt}$ that does not have n^k-size circuits. While we do not know how to design such an algorithm for the NP-membership counting problem, we show a general result that any randomized approximation algorithm for a counting problem can be transformed to an approximation algorithm that has a *constant number* of *influential* random bits. That is, for most settings of these influential bits, the approximation algorithm will be pseudodeterministic.

2012 ACM Subject Classification Theory of computation → Probabilistic computation, Theory of computation → Circuit complexity

Keywords and phrases Approximation Algorithms, Circuit lower bounds, Pseudodeterminism

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.61

1 Introduction

Consider the computational problem: Given an input 1^n, generate an n-bit prime. Is there a *deterministic polynomial-time* algorithm for this problem? Even though primality testing can be done in deterministic polynomial time, we do not know whether a deterministic polynomial-time algorithm exists for this problem. However, a straightforward *probabilistic algorithm* exists for this task: Randomly pick an n-bit integer and output it if it is a prime. The density of primes implies that this algorithm succeeds with inverse polynomial probability. Hence by repeating this procedure polynomially many times, we can output an n-bit prime number with very high probability. A drawback of this algorithm is that the output of the algorithm depends on the random choices it makes. In particular, on two different random choices the algorithm may output two different prime numbers. A natural question to ask is:

[1] Research Supported in part by NSF grant 1421163.
[2] Research Supported in part by NSF grant 1421163.
[3] Research Supported in part by NSF grant 1422668.

is there a randomized, polynomial-time algorithm that on input 1^n outputs a *unique n-bit prime number* on most random choices. Such an algorithm is called a *pseudodeterministic algorithm*. This notion of pseudodeterminism was formulated by Gat and Goldwasser [7], motivated by its applications in distributed computing and cryptography.

Let f be a multi-valued function (i.e. $f(x)$ is a non-empty set). We say that a probabilistic algorithm A computes f if $A(x)$ outputs a value in the set $f(x)$ with high probability. A probabilistic algorithm A for f is *pseudodeterministic* if for every x, there exists *a unique* $u \in f(x)$ such that $A(x)$ outputs u with high probability [7]. The problem of designing pseudodeterministic algorithms is interesting in scenarios where we know of a probabilistic polynomial time algorithm that computes f, but no deterministic algorithm for f is known.

Related Work

Since the work of Gat and Goldwasser, the notion of pseudodeterministic algorithms has received moderate attention [7, 8, 9, 10, 11, 16]. Now we know of pseudodeterministic algorithms for certain algebraic problems such as finding non-residues of $\mathbb{Z}_p$, and finding non-roots of multivariate polynomials [7]. The work of Oliveria and Santhanam gives a pseudodeterministic, sub-exponential time algorithm for generating primes (that works at infinitely many input lengths) [16]. Recently Goldwasser and Grossman obtained a pseudodeterministic NC algorithm that computes perfect matchings in bipartite graphs [9]. The work of Goldreich, Goldwasser and Ron investigates the power and limitations of pseudodeterministic algorithms in the context of general probabilistic algorithms and sub-linear time algorithms [8]. The notion of *pseudodeterminism* has been studied earlier in the literature in the context of approximation algorithms [6], where the authors relate the existence of such algorithms for approximation problems to communication complexity of certain key-agreement problems. They used the term *monic selection* to capture the notion of *pseudodeterminism*. To the best of our knowledge, pseudodeterminism in the context of computing approximations of function has not been studied further.

Our Contribution

In this paper, we investigate pseudodeterministic approximation algorithms. Given a function f whose range is the integers, we say that a probabilistic algorithm A is a *pseudodeterministic approximation algorithm* for f if, for every input x, there is a value v such that $A(x)$ outputs v with high probability and v is a "good approximation" of $f(x)$, (for a formal definition, please see the next section). We consider the following counting problem which we call NP-*membership counting problem*: Given a language L in NP, compute the number of strings in L at a given length. The well-known result due to Stockmeyer [19, 20, 3] shows that NP-membership counting problem can be approximated in $\mathrm{BPP}_{tt}^{\mathrm{NP}}$, where $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ denotes the class of problems solvable in probabilistic polynomial-time with nonadaptive queries to SAT[4]. However, Stockmeyer's algorithm is not pseudodeterministic: the algorithm can output different good approximations on different probabilistic paths. A natural question is: Is there a pseudodeterministic $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ approximation algorithm for NP-membership counting problem? We relate this question to establishing new circuit lower bounds.

Proving circuit lowerbounds is one of the most significant research directions in complexity theory. While it is widely believed that there are languages in NP that cannot be solved by subexponential-size Boolean circuits, we do not even know how to prove that NP does

[4] Even though $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ denotes a class of languages, here we slightly abuse the notation and view it as a function class.

not have linear-size circuits. This leads to investigating the question: "What is the smallest complexity class that provably cannot be solved by a linear-size circuit?" The first result in this direction is due to Kannan [13], who showed that there are languages in the second level of the Polynomial-time Hierarchy (Σ_2^{P}), that do not have linear-size circuits. This result has been improved to show that there are languages in $\mathrm{ZPP}^{\mathrm{NP}}$ that do not have linear-size circuits [4, 15]. Subsequently, Sengupta [5] showed that the complexity class $\mathrm{S}_2^{\mathrm{P}}$ does not have linear-size circuits. This together with Cai's result [5] that $\mathrm{S}_2^{\mathrm{P}} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$, improved the upper bound to $\mathrm{S}_2^{\mathrm{P}}$. It is also known that the complexity class PP has languages that do not have linear-size circuits [21]. Currently a significant open question is to show that the class MA does not have linear-size circuits. A step in this direction is a result due to Santhanam [18] who showed that MA//1 does not have linear-size circuits (MA//1 is a "promise version" of MA: see the next section for a formal definition). We note that in all these circuit lower bound results, "linear-size" can be replaced by "n^k-size circuits" (for a fixed $k \geq 0$).

We ask the following question: Can we show that $\mathrm{ZPP}_{tt}^{\mathrm{NP}}$ has languages that do not have linear-size circuits? Note that $\mathrm{MA} \subseteq \mathrm{ZPP}_{tt}^{\mathrm{NP}} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$ and hence it is a natural question. In this work, we show that this goal can be achieved if we can design a psedodeterminstic version of Stockmeyer's approximate counting algorithm for the NP-membership counting problem.

Theorem. *If there is a pseudodeterministic* $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ *approximation algorithm for the* NP*-membership counting problem, then for every* k *there is a language* L_k *in* $\mathrm{ZPP}_{tt}^{\mathrm{NP}}$ *that does not have* n^k*-size circuits.*

We note that there are oracles with respect to which $\mathrm{ZPP}_{tt}^{\mathrm{NP}}$ has linear-size circuits [1]. Hence a pseudodeterministic algorithm for the NP-membership counting problem will lead to non-relativizable circuit lower bounds.

Can we design a pseudodeterministic $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ approximation algorithm for the NP-membership counting problem? While we are unable to answer this question, we show a very general result: every randomized approximate counting algorithm can be transformed to an approximation algorithm that has a constant number of *influential random bits* in the sense defined recently by Grossman and Liu [12]. Grossman and Liu [12] generalized the notion of pseudodeterministic algorithms to *influential bit algorithms*. They defined this new notion in the context of logspace computation and applied it to certain search problems in randomized logspace. This notion can be adapted to polynomial-time bounded settings. Informally, a probabilistic algorithm A is a $k(n)$*-bit influential algorithm*, if A (on an input x of length n), uses $k(n) + r(n)$ random bits, and for most choices of the first $k(n)$-random bits, A behaves in a pseudodeterministic manner. That is, for most $p \in \Sigma^{k(n)}$, there exists a v such that $A(x, pr)$ outputs v with high probability (where r is randomly chosen from $\Sigma^{r(n)}$). For two different strings p_1 and p_2, the outputs of $A(x, p_1 r)$ and $A(x, p_2 r)$ could differ. However, if we fix a "good" $k(n)$-bit string p, then $A(x, pr_1)$ is the same as $A(x, pr_2)$ for most choices of r_1 and r_2. Note that an algorithm is pseudodeterministic if and only if it is a 0-influential bit algorithm. We show that any randomized relative-error approximate counting algorithm can be made to have a constant number of influential bits. This implies that

Theorem. *There is a* $O(1)$*-bit influential,* $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ *approximation algorithm for the* NP*-membership counting problem.*

2 Preliminaries

We assume familiarity with standard notation and definitions from complexity theory [2].

▶ **Definition 1.** Let f be a function whose range is the integers. We say that a probabilistic algorithm A is an (ϵ, δ)-approximation algorithm for f if for every x, the random variable $A(x)$ has the following property: $\Pr[(1-\epsilon)f(x) \leq A(x) \leq (1+\epsilon)f(x)] \geq (1-\delta)$. We say that A is an (ϵ, δ) *pseudodeterministic approximation algorithm* for f if for every x there exists an integer v such that

$$(1-\epsilon)f(x) \leq v \leq (1+\epsilon)f(x) \text{ and } \Pr[A(x) = v] \geq 1-\delta.$$

In general an approximation algorithm can output different good approximations on different random choices. For an approximation algorithm A to be pseudodeterministic, A has to usually output a *unique* approximation v which is independent of the random string, for every input.

▶ **Definition 2.** Let f be a function whose range is the integers. We say that f has an (ϵ, δ)-$\mathrm{BPP}^{\mathrm{NP}}_{tt}$ *pseudodeterministic approximation algorithm* if there exists an (ϵ, δ), polynomial-time, pseudodeterministic approximation algorithm for f, that makes nonadaptive queries to SAT.

Let $R \subseteq \Sigma^* \times \Sigma^*$ be a binary-relation that is decidable in NP. Let $f_R(x)$ be the number of strings y such that $\langle x, y\rangle \in R$. The class $\sharp\mathrm{NP}$ is defined as $\sharp\mathrm{NP} = \{f_R \mid R \text{ is a relation that is decidable in NP}\}$. The following well-known theorem due to Stockmeyer gives a $\mathrm{BPP}^{\mathrm{NP}}_{tt}$ approximation algorithm for problems in $\sharp\mathrm{NP}$.

▶ **Theorem 3** ([19]). *Every problem in* $\sharp\mathrm{NP}$ *has a* $(1/n, 1/2^n)$-$\mathrm{BPP}^{\mathrm{NP}}_{tt}$ *approximation algorithm.*

Stockmeyer's algorithm is not pseudodeterministic: it may produce different, correct approximations on different random choices.

Recently, Grossman and Liu defined the notion of *influential bits* as a generalization of pseudodeterminism [12]. They defined it in the logspace regime and applied it to search problems in RL. We adapt it to the context of approximation algorithms.

▶ **Definition 4.** Let f be a function whose range is the integers. For a function $k : \mathbb{N} \to \mathbb{N}$, we say that a randomized (ϵ, δ)-approximation algorithm A for f is $k(n)$*-bit influential* if for every x, the following holds: A takes random string $r = st$ where $|s| \leq k(n)$ and for more than $\frac{2}{3}$ of strings s, there exists an integer v_s such that $\Pr_t[A(x, st) = v_s] \geq 1-\delta$ and $(1-\epsilon)f(x) \leq v_s \leq (1+\epsilon)f(x)$.

Note that an algorithm is pseudodeterministic if and only if it is 0-bit influential. Next we define a variant of the complexity class AM.

▶ **Definition 5.** A language L is in $\mathrm{AM}//1$ if there exists a polynomial-time machine V, a polynomial p, and a sequence of bits $b_1, b_2, \cdots$ such that

$$x \in L \Rightarrow \Pr_{r \in \Sigma^{p(n)}}[\exists y \in \Sigma^{p(n)},\ V(xb_n, r, y) = 1] \geq 2/3,$$

$$x \notin L \Rightarrow \Pr_{r \in \Sigma^{p(n)}}[\exists y \in \Sigma^{p(n)},\ V(xb_n, r, y) = 1] \leq 1/3,$$

where n is the length of x.

Santhanam showed that for every k there is a language in AM//1 that does not have n^k-size circuits [18] (Santhanam showed the lower bound for MA//1, but we will only need the AM//1 bound for establishing our result).

▶ **Theorem 6** ([18]). *For every k, there is a language $L_k \in$ AM//1 so that L_k cannot be computed by n^k-size Boolean circuits.*

3 Pseudodeterminism for ♯NP and circuit lower bounds

In this section we show that the existence of a pseudodeterministic version of Stockmeyer's approximate counting algorithm for ♯NP implies new circuit lower bounds.

▶ **Theorem 7.** *If for every function f in ♯NP there exists an $(\frac{1}{10}, \frac{1}{10})$-$\mathrm{BPP}^{\mathrm{NP}}_{tt}$ pseudodeterministic approximation algorithm for f, then for every $k > 0$, there is a language L_k in $\mathrm{ZPP}^{\mathrm{NP}}_{tt}$ such that L_k cannot be computed by n^k-size Boolean circuits.*

Proof of this theorem follows from Lemma 8, and Theorem 9 stated below. Theorem 9 could be of independent interest.

▶ **Lemma 8.** *If for every function f in ♯NP there exists an $(\frac{1}{10}, \frac{1}{10})$-$\mathrm{BPP}^{\mathrm{NP}}_{tt}$ pseudodeterministic approximation algorithm for f, then for every $k > 0$, there is a language L_k in $\mathrm{BPP}^{\mathrm{NP}}_{tt}$ such that L_k cannot be computed by n^k-size Boolean circuits.*

▶ **Theorem 9.** *If* NP *has polynomial-size circuits, then $\mathrm{BPP}^{\mathrm{NP}}_{tt} = \mathrm{ZPP}^{\mathrm{NP}}_{tt}$.*

Assuming Lemma 8 and Theorem 9, Theorem 7 follows from the type of argument due to Kannan [13]. If NP does not have polynomial-size circuits, trivially $\mathrm{ZPP}^{\mathrm{NP}}_{tt}$ does not have n^k-size circuits. Assume that NP has polynomial-size circuits. By Lemma 8 we have $\mathrm{BPP}^{\mathrm{NP}}_{tt}$ does not have n^k-size circuits, and by Theorem 9 $\mathrm{BPP}^{\mathrm{NP}}_{tt}$ is the same as $\mathrm{ZPP}^{\mathrm{NP}}_{tt}$. Hence $\mathrm{ZPP}^{\mathrm{NP}}_{tt}$ does not have n^k-size circuits.

Proof of Lemma 8. Fix a $k > 0$ and let L be a language in AM//1 that does not have n^k-size circuits [18]. Thus there exists a sequence of bits $b_1, b_2, \cdots$, a polynomial p and a polynomial-time machine V such that

$$x \in L \quad \Rightarrow \quad \Pr_{r \in \Sigma^{p(n)}}[\exists y \in \Sigma^{p(n)},\ V(xb_n, r, y) = 1] \geq 2/3, \text{ and}$$
$$x \notin L \quad \Rightarrow \quad \Pr_{r \in \Sigma^{p(n)}}[\exists y \in \Sigma^{p(n)},\ V(xb_n, r, y) = 1] \leq 1/3$$

Consider the following NP relation:

$$R = \{\langle xb, r\rangle \mid r \in \Sigma^{p(|x|)}, b \in \{0,1\}, \exists y \in \Sigma^{p(n)} V(xb, r, y) = 1\}$$

The corresponding ♯NP problem is $f_R(xb) = |\{r \mid \langle xb, r\rangle \in R\}|$. By our hypothesis, there is a $(\frac{1}{10}, \frac{1}{10})$-$\mathrm{BPP}^{\mathrm{NP}}_{tt}$ pseudodeterministic approximation algorithm M for f_R. Let S_{xb} denote the set $\{r \mid \langle xb, r\rangle \in R\}$. Consider the following $\mathrm{BPP}^{\mathrm{NP}}_{tt}$ algorithm.

Input: x and a bit b.
Run M on xb. Let v be the output.
If $v \geq 2^{p(n)-1}$ then accept, else reject.

We will first show that on every input $\langle x, b\rangle$, the above algorithm either accepts with high probability or rejects with high probability. Let $\langle x, b\rangle$ be an input. Since M is a pseudodeterministic algorithm, there is an integer v such that M outputs v with probability at least $\frac{9}{10}$. If v is at least $2^{p(n)-1}$, then the above algorithm accepts, else it rejects. Thus on every input $\langle x, b\rangle$, the above algorithm either accepts with probability $\frac{9}{10}$ or rejects with probability at least $\frac{9}{10}$. Let L' be the language accepted by the above algorithm. Clearly $L' \in \mathrm{BPP}^{\mathrm{NP}}_{tt}$.

▶ **Claim 9.1.** *L' cannot be computed by n^k-size Boolean circuits.*

Assume that L' is accepted by a circuit family of size n^k. We will show that there is an n^k-size circuit family for L, which gives us a contradiction. Fix an input length n and let C_n be a circuit that accepts L' at length $n+1$. The definition of advice guarantees that for each input length, there is some advice string (in our case, a single bit) for which the circuit correctly decides the language. Let b_n be the *correct advice* bit for strings of length n for the language L. Consider the following circuit D_n: $D_n(x)$ is same as $C_n(x, b_n)$. I.e., with b_n hardwired in C_n. Thus D_n on input x (of length n) evaluates C_n on input $\langle x, b_n\rangle$. Clearly the size of D_n is n^k. We now show that the circuit family D_n accepts the language L.

Consider the case that x is in L. Note that $f_R(xb_n)$ is at least $\frac{2}{3}2^{p(n)}$. Thus on input $\langle x, b_n\rangle$ M outputs a number v with probability at least $2/3$ such that $v \geq \frac{9}{10} f_R(xb_n)$. Since $f_R(xb_n)$ is at least $\frac{2}{3}2^{p(n)}$, we have that v is bigger than $2^{p(n)-1}$. Thus M accepts $\langle x, b_n\rangle$ with probability at least $\frac{9}{10}$. Thus $\langle x, b_n\rangle \in L'$, and the circuit C_n accepts $\langle x, b_n\rangle$. Thus D_n accepts x. A similar argument shows that if x is not in L, then D_n rejects x. Since the size of D_n is n^k, we have that L is accepted by n^k-size circuits, which is a contradiction. ◀

Next, we complete the proof of Theorem 9.

Proof of Theorem 9. The argument goes in two steps: (1) under the assumption that NP has polynomial-size circuits, it can be shown that a Boolean function has $S(n)$-size SAT oracle circuits if and only if it has $S'(n)$-size Boolean circuits, where $S'(n)$ and $S(n)$ are polynomially related. (2) A ZPP machine can randomly pick a truth-table t of a Boolean function on $O(\log n)$-bits and verify using the NP oracle that t does not have $2^{\epsilon n}$-size circuits. Use this hard function to construct a pseudorandom generator. Use this PRG to derandomize a $\mathrm{BPP}^{\mathrm{NP}}_{tt}$ computation to $\mathrm{ZPP}^{\mathrm{NP}}_{tt}$. We provide more details.

Since NP has polynomial-size circuits, there is a constant $\ell > 0$ such that SAT has n^ℓ-size circuits. Suppose that a Boolean function g has $2^{\gamma n}$-size SAT-oracle circuit family $\{D_n\}$. Consider D_n. We can convert this into a circuit that does not make oracle calls to SAT. This circuit can generate queries (to SAT) of size at most $2^{\gamma n}$. Since SAT has n^ℓ-size circuits, each query can be answered by a circuit of size $2^{\ell\gamma n}$. Thus if we replace each oracle call in D_n by a circuit of size at most $2^{\ell\gamma n}$, we obtain Boolean circuit of size $2^{\delta n}$ for g where $\delta = 2l\gamma$. This circuit does not make any queries to SAT.

We will use the following derandomization result due to Klivans and Melkebeek [14]

▶ **Theorem 10.** *Let M be a $\mathrm{BPP}^{\mathrm{NP}}_{tt}$ machine that uses $t(n)$ random bits on inputs of length n. For every $\gamma > 0$, there exists b, c and a polynomial-time computable family of functions $\{F_n\}$ such that $F_n : \Sigma^{n^c} \times \Sigma^{b \log n} \to \Sigma^{t(n)}$, and if u is the truth-table of a $c \log n$-bit Boolean function with* SAT*-oracle circuit complexity $\Omega(2^{\gamma n})$, then for every x of length n*

$$\left| \Pr_{r \in \Sigma^{t(n)}}[M(x, r) = 1] - \Pr_{r \in \Sigma^{b \log n}}[M(x, F_n(u, r) = 1] \right| \leq 0.1$$

Let L be a language in $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ and M be a $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ machine that accepts L and runs in $t(n)$ time. Using Theorem 10, the following is a $\mathrm{ZPP}_{tt}^{\mathrm{NP}}$ machine that simulates M: Randomly pick u of size n^c – truth table of a function over $c \log n$ bits. By making a query to the NP oracle, check if u has circuit complexity at least $2^{\epsilon n}$. If not, then output "I do not know". Else, we can conclude that u has SAT-oracle circuit complexity at least $2^{\gamma n}$ for some $\gamma > 0$. Now simulate M using $F_n(u, r)$ as random bits for every $r \in \Sigma^{b \log n}$ and accept if the majority of simulations accept. Since with very high probability a randomly chosen u has circuit complexity at least $2^{\epsilon n}$, by Theorem 10 the simulation is correct. Thus L is in $\mathrm{ZPP}_{tt}^{\mathrm{NP}}$. ◀

We can extend Theorem 7 to a subclass of $\mathrm{ZPP}_{tt}^{\mathrm{NP}}$.

▶ **Theorem 11.** *If for every function f in $\sharp\mathrm{NP}$ there exists an $(\frac{1}{10}, \frac{1}{10})$-$\mathrm{BPP}_{tt}^{\mathrm{NP}}$ pseudodeterministic approximation algorithm, then for every $k > 0$, there is a language L_k in $\mathrm{ZPP}_{tt}^{\mathrm{NP}} \cap \mathrm{S}_2^{\mathrm{P}}$ such that L_k cannot be computed by n^k-size Boolean circuits.*

Proof. If NP does not have polynomial-size circuits, then the statement of the theorem trivially holds. If NP has polynomial-size circuits, then by Sengupta's result [5], the Polynomial-time Hierarchy collapses to $\mathrm{S}_2^{\mathrm{P}}$. Thus $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ is in $\mathrm{S}_2^{\mathrm{P}}$. Thus by Theorem 9, if NP has polynomial-size circuits, then $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ is a subset of $\mathrm{ZPP}_{tt}^{\mathrm{NP}} \cap \mathrm{S}_2^{\mathrm{P}}$. By Lemma 8, under the hypothesis, $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ does not have n^k-size circuits. The theorem follows. ◀

4 Constant-bit Influential algorithm for approximate counting

In this section we show that every probabilistic algorithm that computes an (ϵ, δ)-approximation to a function f can be transformed into a $O(1)$-bit influential algorithm.

▶ **Theorem 12.** *Let $\epsilon \in o(1)$, and $\delta \leq 1/3$. Let f be a function whose range is the integers that admits an (ϵ, δ)-approximation algorithm. Then there is a $O(1)$ influential-bit, $(O(\epsilon), \delta)$-approximation algorithm for f.*

Implicit in the work of Saks and Zhou [17] is a $O(\log n)$-bit influential, *absolute error* approximation algorithm for matrix powering. The main technical tool they use is that of *randomized rounding*. In the *relative error setting* that we are interested in, their rounding scheme does not work. We use an adaptive randomized rounding scheme that can handle relative errors and use this to design a constant-bit influential algorithm. Before we present our proof, we provide a high level overview of Saks and Zhou's proof adapted to our setting, and explain why it is not straight forward to apply in the relative error setting.

Let f be a function from Σ^* to integers such that n-bit integers are mapped to the range $[0, \cdots, 2^n]$. We say that an algorithm A computes (ϵ, δ), *absolute* approximation for f, if for every x, the value of $A(x)$ lies between $f(x) - \epsilon 2^n$ and $f(x) + \epsilon 2^n$ with probability at least $1 - \delta$. Let s be an integer such that $\epsilon 2^n$ is close to 2^{n-s}, thus the approximation error is close to 2^{n-s}. Saks and Zhou consider the following rounding operator $R_{b,s}(x)$, where b is a 4-bit integer. Subtract $2^{n-s} \times b$ from x and replace the last $n - s - 4$ bits of the resulting number with zeros. Saks and Zhou show that if z_1 and z_2 are any two good approximations of $f(x)$ (I.e, both z_1 and z_2 lie between $f(x) - \epsilon 2^n$ and $f(x) + \epsilon 2^n$), then for a random choice of b, $R_{b,s}(z_1) = R_{b,s}(z_2)$, and $R_{b,s}(z_1)$ is still a good approximation of $f(x)$ with high probability. From this it follows that any absolute error approximation algorithm can be made 4-bit influential. In the case of matrix powering, the rounding has to be applied for polynomially many entries, which will lead to an $O(\log n)$-bit influential algorithm for matrix powering.

For the above rounding scheme to work, it is critical that we know the value of s which depends on the value of approximation error which is at most $\epsilon 2^n$. However, in the relative error setting we do not a priori know the value of the approximation error, as it depends on $f(x)$. We could infer the of value of $\epsilon f(x)$ by looking at an output, and try to estimate s. However, since the approximation algorithm can produce multiple outputs, it is not possible to uniquely infer a value for s. We get around these problems by using additional (constant bits) randomness, and an adaptive rounding scheme. We now present a proof of Theorem 12.

Proof. Let f be a function from Σ^* to integers and let A be a (ϵ, δ)-approximation algorithm for f. For ease of presentation, we assume that $f(x) \leq 2^n$, where $n = |x|$. Consider the following algorithm. In this algorithm we set ℓ to 8 and p to $8 - \log(1/\epsilon)$. Note that p is negative. Given a number y in binary, and an integer z, we use $y\rfloor_z$ to denote the value obtained by replacing the last z bits of y with zeros. We use % for the modulo operator: $x\%y$ is the unique integer z in $\{0 \dots y-1\}$ satisfying $yr + z = x$ for some integer r. We say $x \equiv y\%z$ if $x\%y = z\%y$.

1. Input x of length n.
2. Choose m uniformly at random from $\{0 \dots 4\}$.
3. Choose r uniformly at random from $\{1 \dots 2^\ell\}$.
4. Run $A(x)$ and let y be the output.
5. Choose k_y so that $2^{k_y} \leq y < 2^{k_y+1}$.
6. Set $z_y = k_y + p + \{2, 1, 0, -1, -2\}$ so that $z_y \equiv m \ \% \ 5$.
7. Set $r_{z_y} = r \cdot 2^{(z_y - \ell)}$.
8. Output $(y - r_{z_y})\rfloor_{z_y}$.

We prove that this is a $O(1)$-bit influential algorithm for f, where the influential bits describe m and r. Fix an input x. The probability that A outputs a value in the range $[(1-\epsilon)f(x), (1+\epsilon)f(x)]$ is at least $1-\delta$. From now we assume that this event has happened. Let $\{y_1, y_2, \cdots, y_N\}$ be all possible outputs of $A(x)$ such that every $y_i \in (1 \pm \epsilon)f(x)$. Let y_{min} and y_{max} be the smallest and the largest of these values respectively. Since $\epsilon \leq 1/2$, there exists a $k \geq 0$ such that for every y_i, $2^k \leq y_i < 2^{k+2}$. Note that once the input x is fixed, k is also fixed.

We say $m \in \{0, \cdots, 4\}$ is *good* if all z_{y_i} $(1 \leq i \leq N)$ are the same (defined in Line 6). We will establish the following claim.

▶ **Claim 12.1.** *If we choose m randomly from $\{0, 1, 2, 3, 4\}$, m is good with probability $4/5$.*

Proof. Consider y_i and y_j, $i \neq j$. If both y_i and y_j lie between 2^k and 2^{k+1}, then clearly z_{y_i} equals z_{y_j}. Suppose that y_i lies between 2^k and 2^{k+1} and y_j lies between 2^{k+1} and 2^{k+2}. Thus $z_{y_i} \in \{k+p-2, k+p+1, k+p, k+p+1, k+p+2\}$ and $z_{y_j} \in \{k+p-1, k+p, k+p+1, k+p+2, k+p+3\}$. Note that both z_{y_i} and z_{y_j} are equal m modulo 5. If $(k+p-2) \equiv m\%5$, then the value of z_{y_i} equals $k+p-2$ and z_{y_j} equals $k+p+3$ and they differ. In all other cases, z_{y_i} equals z_{y_j}. The probability that randomly chosen $m \equiv (k+p-2)\%5$ is exactly $1/5$. Thus m is good with probability $4/5$. ◀

From now on we will assume that the event "m is good" has happened. Thus $z_{y_1} = z_{y_2} = \cdots = z_{y_N}$. For notational simplicity, we will denote this value by z. Note that r_z is formed by randomly picking $r \in \{1, \cdots 2^\ell\}$ and multiplying with $2^{(z-\ell)}$. We will prove the following claim that will be used later.

▶ **Claim 12.2.** *For every* $X \in [0 \dots 2^z]$,
1. $\frac{X}{2^z} - \frac{1}{2^\ell} \leq \Pr[r_z \leq X] \leq \frac{X}{2^z}$
2. $1 - \frac{X}{2^z} \leq \Pr[r_z > X] \leq 1 - (\frac{X}{2^z} - \frac{1}{2^\ell})$

Proof. Note that r_z is uniformly distributed in $\{\frac{2^z}{2^\ell}, 2 \cdot \frac{2^z}{2^\ell}, 3 \cdot \frac{2^z}{2^\ell}, \dots n^2 \cdot \frac{2^z}{2^\ell}\}$.
Thus, if $i\frac{2^z}{2^\ell} \leq X < (i+1)\frac{2^z}{2^\ell}$, there are i values of r_z that are at most X. We can write i as $\lfloor \frac{X \cdot n^\ell}{2^z} \rfloor \leq \frac{X \cdot 2^\ell}{2^z}$, giving us $\Pr[r_z \leq X] = \frac{i}{2^\ell} \leq \frac{X \cdot 2^\ell}{2^\ell \cdot 2^z} = \frac{X}{2^z}$ and $\Pr[r_z > X] = 1 - \frac{i}{2^\ell} \geq 1 - \frac{X}{2^z}$
Similarly, $i \geq \frac{X \cdot 2^\ell}{2^z} - 1$, so $\Pr[r_z \leq X] = \frac{i}{2^\ell} \geq \frac{X}{2^z} - \frac{1}{2^\ell}$ and $\Pr[r_z > X] = 1 - \frac{i}{2^\ell} \leq 1 - (\frac{X}{2^z} - \frac{1}{2^\ell})$ ◀

▶ **Lemma 13.** *For every good* m*, if we randomly choose* $r \in \{1, \cdots, 2^\ell\}$*, then for at least* $223/256$ *of the possible* r*, the following holds: For every* $i \neq j$ *(*$1 \leq i, j \leq r$*)* $(y_i - r_z)\rfloor_z = (y_j - r_z)\rfloor_z$.

Proof. The $\rfloor_z$ operation can be viewed as dividing the interval $[0, 2^n]$ into subintervals $[0, 2^z - 1], [2^z, 2 \cdot 2^z - 1], [2 \cdot 2^z, 3 \cdot 2^z - 1] \dots [2^n - 2^z, 2^n]$, and mapping the contents of each interval to its left endpoint. Note that $y_{max} - y_{min}$ is at most $2f(x)\epsilon$. Since $f(x) \leq 2^{k+2}$ and $2^z \geq 2^{k+p-2}$, $y_{max} - y_{min}$ is at most $2^{k+3}\epsilon$, which is at most $2^z/8$. Since the size of each interval is 2^z and the difference between y_{max} and y_{min} is less than the size of the interval, either both y_{min} and y_{max} lie in the same interval or lie in two contiguous intervals. We consider both cases.

Case 1. Both y_{max} and y_{min} are in the same interval, say $[i \cdot 2^z, (i+1) \cdot 2^z]$ for some i. If subtracting r_z causes them to be in different intervals, r_z must be large enough to move y_{min} to a new interval, but not so large that y_{max} moves too. That is, $r_z > (y_{min} - i \cdot 2^z)$ and $r_z \leq (y_{max} - i \cdot 2^z)$. They are in the same interval when $r_z \leq (y_{min} - i \cdot 2^z)$ or $r_z > (y_{max} - i \cdot 2^z)$.
So, the probability that y_{min} and y_{max} are in the same interval is

$$\begin{aligned} &\Pr[r_z > (y_{max} - i \cdot 2^z)] + \Pr[r_z \leq (y_{min} - i \cdot 2^z)] \\ &\quad\geq 1 - \frac{y_{max} - i \cdot 2^z}{2^z} + \frac{y_{min} - i \cdot 2^z}{2^z} - \frac{1}{2^\ell} \\ &\quad= 1 - \frac{y_{max} - y_{min}}{2^z} - \frac{1}{2^l} \\ &\quad\geq 223/256 \end{aligned}$$

The first inequality follows from Claim 12.2. The second follows from the conclusion that $y_{max} - y_{min}$ is at most $2^z/8$. The last inequality follows because $\ell = 8$.

Case 2. Now consider the other case, where y_{min} is in the interval $[i \cdot 2^z, (i+1) \cdot 2^z]$ and y_{max} is in $[(i+1) \cdot 2^z, (i+2) \cdot 2^z]$. If subtracting r_z causes y_{max} and y_{min} to be in different intervals, either r_z is not large enough to move y_{max} into the same interval as y_{min}, or r_z is so large that it moves y_{min} into a new interval as well. In this case, the condition is $r_z > (y_{min} - i \cdot 2^z)$ or $r_z \leq (y_{max} - (i+1) \cdot 2^z)$. Then, the probability that y_{min} and y_{max} end up in different intervals is

$$\begin{aligned} &\Pr[r_z > (y_{min} - i \cdot 2^z)] + \Pr[r_z \leq (y_{max} - (i+1) \cdot 2^z)] \\ &\quad\leq 1 - (\frac{y_{min} - i \cdot 2^z}{2^z} - \frac{1}{2^\ell}) + \frac{y_{max} - (i+1) \cdot 2^z}{2^z} \\ &\quad= 1 - \frac{y_{min}}{2^z} + i + \frac{1}{2^\ell} + \frac{y_{max}}{2^z} - i - 1 \\ &\quad= \frac{y_{max} - y_{min}}{2^z} + \frac{1}{2^\ell} \\ &\quad\leq 33/256 \end{aligned}$$

The first inequality follows from Claim 12.2. The second follows from the conclusion that $y_{max} - y_{min}$ is at most $2^z/8$. ◀

Our next claim shows that the operation $\rfloor z$ preserves the approximation (up to a constant factor).

▶ **Claim 13.1.** *If $y \in f(x)(1 \pm \epsilon)$, then $(y - r_z)\rfloor_z \in f(x)(1 \pm O(\epsilon))$.*

Proof. Subtracting r_z and taking the $\rfloor_z$ only decrease, so this is at most y, which is at most $f(x)(1+\epsilon)$. For a given y, the minimum value of $(y - r_z)\rfloor_z$ occurs when r is as large as possible and $\rfloor_z$ changes the last z bits from 1 to 0. r_z is at most 2^z. $\rfloor_z$ can decrease the input by at most 2^z. Thus, $(y - r_z)\rfloor_z$ is at least $y - 2^z - 2^z$. Consider the following inequalities.

$$\begin{aligned} 2^{z+1} &\leq 2^{k+2+p+2} \\ &= 2^{k+13-\log 1/\epsilon} \\ &\leq f(x)(2^{13}\epsilon)(\text{ as } f(x) \geq 2^k) \end{aligned}$$

Since $(y-r_z)\rfloor_z \geq y-2^{z+1}$, and $y \geq f(x)(1-\epsilon)$, we obtain that $(y-r_z)\rfloor_z \geq f(x)(1-2^{14}\epsilon)$. ◀

Now we can finish the proof of Theorem 12. The influential bits of the algorithm are m and r. Note that the number of influential bits is constant. Let us call the set of r for which the consequence of Lemma 13 holds as *good*. Note that for every fixing of good m and r, the output of the algorithm is unique and is a good approximation, with probability at least $1 - \delta$, by Lemma 13 and Claim 13.1. Finally the fraction of bad m and r is at most $1/5 + 33/256$ which is at most $1/3$. Thus the algorithm is a constant-bit influential algorithm.

The statement of Theorem 12 assumed that $\epsilon \in o(1)$. However, the proof goes through for any sufficiently small ϵ. We have the following as a corollary of Theorem 12. ◀

▶ **Theorem 14.** *For every problem in $\sharp$NP, there is a $O(1)$-bit influential, $(1/n, 1/2^n)$-approximation algorithm.*

References

1. S. Aaronson. Oracles are subtle but not malicious. In *IEEE Conference on Computational Complexity*, pages 340–354, 2006.
2. S. Arora and B. Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.
3. M. Bellare, O. Goldreich, and E. Petrank. Uniform generation of np-witnesses using an np-oracle. *Inf. Comput.*, 163(2):510–526, 2000.
4. N. H. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *J. Comput. Syst. Sci.*, 52(3):421–433, 1996.
5. J-Y. Cai. $\mathrm{S}^{\mathrm{p}}_2$ subseteq $\mathrm{zpp}^{\mathrm{np}}$. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 620–629, 2001.
6. J. Y. Cai, R. Lipton, L. Longpré, M. Ogihara, K. Regan, and D. Sivakumar. Communication complexity of key agreement on small ranges. In *STACS*, pages 38–49, 1995.
7. E. Gat and S. Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:136, 2011.
8. O. Goldreich, S. Goldwasser, and D. Ron. On the possibilities and limitations of pseudodeterministic algorithms. In *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*, pages 127–138, 2013.
9. S. Goldwasser and O. Grossman. Bipartite perfect matching in pseudo-deterministic NC. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 87:1–87:13, 2017.

10 S. Goldwasser, O. Grossman, and D. Holden. Pseudo-deterministic proofs. *CoRR*, abs/1706.04641, 2017.

11 O. Grossman. Finding primitive roots pseudo-deterministically. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:207, 2015.

12 O. Grossman and Y. Liu. Reproducibility and pseudo-determinism in log-space. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:48, 2018.

13 R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55(1-3):40–56, 1982.

14 A. Klivans and D. Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM J. Comput.*, 31(5):1501–1526, 2002.

15 J. Köbler and O. Watanabe. New collapse consequences of NP having small circuits. *SIAM J. Comput.*, 28(1):311–324, 1998.

16 I. Oliveira and R. Santhanam. Pseudodeterministic constructions in subexponential time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 665–677, 2017.

17 M. Saks and S. Zhou. BP $_{\mathrm{h}}$space(s) subseteq dspace(s$^{3/2}$). *J. Comput. Syst. Sci.*, 58(2):376–403, 1999.

18 R. Santhanam. Circuit lower bounds for merlin–arthur classes. *SIAM J. Comput.*, 39(3):1038–1061, 2009.

19 L. Stockmeyer. The complexity of approximate counting (preliminary version). In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 118–126, 1983.

20 L. Stockmeyer. On approximation algorithms for #p. *SIAM J. Comput.*, 14(4):849–861, 1985.

21 N. V. Vinodchandran. A note on the circuit complexity of PP. *Theor. Comput. Sci.*, 347(1-2):415–418, 2005.

Testing Simon's congruence

Lukas Fleischer[1]
FMI, University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
fleischer@fmi.uni-stuttgart.de

Manfred Kufleitner
Department of Computer Science, Loughborough University
Epinal Way, Loughborough LE11 3TU, United Kingdom
m.kufleitner@lboro.ac.uk

Abstract

Piecewise testable languages are a subclass of the regular languages. There are many equivalent ways of defining them; Simon's congruence $\sim_k$ is one of the most classical approaches. Two words are $\sim_k$-equivalent if they have the same set of (scattered) subwords of length at most k. A language L is piecewise testable if there exists some k such that L is a union of $\sim_k$-classes.

For each equivalence class of $\sim_k$, one can define a canonical representative in shortlex normal form, that is, the minimal word with respect to the lexicographic order among the shortest words in $\sim_k$. We present an algorithm for computing the canonical representative of the $\sim_k$-class of a given word $w \in A^*$ of length n. The running time of our algorithm is in $\mathcal{O}(|A|\, n)$ even if $k \leq n$ is part of the input. This is surprising since the number of possible subwords grows exponentially in k. The case $k > n$ is not interesting since then, the equivalence class of w is a singleton. If the alphabet is fixed, the running time of our algorithm is linear in the size of the input word. Moreover, for fixed alphabet, we show that the computation of shortlex normal forms for $\sim_k$ is possible in deterministic logarithmic space.

One of the consequences of our algorithm is that one can check with the same complexity whether two words are $\sim_k$-equivalent (with k being part of the input).

2012 ACM Subject Classification Mathematics of computing → Combinatorics on words, Theory of computation → Formal languages and automata theory

Keywords and phrases regular language, scattered subword, piecewise testability, string algorithm

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.62

Acknowledgements We would like to thank the anonymous reviewers for their helpful comments. Their suggestions significantly improved the presentation of this paper.

1 Introduction

We write $u \preccurlyeq v$ if the word u is a (scattered) subword of v, that is, if there exist factorizations $u = u_1 \cdots u_n$ and $v = v_0 u_1 v_1 \cdots u_n v_n$. In the literature, subwords are sometimes called *piecewise* subwords to distinguish them from factors. Higman showed that, over finite alphabets, the relation $\preccurlyeq$ is a well-quasi-ordering [3]. This means that every language contains only finitely many minimal words with respect to the subword ordering. This led to the consideration of piecewise testable languages. A language L is *piecewise testable* if there exists a finite set of words T such that $v \in L$ only depends on $\{u \in T \mid u \preccurlyeq v\}$; in

[1] Supported by the German Research Foundation (DFG) under grant DI 435/5–2.

other words, the occurrence and non-occurrence of subwords in T determines membership in L. Equivalently, a language L is piecewise testable if it is a finite Boolean combination of languages of the form $A^*a_1A^* \cdots a_nA^*$ with $a_i \in A$ (using the above notation, the sequences $a_1 \cdots a_n$ in this combination give the words in T). The piecewise testable languages are a subclass of the regular languages and they play a prominent role in many different areas. For instance, they correspond to the languages definable in alternation-free first-order logic [20] which plays an important role in database queries. They also occur in learning theory [10, 15] and computational linguistics [2, 14].

In the early 1970s, Simon proved his famous theorem on piecewise testable languages: A language is piecewise testable if and only if its syntactic monoid is finite and $\mathcal{J}$-trivial [18]. An immediate consequence of Simon's Theorem is that it is decidable whether or not a given regular language L is piecewise testable. Already in his PhD thesis [17], Simon considered the complexity of this problem when L is given as a deterministic finite automaton (DFA). His algorithm can be implemented to have a running time of $\mathcal{O}(2^{|A|}n^2)$ for an n-state DFA over the alphabet A. This result was successively improved over the years [1, 9, 19, 21] with the latest algorithm having a running time of $\mathcal{O}(|A|^2\, n)$; see [8]. If the input is a DFA, then the problem is NL-complete [1]; and if the input is a nondeterministic finite automaton, the problem is PSPACE-complete [4]. Restricting the length of the relevant subwords T to some constant k leads to the notion of k-piecewise testable languages. At first sight, it is surprising that, for every fixed $k \geq 4$, deciding whether a given DFA accepts a k-piecewise testable language is coNP-complete [8]; see also [11].

One of the main tools in the original proof of Simon's Theorem is the congruence $\sim_k$ for $k \in \mathbb{N}$. By definition, two words u and v satisfy $u \sim_k v$ if u and v have the same subwords of length at most k. Naturally, the relation $\sim_k$ is nowadays known as *Simon's congruence*. It is easy to see that a language L is piecewise testable if and only if there exists k such that L is a union of $\sim_k$-classes. Understanding the combinatorial properties of $\sim_k$ is one of the main tools in the study of piecewise testable languages. For example, in the proof of his theorem, Simon already used that $(uv)^k \sim_k (uv)^k u$ for all words u, v. Upper and lower bounds on the index of $\sim_k$ were given by Kátai-Urbán et al. [7] and Karandikar et al. [6].

There are two natural approaches for testing whether or not $u \sim_k v$ holds. The first approach constructs a DFA $\mathcal{A}_{k,u}$ for the language $\{w \preccurlyeq u \mid k \geq |w|\}$ of the subwords of u of length at most k and a similar DFA $\mathcal{A}_{k,v}$ for v. Then $u \sim_k v$ if and only if $\mathcal{A}_{k,u}$ and $\mathcal{A}_{k,v}$ accept the same language. This can be tested with Hopcroft's algorithm in time almost linear in the size of the automata [5]. Here, *almost linear* in n means $\mathcal{O}(n \cdot a(n))$ where $a(n)$ is the inverse Ackermann function. It is possible to construct the automata such that $\mathcal{A}_{k,u}$ has at most $k\,|u| + 2$ states, see the remark at the end of Section 2 below. Hence, the resulting test is almost linear in $|A|\ k\ |uv|$ if the alphabet is A.

The second approach to testing $u \sim_k v$ is the computation of normal forms. A normal form is a unique representative of a $\sim_k$-class. In particular, we have $u \sim_k v$ if and only if u and v have the same normal form. By computing the normal forms for both words and then checking whether they are identical, the complexity of this test of $u \sim_k v$ is the same as the computation of the normal forms. We should mention that the computation of normal forms is also interesting in its own right since it can provide some insight into the combinatorial properties of $\sim_k$. Normal forms for $k = 2$ and $k = 3$ were considered by Kátai-Urbán et al. [7] and normal forms for $k = 4$ were given by Pach [12]. An algorithm for computing normal forms for arbitrary k was found only recently by Pach [13]. Its running time is $\mathcal{O}(|A|^k\,(n + |A|))$ for inputs of length n over the alphabet A, that is, polynomial for fixed k and exponential otherwise.

We significantly improve this result by providing an algorithm with a running time in $\mathcal{O}(|A|\, n)$ even if k is part of the input. For every fixed alphabet, the running time is linear, which is optimal. Moreover, the algorithm can easily be adapted to run in deterministic logarithmic space, thereby addressing an open problem from [7]. As a consequence we can check with the same running time (or the same complexity) whether two given words are $\sim_k$-equivalent even if k is part of the input, thereby considerably improving on the above automaton approach.

Our algorithm actually does not compute just some normal form but the *shortlex normal form* of the input word u, *i.e.*, a shortest, and among all shortest the lexicographically smallest, word v such that $u \sim_k v$. Our main tools are so-called *rankers* [16, 22]. For each position i in the input word, the algorithm computes the lengths of the shortest X-rankers and Y-rankers reaching i. One can then derive the shortlex normal form by deleting and sorting certain letters based on these attributes. A more detailed outline of the paper is given in Section 3.

2 Preliminaries

Let A be a finite alphabet. The elements in A are called *letters* and a sequence of letters $u = a_1 \cdots a_\ell$ is a *word* of *length* $|u| = \ell$. The set of all words over the alphabet A is A^*. Throughout this paper, a, b and c are used to denote letters. For a word $a_1 \cdots a_\ell$, the numbers $\{1, \ldots, \ell\}$ are called *positions* of the word, and i is a *c-position* if $a_i = c$. The letter a_i is the *label* of position i. Two positions i and j with $i < j$ are *consecutive* c-positions if $a_i = a_j = c$ and $a_\ell \neq c$ for all $\ell \in \{i+1, \ldots, j-1\}$. A word $a_1 \cdots a_\ell$ is a *subword* of a word $v \in A^*$ if v can be written as $v = v_0 a_1 \cdots v_{\ell-1} a_\ell v_\ell$ for words $v_i \in A^*$. We write $u \preccurlyeq v$ if u is a subword of v. A *congruence* on A^* is an equivalence relation $\sim$ such that $u \sim v$ implies $puq \sim pvq$ for all $u, v, p, q \in A^*$. For $k \in \mathbb{N}$, *Simon's congruence* $\sim_k$ on A^* is defined by $u \sim_k v$ if and only if u and v contain the same subwords of length at most k.

We assume that the letters of the alphabet are totally ordered. A word u is *lexicographically smaller than* v if, for some common $p \in A^*$, there exists a prefix pa of u and a prefix pb of v such that $a < b$. (We apply the lexicographic order only for words of the same length; in particular, we do not care about the case when u is a proper prefix of v.) Given a congruence $\sim$ on A^*, we define the *shortlex normal form* of a word u to be the shortest word v such that $u \sim v$ and such that no other word $w \in A^*$ with $w \sim v$ and $|w| = |v|$ is lexicographically smaller than v. In other words, we first pick the shortest words in the $\sim$-class of u and among those, we choose the lexicographically smallest one.

Our main tools are so-called *rankers* [16, 22]. An X-*ranker* is a nonempty word over the alphabet $\{\mathsf{X}_a \mid a \in A\}$ and a Y-*ranker* is a nonempty word over $\{\mathsf{Y}_a \mid a \in A\}$. The length of a ranker is its length as a word. The modality X_a means ne$\mathsf{X}$t-a and is interpreted as an instruction of the form "go to the next a-position"; similarly, Y_a is a shorthand for Yesterday-a and means "go to the previous a-position". More formally, we let $\mathsf{X}_a(u) = i$ if i is the smallest a-position of u, and we let $r\mathsf{X}_a(u) = i$ for a ranker r if i is the smallest a-position greater than $r(u)$. Symmetrically, we let $\mathsf{Y}_a(u) = i$ if i is the greatest a-position of u and we let $r\mathsf{Y}_a(u) = i$ if i is the greatest a-position smaller than $r(u)$. In particular, rankers are processed from left to right. Note that the position $r(u)$ for a ranker r and a word u can be undefined. If $r(u)$ is defined, then we say that r *reaches* the position $r(u)$. Similarly, r *visits* a position i if $s(u) = i$ for some prefix s of r. A word $b_1 \cdots b_\ell$ *defines* an X-ranker $\mathsf{X}_{b_1} \cdots \mathsf{X}_{b_\ell}$ and a Y-ranker $\mathsf{Y}_{b_\ell} \cdots \mathsf{Y}_{b_1}$. We have $u \preccurlyeq v$ if and only if $r(v)$ is defined for the

X-ranker (resp. Y-ranker) r defined by u. Similarly, if r is the X-ranker defined by u and s is the Y-ranker defined by v, then $uv \preccurlyeq w$ if and only if $r(w) < s(w)$. The correspondence between X-rankers and subwords leads to the following automaton construction.

▶ Remark. Let u be a word of length n. We construct a DFA $\mathcal{A}_{k,u}$ for the language $\{w \preccurlyeq u \mid |w| \leq k\}$. The set of states is $\{(0,0)\} \cup \{1,\dots,k\} \times \{1,\dots,n\}$ plus some sink state which collects all missing transitions. The initial state is $(0,0)$ and all states except for the sink state are final. We have a transition $(\ell,i) \xrightarrow{a} (\ell+1,j)$ if $\ell < k$ and j is the smallest a-position greater than i. The idea is that the first component counts the number of instructions and the second component gives the current position.

3 Attributes and outline of the paper

To every position $i \in \{1,\dots,n\}$ of a word $a_1 \cdots a_n \in A^n$, we assign an *attribute* (x_i, y_i) where x_i is the length of a shortest X-ranker reaching i and y_i is the length of a shortest Y-ranker reaching i. We call x_i the *x-coordinate* and y_i the *y-coordinate* of position i.

▶ **Example 1.** We will use the word $u = bacbaabada$ as a running example throughout this paper. The attributes of the positions in u are as follows:

12	12	11	22	23	32	31	42	11	21
b	a	c	b	a	a	b	a	d	a

The letter a at position 5 can be reached by the Y-ranker $\mathsf{Y}_b\mathsf{Y}_a\mathsf{Y}_a$ and the a at position 6 can be reached by the X-ranker $\mathsf{X}_c\mathsf{X}_a\mathsf{X}_a$. Both rankers visit both positions 5 and 6. No X-ranker visiting position 6 can avoid position 5 and no Y-ranker visiting position 5 can avoid position 6. Deleting either position 5 or 6 reduces the attributes of the other position to $(2,2)$.

We propose a two-phase algorithm for computing the shortlex normal form of a word u within its $\sim_k$-class. The first phase deletes letters and results in a word of minimal length within the $\sim_k$-class of u. The second phase sorts blocks of letters to get the minimal word with respect to the lexicographic ordering. Both phases depend on the attributes. The computation of the attributes and the first phase are combined as follows.

Phase 1a: Compute all x-coordinates from left to right.
Phase 1b: Compute all y-coordinates from right to left while dynamically deleting a position whenever the sum of its coordinates would be bigger than $k+1$.
Phase 2: Swap consecutive letters b and a (with $b > a$) whenever they have the same attributes and the sum of the x- and the y-coordinate equals $k+1$.

As we will show, a crucial property of Phase 1b is that the dynamic process does not mess up the x-coordinates of the remaining positions that were previously computed in Phase 1a.

The **outline** of the paper is as follows. In Section 4, we prove that successively deleting all letters where the sum of the attributes is bigger than $k+1$ eventually yields a length-minimal word within the $\sim_k$-class of the input. This statement has two parts. The easier part is to show that we can delete such a position without changing the $\sim_k$-class. The more difficult part is to show that if no such deletions are possible, the word is length-minimal within its $\sim_k$-class. In particular, no other types of deletions are required. Also note that deleting letters can change the attributes of the remaining letters.

Section 5 has two components. First, we show that commuting consecutive letters does not change the $\sim_k$-class if (a) the two letters have the same attribute and (b) the sum of the x- and the y-coordinate equals $k+1$. Moreover, such a commutation does not change

any attributes. Then, we prove that no other types of commutation are possible within the $\sim_k$-class. This is quite technical to formalize since, a priori, we could temporarily leave the $\sim_k$-class only to re-enter it again with an even smaller word.

Finally, in Section 6, we present an easy and efficient algorithm for computing shortlex normal forms for $\sim_k$. First, we show how to efficiently compute the attributes. Then we combine this computation with a single-pass deletion procedure; in particular, we do not have to successively re-compute the attributes after every single deletion. Finally, an easy observation shows that we only have to sort disjoint factors where the length of each factor is bounded by the size of the alphabet. Altogether, this yields an $\mathcal{O}(|A|\, n)$ algorithm for computing the shortlex normal form of an input word of length n over the alphabet A. Surprisingly, this bound also holds if k is part of the input.

4 Length reduction

In order to reduce words to shortlex normal form, we want to identify positions in the word which can be deleted without changing its $\sim_k$-class. The following proposition gives a sufficient condition for such deletions.

▶ **Proposition 2.** *Consider a word uav with $a \in A$ and $|ua| = i$. If the attribute (x_i, y_i) at position i satisfies $x_i + y_i > k + 1$, then $uav \sim_k uv$.*

Proof. Let $w \preccurlyeq uav$ with $|w| \leq k$. Assume that $w \not\preccurlyeq uv$. Let $w = paq$ such that $p \preccurlyeq u$ and $q \preccurlyeq v$. Note that $pa \not\preccurlyeq u$ and $aq \not\preccurlyeq v$. If $|p| \geq x_i - 1$ and $|q| \geq y_i - 1$, then

$$k \geq |w| = |p| + 1 + |q| \geq (x_i - 1) + 1 + (y_i - 1) = x_i + y_i - 1 > k,$$

a contradiction. Therefore, we have either $|p| < x_i - 1$ or $|q| < y_i - 1$. By left-right symmetry, it suffices to consider $|p| < x_i - 1$. The word pa defines an X-ranker of length less than x_i which reaches position i. This is not possible by definition of x_i. Hence, $w \preccurlyeq uv$. Conversely, if $w \preccurlyeq uv$ for a word w, then obviously we have $w \preccurlyeq uav$. This shows $uav \sim_k uv$. ◀

▶ **Example 3.** Let $u = bacbaabada$ as in Example 1 and let $k = 3$. Note that the attributes (x_i, y_i) at positions $i \in \{5, 6\}$ satisfy the condition $x_i + y_i > k + 1$. By Proposition 2, deleting any of these positions yields a $\sim_k$-equivalent word. However, deleting both positions yields the word $bacbbada \not\sim_k u$ since $cab \preccurlyeq u$ and $cab \not\preccurlyeq bacbbada$.

Consider a position i with attribute (x_i, y_i) in a word u. Let

$$R_i^u = \{r \mid r \text{ is an } \mathsf{X}\text{-ranker with } r(u) = i \text{ and } |r| = x_i\}.$$

We have $R_i^u \neq \emptyset$ by definition of x_i. We define a *canonical* X*-ranker* $r_i^u \in R_i^u$ by minimizing the reached positions, and the minimization procedure goes from right to left: Let $S_{x_i} = R_i^u$ and, inductively, we define S_j as a nonempty subset of S_{j+1} as follows. Let p_j be the minimal position in u visited by the prefixes s of length j of the rankers in S_{j+1}; then S_j contains all rankers in S_{j+1} such that their prefixes of length j visit the position p_j. Since the minimal positions (and their labels) in this process are unique, we end up with $|S_1| = 1$. Now, the ranker r_i^u is given by $S_1 = \{r_i^u\}$. By abuse of notation, we will continue to use the symbol r for arbitrary rankers while r_i^u denotes canonical rankers. The following example shows that minimizing from right to left (and not the other way round) is crucial.

▶ **Example 4.** Let $u = abcabcdaefccabc$. The attributes of the letters are as follows:

13	13	13	22	22	22	11	22	11	11	23	32	21	21	31
a	b	c	a	b	c	d	a	e	f	c	c	a	b	c
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The last c is at position 15 and its attribute is $(3, 1)$. It is easy to verify that $\mathsf{X}_e\mathsf{X}_a\mathsf{X}_c$ is an X-ranker of length 3 visiting position 15 and that there is no X-ranker of length 2 reaching this position. The unique Y-ranker of length 1 reaching position 15 is Y_c. We have

$$R^u_{15} = \{\mathsf{X}_d\mathsf{X}_b\mathsf{X}_c, \mathsf{X}_e\mathsf{X}_a\mathsf{X}_c, \mathsf{X}_e\mathsf{X}_b\mathsf{X}_c, \mathsf{X}_f\mathsf{X}_a\mathsf{X}_c, \mathsf{X}_f\mathsf{X}_b\mathsf{X}_c\}\,.$$

Using the above notation, it is easy to see that $S_3 = R^u_{15}$, $S_2 = \{\mathsf{X}_e\mathsf{X}_a\mathsf{X}_c, \mathsf{X}_f\mathsf{X}_a\mathsf{X}_c\}$, and $S_1 = \{\mathsf{X}_e\mathsf{X}_a\mathsf{X}_c\}$. All prefixes of length 2 of rankers in S_2 reach position $p_2 = 13$; the prefix of length 1 of the ranker in S_1 reaches position $p_1 = 9$. The ranker visiting positions 9, 13 and 15 (and no other positions) is $r^u_{15} = \mathsf{X}_e\mathsf{X}_a\mathsf{X}_c$, the unique ranker in S_1.

Also note that the minimal positions m_j visited by prefixes of length j of the rankers in R^u_{15} are $m_1 = 7$, $m_2 = 13$, and $m_3 = 15$; but there is no single ranker of length 3 visiting positions 7, 13, and 15.

While r^u_i is defined in some right-to-left manner, it still has an important left-to-right property when positions of the same label are considered.

▶ **Lemma 5.** *Let $i < j$ be two consecutive c-positions in a word u with attributes (x_i, y_i) and (x_j, y_j), respectively. If $x_j > x_i$, then $r^u_j = r^u_i\mathsf{X}_c$.*

Proof. Since no position ℓ with $i < \ell < j$ is labelled by c, we have $r^u_i\mathsf{X}_c(u) = j$. In particular, $x_j \leq x_i + 1$ and, hence, $x_j = x_i + 1$ by assumption. This shows $R^u_i\mathsf{X}_c \subseteq R^u_j$. Let $r^u_j = r\mathsf{X}_c$. We have $r(u) \geq r^u_i(u)$, since otherwise $r\mathsf{X}_c(u) \leq i < j$, a contradiction. The minimization in the definition of r^u_j now yields $r(u) = r^u_i(u)$. The remaining minimization steps in the definition of r^u_i and r^u_j consider the same rankers and thus the same positions. Hence, $r = r^u_i$. ◀

We now want to prove that the condition introduced in Proposition 2 always results in a shortest word within the corresponding $\sim_k$-class. To this end, we first need the following technical lemma and then prove the main theorem of this section.

▶ **Lemma 6.** *Let $u = a_1 \cdots a_n$ be a word and let $i < j$ be consecutive c-positions. Moreover, let the parameters (x_i, y_i) and (x_j, y_j) satisfy $x_i + y_i \leq k + 1$ and $x_j \leq k$, respectively. For every word v with $u \sim_k v$, we have $r^u_i(v) < r^u_j(v)$.*

Proof. We have $x_i \leq k$ and $x_j \leq k$. Therefore, both $r^u_i(v)$ and $r^u_j(v)$ are defined because this only depends on subwords of length at most k which are identical for u and v. Since $j = r^u_i\mathsf{X}_c(u)$, we have $x_j \leq x_i + 1$. If $x_j = x_i + 1$, then $r^u_j = r^u_i\mathsf{X}_c$ by Lemma 5 and hence $r^u_i(v) < r^u_j(v)$. Therefore, we can assume $x_j \leq x_i$. Suppose that $r^u_i(v) \geq r^u_j(v)$. Let $q\mathsf{Y}_c$ be a Y-ranker with $q\mathsf{Y}_c(u) = i$ and $|q\mathsf{Y}_c| = y_i$. Let w_i be the word which defines r^u_i, let w_j be the word which defines r^u_j, and let z be the word which defines q. We have:

$$\begin{aligned}
& w_i z \preccurlyeq u && \text{since } r^u_i(u) = i < q(u)\\
\Rightarrow\ & w_i z \preccurlyeq v && \text{since } |w_i z| = x_i + y_i - 1 \leq k \text{ and } u \sim_k v\\
\Rightarrow\ & w_j z \preccurlyeq v && \text{since } r^u_i(v) \geq r^u_j(v)\\
\Rightarrow\ & w_j z \preccurlyeq u && \text{since } |w_j z| = x_j + y_i - 1 \leq x_i + y_i - 1 \leq k\\
\Rightarrow\ & q(u) > j\\
\Rightarrow\ & q\mathsf{Y}_c(u) \geq j > i.
\end{aligned}$$

This contradicts $q\mathsf{Y}_c(u) = i$. Therefore, we have $r^u_i(v) < r^u_j(v)$. ◀

▶ **Theorem 7.** *If u is a word such that the attribute (x_i, y_i) of every position i satisfies $x_i + y_i \leq k + 1$, then u has minimal length within its $\sim_k$-class.*

Proof. Let v be any word satisfying $u \sim_k v$. Let ρ map the position j of u to the position $r_j^u(v)$ of v. Consider some letter c occurring in u. Then, by Lemma 6, the function ρ is order-preserving on the set of c-positions in u. In particular, the word v has at least as many occurrences of c as u. This holds for all letters c in u. Hence, $|u| \leq |v|$. ◀

▶ **Example 8.** Consider $u = bacbaabada$ from Example 1 and let $k = 3$. As explained in Example 3, we must not delete both position 5 and position 6. However, we can delete positions 5 and 8 to obtain a $\sim_k$-equivalent word with the following attributes:

12	12	11	22	22	31	11	21
b	a	c	b	a	b	d	a

By Theorem 7, there is no shorter word in the same $\sim_k$-class.

5 Commutation

In the previous section, we described how to successively delete letters of a word in order to obtain a length-minimal $\sim_k$-equivalent word. It remains to show how to further transform a word of minimal length into shortlex normal form. In the first two statements, we give a sufficient condition which allows us to swap letters b and a while preserving the $\sim_k$-class.

▶ **Lemma 9.** *Consider two words $ubav$ and $uabv$ with $a, b \in A$. Let (x_ℓ, y_ℓ) denote the attribute of position ℓ in $ubav$, and let (x'_ℓ, y'_ℓ) denote the attribute of position ℓ in $uabv$. Suppose that $|ub| = i$ and that the attributes (x_i, y_i) and (x_{i+1}, y_{i+1}) satisfy $x_i = x_{i+1}$. Then all positions ℓ satisfy $x'_\ell = x_\ell$.*

Proof. Throughout this proof, we frequently rely on the following simple observation: If rs is a ranker of minimal length reaching position ℓ in a word w, then r is also of minimal length reaching position $r(w)$.

We can assume that $a \neq b$. It suffices to show that no ranker in R_{i+1}^{ubav} visits position i in $ubav$ and no ranker in R_{i+1}^{uabv} visits position i in $uabv$. This implies that for all $\ell \in \{1, \ldots, n\}$, no ranker in R_ℓ^{ubav} or in R_ℓ^{uabv} visits both i and $i+1$ in the corresponding words and thus, we have $R_i^{ubav} = R_{i+1}^{uabv}$ and $R_{i+1}^{ubav} = R_i^{uabv}$ as well as $R_\ell^{ubav} = R_\ell^{uabv}$ for $\ell \notin \{i, i+1\}$. Note that all rankers in R_i^{ubav} and in R_{i+1}^{ubav} have length $x_i = x_{i+1}$.

Suppose, for the sake of contradiction, that a ranker $r \in R_{i+1}^{ubav}$ visits position i in $ubav$. Then, we can write $r = s\mathsf{X}_b\mathsf{X}_a$ with $s\mathsf{X}_b(ubav) = i$. Note that $|s\mathsf{X}_b| \geq x_i$ by the definition of x_i. Since $x_{i+1} = x_i \leq |s\mathsf{X}_b|$, there exists a ranker of length at most $|s\mathsf{X}_b| < |r|$ reaching position $i+1$ in $ubav$, contradicting the choice of r.

Suppose that a ranker $r \in R_{i+1}^{uabv}$ visits position i in $uabv$. Let $r = s\mathsf{X}_a\mathsf{X}_b$ with $s\mathsf{X}_a(uabv) = i$. Note that $s\mathsf{X}_a(ubav) = i+1$ and, since $x_{i+1} = x_i$, there exists a ranker $\hat{s}$ of length at most $|s\mathsf{X}_a|$ such that $\hat{s}(ubav) = i$. Now, $\hat{s}$ is a ranker of length $|\hat{s}| \leq |s\mathsf{X}_a| < |r|$ with $\hat{s}(uabv) = i+1$, a contradiction to $r \in R_{i+1}^{uabv}$. ◀

▶ **Proposition 10.** *Let $ubav$ be a word with $|ub| = i$ and attributes $(x_i, y_i) = (x_{i+1}, y_{i+1})$ satisfying $x_i + y_i = k + 1$. Then $ubav \sim_k uabv$.*

Proof. Suppose that there exists a word w with $|w| \leq k$ such that $w \preccurlyeq ubav$ but $w \not\preccurlyeq uav$ and $w \not\preccurlyeq ubv$. Then we can write $w = w_1baw_2$ such that $w_1b \preccurlyeq ub$, $w_1b \not\preccurlyeq u$, $aw_2 \preccurlyeq av$, and $aw_2 \not\preccurlyeq v$. Thus, the word w_1b defines an X-ranker r with $r(ubav) = i$ and, similarly, aw_2

defines a Y-ranker s with $s(ubav) = i+1$. We see that $|r|+|s| = |w| \leq k$, but this contradicts $|r| + |s| \geq x_i + y_{i+1} = k + 1$. Therefore, every subword of $ubav$ of length at most k is also a subword of $uabv$.

By Lemma 9 and its left-right dual, the attributes of the positions i and $i+1$ in $uabv$ are both identical to (x_i, y_i). Therefore, the same reasoning as above shows that every subword of $uabv$ of length at most k is also a subword of $ubav$. This shows $ubav \sim_k uabv$. ◀

▶ **Example 11.** Let us reconsider the length-minimal word $u = bacbabda$ from Example 8 and let again $k = 3$. The attributes are as follows:

12	12	11	22	22	31	11	21
b	a	c	b	a	b	d	a

The attributes (x_4, y_4) and (x_5, y_5) at positions 4 and 5 satisfy $x_4 = x_5$, $y_4 = y_5$ and $x_4 + y_4 = k + 1$. By Proposition 10, we obtain $bacabbda \sim_k u$. Note that the attributes (x_1, y_1) and (x_2, y_2) at the first two positions satisfy $x_1 = x_2$, $y_1 = y_2$ but $x_1 + x_2 < k + 1$. And, in fact, $abcbabda \not\sim_k u$ since $abc \preccurlyeq abcbabda$ but $abc \not\preccurlyeq u$.

It remains to show that repeated application of the commutation rule described in Proposition 10 actually suffices to obtain the lexicographically smallest representative of a $\sim_k$-class. The next lemma shows, using canonical rankers, that indeed all length-minimal representatives of a $\sim_k$-class can be transformed into one another using this commutation rule.

▶ **Lemma 12.** *Let $u \sim_k v$ such that both words u and v have minimal length in their $\sim_k$-class. Let (x_ℓ, y_ℓ) denote the attribute of position ℓ of u. Consider two positions $i < j$ of u. If either $(x_i, y_i) \neq (x_j, y_j)$ or $x_i + y_i < k + 1$, then $r_i^u(v) < r_j^u(v)$.*

Proof. If i and j have the same label, then the claim follows from Lemma 6. In the remainder of this proof, let their labels be different. In particular, we cannot have $r_i^u(v) = r_j^u(v)$. Suppose $(x_i, y_i) \neq (x_j, y_j)$ or $x_i + y_i < k + 1$. If $x_i + y_j \geq k + 1$ and $x_j + y_i \geq k + 1$, then, by minimality and Proposition 2, we have $x_i + y_i = k + 1$ and $x_j + y_j = k + 1$. This yields $x_i + y_j = k + 1$ and $x_j + y_i = k + 1$. Thus, $x_i = x_i + x_j + y_j - k - 1 = x_j$ and, similarly, $y_i = y_i + x_j + y_j - k - 1 = y_j$; this shows $(x_i, y_i) = (x_j, y_j)$, a contradiction. Therefore, we have either $x_i + y_j \leq k$ or $x_j + y_i \leq k$.

Let p_i and p_j be the words defining the rankers r_i^u and r_j^u, respectively. Symmetrically to the definition of the canonical X-ranker, we could also define canonical Y-rankers s_i^u and s_j^u such that $s_i^u(u) = i$, $|s_i^u| = y_i$, $s_j^u(u) = j$, and $|s_j^u| = y_j$. If the label c of u at position i is the ℓ-th occurrence of the letter c in u, then, by Lemma 6, both r_i^u and s_i^u end up at the position with the ℓ-th occurrence of the letter c in v. This shows $r_i^u(v) = s_i^u(v)$. Similarly, we see that $r_j^u(v) = s_j^u(v)$. Let q_i and q_j be the words defining the rankers s_i^u and s_j^u, respectively.

First, let $x_i + y_j \leq k$. Then $p_i q_j \preccurlyeq u$ yields $p_i q_j \preccurlyeq v$ since $u \sim_k v$ and $|p_i q_j| = x_i + y_j \leq k$. This shows $r_i^u(v) < s_j^u(v) = r_j^u(v)$, as desired. Let now $x_j + y_i \leq k$ and assume $r_i^u(v) > r_j^u(v)$. Then $p_j q_i \preccurlyeq v$ yields $p_j q_i \preccurlyeq u$ and, thus, $j = r_j^u(u) < s_i^u(u) = i$. This is a contradiction; hence, $r_i^u(v) < r_j^u(v)$. ◀

Using the previous lemma, we can finally show that iterating the commutation procedure from Lemma 9 and Proposition 10 yields the desired shortlex normal form.

▶ **Theorem 13.** *Let $u = a_1 \cdots a_n$ with $a_i \in A$ be a length-minimal word within its $\sim_k$-class. Suppose that the attributes (x_i, y_i) for all positions $i < n$ satisfy the following implication:*

$$\text{If } (x_i, y_i) = (x_{i+1}, y_{i+1}) \text{ and } x_i + y_i = k + 1, \text{ then } a_i \leq a_{i+1}. \tag{1}$$

Then u is the shortlex normal form of its $\sim_k$-class.

Proof. Let v be the shortlex normal form of the $\sim_k$-class of u. We want to show that $u = v$. Let ρ map position i of u to position $r_i^u(v)$ of v. As we have seen in the proof of Theorem 7, the function ρ is bijective. It remains to show that ρ is order-preserving. By contradiction, assume that there are positions i and j of u with $i < j$ such that $\rho(i) > \rho(j)$; let i be minimal with this property and let $i = \rho(j)$, *i.e.*, we choose j to be the preimage of position i in v. We already know that $\rho(i) < \rho(j)$ in all of the following cases:

- $a_i = a_j$ (by Lemma 6),
- $(x_i, y_i) \neq (x_j, y_j)$ (by Lemma 12),
- $x_i + y_i < k + 1$ (again by Lemma 12).

Therefore, the only remaining case is $a_i \neq a_j$, $(x_i, y_i) = (x_j, y_j)$ and $x_i + y_i = k + 1$. First, suppose that $(x_i, y_i) = (x_\ell, y_\ell)$ for all $\ell \in \{i, \ldots, j\}$. Then, by the implication in Equation (1), we have $a_i \leq \cdots \leq a_j$. Since $a_i \neq a_j$, we have $a_i < a_j$. Now, u has the prefix $a_1 \cdots a_i$ and v has the prefix $a_1 \cdots a_{i-1} a_j$. In particular, u is lexicographically smaller than v; this is a contradiction. Next, suppose that there exists a position $\ell \in \{i, \ldots, j\}$ with $(x_i, y_i) \neq (x_\ell, y_\ell)$. Note that $i < \ell < j$. By Lemma 12, we have $\rho(i) < \rho(\ell)$ and $\rho(\ell) < \rho(j)$. In particular, we have $\rho(i) < \rho(j)$ in contradiction to our assumption. Altogether, this shows that $i < j$ and $\rho(i) > \rho(j)$ is not possible, *i.e.*, ρ is order-preserving. Hence, $u = v$. ◀

We summarize our knowledge on shortest elements of a $\sim_k$-class as follows. A word u has minimal length within its $\sim_k$-class if and only if all attributes (x_i, y_i) satisfy $x_i + y_i \leq k + 1$. The canonical rankers define a bijective mapping between any two shortest words u and v of a common $\sim_k$-class. This map preserves the labels and the attributes. It is almost order preserving, with the sole exception that $i < j$ could lead to $r_i^u(v) > r_j^u(v)$ whenever the attributes in u satisfy both $x_i + y_i = k + 1$ and $(x_i, y_i) = (x_\ell, y_\ell)$ for all $\ell \in \{i, \ldots, j\}$.

6 Computing shortlex normal forms

The results from the previous sections immediately lead to the following algorithm for computing shortlex normal forms. First, we successively delete single letters of the input word until the length is minimal. Let $a_1 \cdots a_n$ be the resulting word. In the second step, we lexicographically sort maximal factors $a_i \cdots a_j$ with attributes $(x_i, y_i) = \cdots = (x_j, y_j)$ and $x_i + y_i = k + 1$. We now improve the first step of this algorithm.

Algorithm 1 Computing the x-coordinates of $a_1 \cdots a_n$.

```
1: for all a ∈ A do n_a ← 1
2: for i ← 1, ..., n do
3:     suppose a_i = c
4:     x_i ← n_c
5:     n_c ← n_c + 1
6:     for all a ∈ A do n_a ← min(n_a, n_c)
```

The following lemma proves the correctness of Algorithm 1. Its running time is in $\mathcal{O}(|A|\, n)$ since there are n iterations of the main loop, and each iteration updates $|A|$ counters.

▶ **Lemma 14.** *Algorithm 1 computes the correct x-coordinates of the attributes of $a_1 \cdots a_n$.*

Proof. The algorithm reads the input word from left to right, letter by letter. In each step it updates some of its counters n_a. The semantics of the counters n_a is as follows: if the next letter a_i is c, then x_i is n_c. This invariant is true after the initialization in the first line.

Algorithm 2 Computing the y-coordinates of $a_1 \cdots a_n$ plus deletion.

```
1: for all a ∈ A do n_a ← 1
2: for i ← n, ..., 1 do
3:     suppose a_i = c
4:     if x_i + n_c ≤ k + 1 then
5:         y_i ← n_c
6:         n_c ← n_c + 1
7:         for all a ∈ A do n_a ← min(n_a, n_c)
8:     else
9:         position i is marked for deletion
```

Suppose that we start an iteration of the loop at letter $a_i = c$. Then the invariant tells us that $x_i = n_c$. If a_{i+1} were c, then one more step X_c would be needed for a ranker to reach position $i+1$, hence $n_c \leftarrow n_c + 1$. If a_{i+1} were some letter $a \neq c$, then we could either use the ranker corresponding to the old value n_a or we could use the ranker going to position i and from there do an X_a-modality; the latter would yield a ranker whose length is the new value of n_c. We choose the shorter of these two options. Since all counter values were correct before reading position i, there is no other counter n_a which needs to be updated before proceeding with position $i+1$. ◀

With Algorithm 2, we give a procedure for computing the y-coordinates of $a_1 \cdots a_n$ similar to Algorithm 1, but with the modification that we mark some letters for deletion. The positions marked for deletion depend on the number k in Simon's congruence $\sim_k$. The computed y-coordinates are those where all marked letters are actually deleted. We assume that the x-coordinates of the input word are already known.

The algorithm correctly computes the y-coordinates of the word where all marked letters are deleted. This follows from the left-right dual of Lemma 14 and the fact that the counters remain unchanged if a position is marked for deletion.

▶ **Lemma 15.** *Let u be the input for Algorithm 2 and let v be the word with all marked letters removed. Then $u \sim_k v$.*

Proof. Whenever a position i with label c is marked for deletion, the value x_i is correct since no letter to the left of position i is marked for deletion. The counter n_c would be the correct y-coordinate for position i if we deleted all positions which have been marked so far. By Proposition 2 we know that each deletion preserves the $\sim_k$-class. ◀

It remains to show that the x-coordinates are still correct for the resulting word in which all marked letters are deleted.

▶ **Lemma 16.** *Consider a word $u = a_1 \cdots a_n$ with x-coordinate x_ℓ at position ℓ. Let i be the maximal position of u such that $x_i + y_i > k+1$ and let $v = a_1 \cdots a_{i-1} a_{i+1} \cdots a_n$. The x-coordinate of position ℓ of v is denoted by x'_ℓ. Then, for all $j \in \{i+1, \ldots, n\}$, we have $x'_{j-1} = x_j$.*

Proof. It suffices to prove the statement $x'_{j-1} = x_j$ for all positions j of u reachable by a ranker of the form $r\mathsf{X}_c$ with $r(u) = i$ and $c \in A$. By contradiction, suppose that there exists some position $j = r\mathsf{X}_c(u)$ with $x'_{j-1} \neq x_j$ where $r(u) = i$ and $c \in A$; we choose $c \in A$ such that j is minimal with this property. Let $b = a_i$. We have to distinguish two cases.

First suppose that there is no b-position f with $i < f < j$ in u. The ranker r_j^u has to visit position i in u; otherwise $r_j^u(v) = j-1$ and $r_{j-1}^v(u) = j$, a contradiction to $x'_{j-1} \neq x_j$. This implies $x_i < x_j$. Moreover, position i is reachable from j in u with a single Y_b-modality, and hence, we have $x_i + y_i \leq (x_j - 1) + (y_j + 1) \leq k + 1$. This contradicts the choice of i.

Next, let f be the minimal b-position with $i < f < j$. In particular, we have $b \neq c$ because $j \neq f$ is the smallest c-position of u greater than i. Let $r\mathsf{X}_c$ be an X-ranker of length x_j such that $r\mathsf{X}_c(u) = j$. If $r(u) = i$, then $r(v) = f - 1$ and hence $r\mathsf{X}_c(v) = j - 1$. If $r(u) < i$, then the ranker $r\mathsf{X}_c$ does not visit the position i in u and we have $r\mathsf{X}_c(v) = j - 1$. Finally, if $r(u) > i$, then (by choice of c) the position $r(u) < j$ keeps its x-coordinate. In other words, there exists an X-ranker r' with $|r| = |r'|$ and $r'(v) = r(u) - 1$. It follows that $r'\mathsf{X}_c(v) = j - 1$. Therefore, in any case, there exists a ranker s of length at most x_j such that $s(v) = j - 1$. This shows $x'_{j-1} \leq x_j$, and together with $x'_{j-1} \neq x_j$ we obtain $x'_{j-1} < x_j$.

Consider an X-ranker $s\mathsf{X}_c$ of length $x'_{j-1} < x_j$ with $s\mathsf{X}_c(v) = j - 1$. We are still in the situation that there exists a b-position f in u with $i < f < j$. We cannot have $s(v) < i$ since otherwise $s(u) = s(v)$ and, thus, $s\mathsf{X}_c(u) = j$; the latter uses the fact that $b \neq c$. Let now $s(v) \geq i$ and write $s = t\mathsf{X}_d$. We have $t(v) < i$ since otherwise $t\mathsf{X}_c$ would be a shorter X-ranker with $t\mathsf{X}_c(v) = j - 1$. We have $d = b$: if $d \neq b$, then $s(v) = s(u) - 1$ and $s\mathsf{X}_c(u) = j$; this would show $x'_{j-1} \geq x_j$, thereby contradicting $x'_{j-1} < x_j$. It follows that $s(u) = i$ and $s\mathsf{X}_c(u) = j$. As before, this is a contradiction. This completes the proof that $x'_{j-1} = x_j$. ◀

▶ **Example 17.** Let $u = bacbaabada$ be the word from Example 1 and let $k = 3$. Suppose that the alphabet $A = \{a, b, c, d\}$ is ordered by $a < b < c < d$. The attributes of u are as follows:

12	12	11	22	23	32	31	42	11	21
b	a	c	b	a	a	b	a	d	a

Note that each of the attributes (x_i, y_i) at positions $i \in \{5, 6, 8\}$ satisfies the condition $x_i + y_i > k + 1$. As seen in Example 3 we must not delete all these positions. The algorithm only marks positions 6 and 8 for deletion and takes these deletions into account when computing the y-coordinates of the remaining letters:

12	12	11	22	22	3	31	4	11	21
b	a	c	b	a	a	b	a	d	a

The letters are now sorted as in Example 11 and the resulting normal form is *bacabbda*.

The following lemma allows us to improve the estimated time for the sorting step of the main algorithm by showing that any sequence of letters which needs to be sorted contains every letter at most once.

▶ **Lemma 18.** *Consider a word $uaav$ with $a \in A$ and $|ua| = i$. Then $x_i \neq x_{i+1}$ and $y_i \neq y_{i+1}$.*

Proof. Suppose $x_i = x_{i+1}$. Let r_i and r_{i+1} be X-rankers with $r_i(uaav) = i$, $|r_i| = x_i$, $r_{i+1}(uaav) = i + 1$, and $|r_{i+1}| = x_{i+1} = x_i$. Let $r_{i+1} = s\mathsf{X}_a$. If $s(uaav) < i$, then $i + 1 = r_{i+1}(uaav) = s\mathsf{X}_a(uaav) \leq i$. If $s(uaav) = i$, then $|s| = x_i - 1 < x_i = |r_i|$ contradicts the definition of x_i. Therefore, we cannot have $x_i = x_{i+1}$. Symmetrically, we cannot have $y_i = y_{i+1}$. ◀

We are now able to state our main result.

▶ **Theorem 19.** *One can compute the shortlex normal form of a word w of length n, including all attributes of the normal form, with $\mathcal{O}(|A|\, n)$ arithmetic operations and with bit complexity $\mathcal{O}(|A|\, n \log n)$. Alternatively, the computation can be done in deterministic space $\mathcal{O}(|A| \log n)$.*

Proof. The attributes of the normal form can be computed as described in Algorithms 1 and 2. The normal form itself is obtained by filtering out all positions i where the corresponding attribute (x_i, y_i) satisfies $x_i + y_i \leq k + 1$ and by sorting blocks of letters with the same attributes satisfying $x_i + y_i = k + 1$. By Lemma 18, the sorting step can be performed by reading each such block of letters, storing all letters appearing in the block and only outputting all these letters in sorted order once the next block is reached.

If we assume that the comparison of two letters and the modification of the counters is possible in constant time, then running Algorithm 2 on the output of Algorithm 1 takes $\mathcal{O}(|A|\, n)$ steps for input words of length n over alphabet A: for each position of the input word, we need to update $|A|$ counters. Over a fixed alphabet, the resulting algorithm runs in linear time – even if k is part of the input. We could bound all arithmetic operations by $k + 2$, *i.e.*, by replacing the usual addition by $n \oplus m = \min(k + 2, n + m)$. This way, each counter and all results of arithmetic operations would require only $\mathcal{O}(\log k) \subseteq \mathcal{O}(\log n)$ bits. Similarly, $\mathcal{O}(\log |A|) \subseteq \mathcal{O}(\log n)$ bits are sufficient to encode the letters. This leads to a bit complexity of $\mathcal{O}(|A|\, n \log n)$. Note that if $k > n$, then the $\sim_k$-class of the input is a singleton and we can immediately output the input without any further computations. If $|A| > n$, then we could replace A by the letters which occur in the input word.

For the $\mathcal{O}(|A| \log n)$ space algorithm, one can again use Algorithms 1 and 2 to compute the attributes of each position. To compute the shortlex normal form, we do not store all the attributes but use the standard recomputation technique to decide whether a letter gets deleted. The sorting step can be implemented by repeatedly scanning each block of positions with common attributes (x, y) satisfying $x + y = k + 1$. A single scan checks, for a fixed letter $a \in A$, whether a occurs in the block. This is repeated for every $a \in A$ in ascending order. The attributes of the currently investigated block and the current letter a can be stored in space $\mathcal{O}(\log n)$. ◀

7 Summary and Outlook

We considered Simon's congruence $\sim_k$ for piecewise testable languages. The main contribution of this paper is an $\mathcal{O}(|A|\, n)$ algorithm for computing the shortlex normal form of a word of length n within its $\sim_k$-class; surprisingly, this bound also holds if k is part of the input. The algorithm can be adapted to work in deterministic logarithmic space over a fixed alphabet. As a consequence, on input u, v, k, one can test in time $\mathcal{O}(|A|\, |uv|)$ whether $u \sim_k v$ holds. The main tool are the minimal lengths of X-rankers and Y-rankers reaching any position of a word. The key ingredient in the proofs are the so-called canonical rankers.

It would be interesting to see whether the space complexity for an arbitrary alphabet can be further improved from $\mathcal{O}(|A| \log n)$ to nondeterministic log-space or even deterministic log-space if the alphabet A is part of the input. In addition, we still lack corresponding lower bounds for the computation of shortlex normal forms and for the test of whether $u \sim_k v$ holds.

References

1. Sung Cho and Dung T. Huynh. Finite automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88:96–116, 1991.
2. Jie Fu, Jeffrey Heinz, and Herbert G. Tanner. An algebraic characterization of strictly piecewise languages. In Mitsunori Ogihara and Jun Tarui, editors, *Theory and Applications of Models of Computation*, volume 6648 of *LNCS*, pages 252–263, Berlin, Heidelberg, 2011. Springer.
3. Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society. Third Series*, 2:326–336, 1952.
4. Stepan Holub, Tomás Masopust, and Michaël Thomazo. Alternating towers and piecewise testable separators. *CoRR*, abs/1409.3943, 2014.
5. J. E. Hopcroft and R. Karp. A linear algorithm for testing equivalence of finite automata. Technical Report 71-114, Dept. of Computer Science, Cornell U, December 1971.
6. P. Karandikar, M. Kufleitner, and Ph. Schnoebelen. On the index of Simon's congruence for piecewise testability. *Information Processing Letters*, 115(4):515–519, 2015.
7. Kamilla Kátai-Urbán, Péter Pál Pach, Gabriella Pluhár, András Pongrácz, and Csaba Szabó. On the word problem for syntactic monoids of piecewise testable languages. In *Semigroup Forum*, volume 84, pages 323–332. Springer, 2012.
8. O. Klíma, M. Kunc, and L. Polák. Deciding k-piecewise testability, submitted.
9. Ondřej Klíma and Libor Polák. Alternative automata characterization of piecewise testable languages. In Marie-Pierre Béal and Olivier Carton, editors, *DLT 2013, Proceedings*, volume 7907 of *Lecture Notes in Computer Science*, pages 289–300. Springer, 2013.
10. Leonid Aryeh Kontorovich, Corinna Cortes, and Mehryar Mohri. Kernel methods for learning languages. *Theoretical Computer Science*, 405(3):223–236, 2008.
11. Tomás Masopust and Michaël Thomazo. On the complexity of k-piecewise testability and the depth of automata. In *DLT 2015, Proceedings*, volume 9168 of *Lecture Notes in Computer Science*, pages 364–376. Springer, 2015.
12. Péter Pál Pach. Solving equations under Simon's congruence. In *JHSDM 2015, Proceedings*, pages 201–206, 2015.
13. Péter Pál Pach. Normal forms under Simon's congruence. *Semigroup Forum*, Dec 2017.
14. James Rogers, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Visscher, David Wellcome, and Sean Wibel. On languages piecewise testable in the strict sense. In Christian Ebert, Gerhard Jäger, and Jens Michaelis, editors, *The Mathematics of Language*, pages 255–265, Berlin, Heidelberg, 2010. Springer.
15. José Ruiz and Pedro García. Learning k-piecewise testable languages from positive data. In Laurent Miclet and Colin de la Higuera, editors, *Grammatical Interference: Learning Syntax from Sentences*, pages 203–210, Berlin, Heidelberg, 1996. Springer.
16. Thomas Schwentick, Denis Thérien, and Heribert Vollmer. Partially-ordered two-way automata: A new characterization of DA. In *DLT 2001, Proceedings*, volume 2295 of *LNCS*, pages 239–250. Springer, 2002.
17. Imre Simon. *Hierarchies of events with dot-depth one*. PhD thesis, University of Waterloo, 1972.
18. Imre Simon. Piecewise testable events. In *Autom. Theor. Form. Lang., 2nd GI Conf.*, volume 33 of *LNCS*, pages 214–222. Springer, 1975.
19. Jacques Stern. Characterization of some classes of regular events. *Theor. Comput. Sci.*, 35:17–42, 1985.
20. Wolfgang Thomas. Classifying regular events in symbolic logic. *J. Comput. Syst. Sci.*, 25:360–376, 1982.
21. A. N. Trahtman. Piecewise and local threshold testability of DFA. In Rusins Freivalds, editor, *FCT 2001, Proceedings*, volume 2138 of *LNCS*, pages 347–358. Springer, 2001.
22. Philipp Weis and Neil Immerman. Structure theorem and strict alternation hierarchy for FO^2 on words. *Log. Methods Comput. Sci.*, 5(3):1–23, 2009.

On the Price of Independence for Vertex Cover, Feedback Vertex Set and Odd Cycle Transversal

Konrad K. Dabrowski[1]
Department of Computer Science, Durham University, UK
konrad.dabrowski@durham.ac.uk
https://orcid.org/0000-0001-9515-6945

Matthew Johnson[2]
Department of Computer Science, Durham University, UK
matthew.johnson2@durham.ac.uk
https://orcid.org/0000-0002-7295-2663

Giacomo Paesani
Department of Computer Science, Durham University, UK
giacomo.paesani@durham.ac.uk
https://orcid.org/0000-0002-2383-1339

Daniël Paulusma[3]
Department of Computer Science, Durham University, UK
daniel.paulusma@durham.ac.uk
https://orcid.org/0000-0001-5945-9287

Viktor Zamaraev[4]
Department of Computer Science, Durham University, UK
viktor.zamaraev@durham.ac.uk
https://orcid.org/0000-0001-5755-4141

Abstract

Let $\mathrm{vc}(G)$, $\mathrm{fvs}(G)$ and $\mathrm{oct}(G)$ denote, respectively, the size of a minimum vertex cover, minimum feedback vertex set and minimum odd cycle transversal in a graph G. One can ask, when looking for these sets in a graph, how much bigger might they be if we require that they are independent; that is, what is the *price of independence*? If G has a vertex cover, feedback vertex set or odd cycle transversal that is an independent set, then we let, respectively, $\mathrm{ivc}(G)$, $\mathrm{ifvs}(G)$ or $\mathrm{ioct}(G)$ denote the minimum size of such a set. We investigate for which graphs H the values of $\mathrm{ivc}(G)$, $\mathrm{ifvs}(G)$ and $\mathrm{ioct}(G)$ are bounded in terms of $\mathrm{vc}(G)$, $\mathrm{fvs}(G)$ and $\mathrm{oct}(G)$, respectively, when the graph G belongs to the class of H-free graphs. We find complete classifications for vertex cover and feedback vertex set and an almost complete classification for odd cycle transversal (subject to three non-equivalent open cases).

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases vertex cover, feedback vertex set, odd cycle transversal, price of independence

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.63

[1] Supported by EPSRC (EP/K025090/1) and the Leverhulme Trust (RPG-2016-258).
[2] Supported by the Leverhulme Trust (RPG-2016-258).
[3] Supported by EPSRC (EP/K025090/1) and the Leverhulme Trust (RPG-2016-258).
[4] Supported by EPSRC (EP/P020372/1).

1 Introduction

We define a number of transversals of the vertex set of a graph G. A set $S \subseteq V(G)$ is a *vertex cover* if for every edge $uv \in E(G)$, at least one of u and v is in S, or, equivalently, if the graph $G - S$ contains no edges. A set $S \subseteq V(G)$ is a *feedback vertex set* if for every cycle in G, at least one vertex of the cycle is in S, or, equivalently, if the graph $G - S$ is a forest. A set $S \subseteq V(G)$ is an *odd cycle transversal* if for every cycle in G containing an odd number of vertices, at least one vertex of the cycle is in S, or, equivalently, if the graph $G - S$ is bipartite. For each of these transversals, one usually wishes to investigate how small they can be and there is a vast research literature on this topic.

One can add an additional constraint: require the transversal to be an *independent set*, that is, a set of vertices that are pairwise non-adjacent. It might be possible that no such transversal exists under this constraint. For example, a graph G has an independent vertex cover if and only if G is bipartite. We are interested in the following research question:

> *How is the minimum size of a transversal in a graph affected by adding the requirement that the transversal is independent?*

Of course, this question can be interpreted in many ways; for example, one might ask about the computational complexity of finding the transversals. In this paper, we focus on the following: for the three transversals introduced above, is the size of a smallest possible independent transversal (assuming one exists) bounded in terms of the minimum size of a transversal? That is, one might say, what is the *price of independence*?

To the best of our knowledge, the term price of independence was first used by Camby [4] in a recent unpublished manuscript. She considered dominating sets of graphs (sets of vertices such that every vertex outside the set has a neighbour in the set). As she acknowledged, though first to coin the term, she was building on past work. In fact, Camby and her co-author Plein had given a forbidden induced subgraph characterization of those graphs G for which, for every induced subgraph of G, there are minimum size dominating sets that are already independent [6], and there are a number of further papers on the topic of the price of independence for dominating sets (see the discussion in [4]).

We observe that this incipient work on the price of independence is a natural companion to recent work on the *price of connectivity*, investigating the relationship between minimum size transversals and minimum size connected transversals (which, in contrast to independent transversals, will always exist for the transversals we consider, assuming the input graph is connected). This work began with the work of Cardinal and Levy in their 2010 paper [8] and has since been taken in several directions; see, for example, [1, 5, 7, 9, 11, 12].

In this paper, as we broaden the study of the price of independence by investigating the three further transversals defined above, we will concentrate on classes of graphs defined by a single forbidden induced subgraph H, just as was done for the price of connectivity [1, 12]. That is, for a graph H, we ask what, for a given type of transversal, is the price of independence in the class of H-free graphs? The ultimate aim in each case is to find a dichotomy that allows us to say, given H, whether or not the size of a minimum size independent transversal can be bounded in terms of the size of a minimum transversal. We briefly give some necessary definitions and notation before presenting our results.

A *colouring* of a graph G is an assignment of positive integers (called *colours*) to the vertices of G such that if two vertices are adjacent, then they are assigned different colours. A graph is *k-colourable* if there is a colouring that only uses colours from the set $\{1, \ldots, k\}$. Equivalently, a graph is k-colourable if we can partition its vertex set into k (possibly empty)

independent sets (called *colour classes* or *partition classes*). For $s, t \geq 0$, let $K_{s,t}$ denote the complete bipartite graph with partition classes of size s and t, respectively (note that $K_{s,t}$ is edgeless if $s = 0$ or $t = 0$). For $r \geq 0$, the graph $K_{1,r}$ is also called the $(r+1)$-vertex *star*; if $r \geq 2$ we say that the vertex in the partition class of size 1 is the *central vertex* of this star. For $n \geq 1$, let P_n and K_n denote the path and complete graph on n vertices, respectively. For $n \geq 3$, let C_n denote the cycle on n vertices. For $r \geq 1$, let $K^+_{1,r}$ denote the graph obtained from $K_{1,r}$ by subdividing one edge. The *disjoint union* $G + H$ of two vertex-disjoint graphs G and H is the graph with vertex set $V(G) \cup V(H)$ and edge set $E(G) \cup E(H)$. We denote the disjoint union of r copies of a graph G by rG.

The Price of Independence for Vertex Cover. As mentioned above, a graph has an independent vertex cover if and only if it is bipartite. For a bipartite graph G, let $\mathrm{vc}(G)$ denote the size of a minimum vertex cover, and let $\mathrm{ivc}(G)$ denote the size of a minimum independent vertex cover. Given a class $\mathcal{X}$ of bipartite graphs, we say that $\mathcal{X}$ is ivc-bounded if there is a function $f : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ such that $\mathrm{ivc}(G) \leq f(\mathrm{vc}(G))$ for every $G \in \mathcal{X}$ and we say that $\mathcal{X}$ is ivc-unbounded if no such function exists, that is, if there is a k such that for every $s \geq 0$ there is a graph G in $\mathcal{X}$ with $\mathrm{vc}(G) \leq k$, but $\mathrm{ivc}(G) \geq s$.

In our first main result, proven in Section 2, we determine for every graph H, whether or not the class of H-free bipartite graphs is ivc-bounded.

▶ **Theorem 1.** *Let H be a graph. The class of H-free bipartite graphs is* ivc*-bounded if and only if H is an induced subgraph of $K_{1,r} + rP_1$ or $K^+_{1,r}$ for some $r \geq 1$.*

The Price of Independence for Feedback Vertex Set. A graph has an independent feedback vertex set if and only if its vertex set can be partitioned into an independent set and a set of vertices that induces a forest; graphs that have such a partition are said to be *near-bipartite*. In fact, minimum size independent feedback vertex sets have been the subject of much research from a computational perspective: to find such a set is, in general, NP-hard, but there are fixed-parameter tractable algorithms and polynomial-time algorithms for certain graph classes; we refer to [2] for further details. For a near-bipartite graph G, let $\mathrm{fvs}(G)$ denote the size of a minimum feedback vertex set, and let $\mathrm{ifvs}(G)$ denote the size of a minimum independent feedback vertex set. Given a class $\mathcal{X}$ of near-bipartite graphs, we say that $\mathcal{X}$ is ifvs-bounded if there is a function $f : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ such that $\mathrm{ifvs}(G) \leq f(\mathrm{fvs}(G))$ for every $G \in \mathcal{X}$ and ifvs-unbounded otherwise.

In our second main result, proven in Section 3, we determine for every graph H, whether or not the class of H-free near-bipartite graphs is ifvs-bounded.

▶ **Theorem 2.** *Let H be a graph. The class of H-free near-bipartite graphs is* ifvs*-bounded if and only if H is isomorphic to $P_1 + P_2$, a star or an edgeless graph.*

The Price of Independence for Odd Cycle Transversal. A graph has an independent odd cycle transversal S if and only if it has a 3-colouring, since, by definition, we are requesting that S is an independent set of G such that $G - S$ has a 2-colouring. For a 3-colourable graph G, let $\mathrm{oct}(G)$ denote the size of a minimum odd cycle transversal, and let $\mathrm{ioct}(G)$ denote the size of a minimum independent odd cycle transversal. Given a class $\mathcal{X}$ of 3-colourable graphs, we say that $\mathcal{X}$ is ioct-bounded if there is a function $f : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ such that $\mathrm{ioct}(G) \leq f(\mathrm{oct}(G))$ for every $G \in \mathcal{X}$ and ioct-unbounded otherwise.

In our third main result, proven in Section 4, we address the question of whether or not, for a graph H, the class of H-free 3-colourable graphs is ifvs-bounded. Here, we do not have a complete dichotomy due to three missing cases, as we will discuss at the end of Section 4.

▶ **Theorem 3.** *Let H be a graph. The class of H-free 3-colourable graphs is* ioct*-bounded:*
- *if H is an induced subgraph of P_4 or $K_{1,3} + sP_1$ for some $s \geq 0$ and*
- *only if H is an induced subgraph of $K^+_{1,4}$ or $K_{1,4} + sP_1$ for some $s \geq 0$.*

Further Notation. Let G be a graph. For $x \in V(G)$, the *(open) neighbourhood* $N(v)$ of v is the set of vertices adjacent to v; the *closed neighbourhood* $N[v] = N(v) \cup \{v\}$. For $x \in V(G)$, let $\deg(v) = |N(v)|$ denote the *degree* of v. If $S \subseteq V(G)$, then $N(S) = \{v \in V(G) \setminus S \mid \exists u \in S, uv \in E(G)\}$ and $N[S] = N(S) \cup S$. For two vertices $x, y \in V(G)$, let $\text{dist}(x, y)$ denote the length of a shortest path from x to y (let $\text{dist}(x, y) = \infty$ if x and y are in different connected components of G). For $S \subseteq V(G)$, let $G[S]$ be the *induced subgraph of G on S*, that is, the graph with vertex set S, where two vertices in S are adjacent if and only if they are adjacent in G. For $S \subseteq V(G)$, let $G - S$ denote $G[V(G) \setminus S]$. A vertex $y \in V(G)$ is complete (resp. anti-complete) to a set $X \subseteq V(G)$ if y is adjacent (resp. not adjacent) to every vertex in X. A set $Y \subseteq V(G)$ is complete (resp. anti-complete) to a set $X \subseteq V(G)$ if every vertex of Y is complete (resp. anti-complete) to X. A vertex $v \in V(G)$ is *dominating* if it is complete to $V(G) \setminus \{v\}$. The *complement* $\overline{G}$ of a graph G has the same vertex set as G and an edge between two distinct vertices if and only if these vertices are not adjacent in G. A graph is *complete multi-partite* if its vertex set can be partitioned into independent sets that are complete to each other. For a set of graphs $\{H_1, \ldots, H_s\}$, a graph is said to be $(H_1, \ldots, H_s)$*-free* if it contains no induced subgraph isomorphic to a graph in the set.

2 Vertex Cover

We start with a useful lemma.

▶ **Lemma 4.** *Let $r, s \geq 1$. If G is a $(K_{1,r} + sP_1)$-free bipartite graph with bipartition (X, Y) such that $|X|, |Y| \geq rs + r - 1$, then either:*
- *every vertex of G has degree less than r or*
- *fewer than s vertices of X have more than $s - 1$ non-neighbours in Y and fewer than s vertices of Y have more than $s - 1$ non-neighbours in X.*

Proof. Let G be a $(K_{1,r} + sP_1)$-free bipartite graph with bipartition (X, Y) such that $|X|, |Y| \geq rs + r - 1$. No vertex in X can have both r neighbours and s non-neighbours in Y, otherwise G would contain an induced $K_{1,r} + sP_1$. Therefore every vertex in X has degree either at most $r - 1$ or at least $|Y| - (s - 1) \geq rs + r - s$. By symmetry, we may assume that there is a vertex $x \in X$ of degree at least r. Suppose, for contradiction, that there is a set $X' \subseteq X$ of s vertices, each of which has more than $s - 1$ non-neighbours in Y. Then every vertex of X' has degree at most $r - 1$. Since $\deg(x) \geq rs + r - s = s(r - 1) + r$, there must be a set $Y' \subseteq N(x)$ of r neighbours of x that have no neighbours in X'. Then $G[\{x\} \cup Y' \cup X']$ is a $K_{1,r} + sP_1$, a contradiction. It follows that fewer than s vertices in X have more than $s - 1$ non-neighbours in Y. Since $|X| \geq r + (s - 1)$, there is a set $X'' \subsetneq X$ of r vertices, each of which have at most $s - 1$ non-neighbours in Y. Since $|Y| > r(s - 1)$, there must be a vertex $y \in Y$ that is complete to X'', and therefore has $\deg(y) \geq r$. Repeating the above argument, it follows that fewer than s vertices of Y have more than $s - 1$ non-neighbours in X. This completes the proof. ◀

Recall that a graph has an independent vertex cover if and only if it is bipartite.

▶ **Lemma 5.** *Let $r, s \geq 1$. If G is a $(K_{1,r} + sP_1)$-free bipartite graph, then* $\text{ivc}(G) \leq \text{vc}(G)r + rs$.

Proof. Let G be a $(K_{1,r} + sP_1)$-free bipartite graph. Fix a bipartition (X, Y) of G. Let S be a minimum vertex cover of G, so $|S| = \mathrm{vc}(G)$. We may assume that $\mathrm{vc}(G) \geq 2$, otherwise $\mathrm{ivc}(G) = \mathrm{vc}(G)$, in which case we are done. We may also assume that $|X|, |Y| > \mathrm{vc}(G)r + rs > rs + r - 1$, otherwise X or Y is an independent vertex cover of the required size, and we are done. If every vertex of G has degree at most $r - 1$, then $S' = (S \cap Y) \cup (N(S \cap X))$ is an independent vertex cover in G of size at most $\mathrm{vc}(G)(r - 1)$, and we are done. By Lemma 4, we may therefore assume that fewer than s vertices of X have more than $s-1$ non-neighbours in Y. We will show that this leads to a contradiction. Since $|X|, |Y| \geq \mathrm{vc}(G) + s$, there must be a set S' of $\mathrm{vc}(G) + 1$ vertices in X that have at least $\mathrm{vc}(G) + 1$ neighbours in Y. If a vertex $x \in V(G)$ has degree at least $\mathrm{vc}(G) + 1$, then $|N(x)| > |S|$, so $x \in S$. Therefore every vertex of S' must be in S, contradicting the fact that $|S'| = \mathrm{vc}(G) + 1 > \mathrm{vc}(G) = |S|$. ◀

▶ **Lemma 6.** *Let $r \geq 2$. If G is a $K^+_{1,r}$-free bipartite graph, then* $\mathrm{ivc}(G) \leq (\mathrm{vc}(G))^2(r - 1)$.

Proof. Clearly it is sufficient to prove the lemma for connected graphs G. Let G be a connected $K^+_{1,r}$-free bipartite graph. Fix a bipartition (X, Y) of G. Let S be a minimum vertex cover of G, so $|S| = \mathrm{vc}(G)$. We may assume that $\mathrm{vc}(G) \geq 2$, otherwise $\mathrm{ivc}(G) = \mathrm{vc}(G)$ and we are done. We may also assume that $|X|, |Y| > (\mathrm{vc}(G))^2(r - 1)$, otherwise X or Y is an independent vertex cover of the required size.

If there are two vertices $x, y \in X$ with $\mathrm{dist}(x, y) = 2$ and $\deg(x) \geq \deg(y) + (r - 1)$, then x, y, a common neighbour of x and y, and $r - 1$ vertices from $N(x) \setminus N(y)$ would induce a $K^+_{1,r}$ in G, a contradiction. Therefore, if $x, y \in X$ with $\mathrm{dist}(x, y) = 2$, then $|\deg(x) - \deg(y)| \leq r - 2$, so $|\deg(x) - \deg(y)| \leq (\frac{r-2}{2})\,\mathrm{dist}(x, y)$. By the triangle inequality and induction, it follows that if $x, y \in X$, then $|\deg(x) - \deg(y)| \leq (\frac{r-2}{2})\,\mathrm{dist}(x, y)$. Observe that $\mathrm{vc}(P_{2\,\mathrm{vc}(G)+2}) = \mathrm{vc}(G) + 1$, so G must be $P_{2\,\mathrm{vc}(G)+2}$-free. Since G is connected, it follows that if $x, y \in V(G)$, then $\mathrm{dist}(x, y) < 2\,\mathrm{vc}(G) + 1$. We conclude that if $x, y \in X$, then $|\deg(x) - \deg(y)| \leq \mathrm{vc}(G)(r - 2)$. Note that if a vertex $x \in V(G)$ has degree at least $\mathrm{vc}(G) + 1$, then $|N(x)| > |S|$ and so $x \in S$.

Since $|X| > (\mathrm{vc}(G))^2(r - 1) > \mathrm{vc}(G) = |S|$, there must be a vertex $y \in X \setminus S$. Since $y \in X \setminus S$, it follows that $\deg(y) \leq \mathrm{vc}(G)$. It follows that $\deg(x) \leq \deg(y) + \mathrm{vc}(G)(r - 2) \leq \mathrm{vc}(G)(r - 1)$ for all $x \in X$. We conclude that $S' = (S \cap Y) \cup (N(S \cap X))$ is an independent vertex cover in G of size at most $(\mathrm{vc}(G))^2(r - 1)$. This completes the proof. ◀

We are now ready to prove the main result of this section.

▶ **Theorem 1 (restated).** *Let H be a graph. The class of H-free bipartite graphs is* ivc*-bounded if and only if H is an induced subgraph of $K_{1,r} + rP_1$ or $K^+_{1,r}$ for some $r \geq 1$.*

Proof. If H is an induced subgraph of $K_{1,r} + rP_1$ or $K^+_{1,r}$ for some r, then Lemma 5 or 6, respectively, implies that the class of H-free bipartite graphs is ivc-bounded.

Now let H be a graph and suppose that there is a function $f : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ such that $\mathrm{ivc}(G) \leq f(\mathrm{vc}(G))$ for all H-free bipartite graphs G. We will show that H is an induced subgraph of $K_{1,r} + rP_1$ or $K^+_{1,r}$ for some r.

For $r \geq 1$, $s \geq 2$, let D^r_s denote the graph formed from $2K_{1,s}$ and P_{2r} by identifying the two end-vertices of the P_{2r} with the central vertices of the respective $K_{1,s}$'s (see also Figure 1; a graph of the form D^1_s is said to be a *double-star*). It is easy to verify that $\mathrm{vc}(D^r_s) = r + 1$ and $\mathrm{ivc}(D^r_s) = r + s$. Therefore, for every $r \geq 1$, $D^r_{f(r+1)}$ cannot be H-free by definition of f. Note that for $r, s, t \geq 1$, if $s \leq t$ then D^r_s is an induced subgraph of D^r_t. Therefore, for each $r \geq 1$, there must be an s such that D^r_s is not H-free. In other words, for each $r \geq 1$, H must be an induced subgraph of D^r_s for some s.

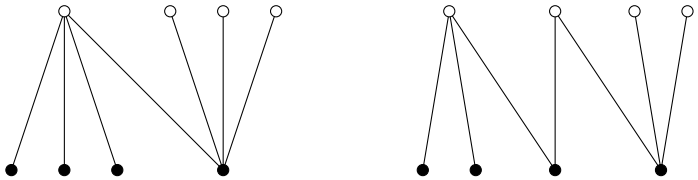


Figure 1 The graphs D_3^1 and D_2^2. The black vertices form a minimum independent vertex cover.

In particular, this means that we may assume that H is an induced subgraph of D_t^1 for some $t \geq 1$. If H contains at most one of the central vertices of the stars that form the D_t^1, then H is an induced subgraph of $K_{1,t} + tP_1$ and we are done, so we may assume H contains both central vertices. If one of these central vertices has at most one neighbour that is not a central vertex, then H is an induced subgraph of $K_{1,t+1}^+$, and we are done. We may therefore assume that H contains an induced D_2^1. However, for every $s \geq 1$, D_s^2 is D_2^1-free and therefore H-free. This contradiction completes the proof. ◀

3 Feedback Vertex Set

Recall that a graph has an independent feedback vertex set if and only if it is near-bipartite.

▶ **Lemma 7.** *If G is a $(P_1 + P_2)$-free near-bipartite graph, then* $\text{ifvs}(G) = \text{fvs}(G)$.

Proof. Let G be a $(P_1 + P_2)$-free near-bipartite graph. Note that $\overline{G}$ is a P_3-free graph, so $\overline{G}$ is a disjoint union of cliques. It follows that G is a complete multi-partite graph, say with a partition of its vertex sets into k non-empty independent sets $V_1, \ldots, V_k$. We may assume that $k \geq 2$, otherwise G is an edgeless graph, in which case $\text{ifvs}(G) = \text{fvs}(G) = 0$ and we are done. Since G is near-bipartite, it contains an independent set I such that $G - I$ is a forest. Note that $I \subseteq V_i$ for some $i \in \{1, \ldots, k\}$. Since near-bipartite graphs are 3-colourable, it follows that $k \leq 3$. Furthermore, if $k = 3$, then $|V_j| = 1$ for some $j \in \{1, 2, 3\} \setminus \{i\}$, otherwise $G - I$ would contain an induced C_4, a contradiction. In other words G is either a complete bipartite graph or the graph formed from a complete bipartite graph by adding a dominating vertex.

First suppose that $k = 2$, so G is a complete bipartite graph. Without loss of generality assume that $|V_1| \geq |V_2| \geq 1$. Let S be a feedback vertex set of G. If there are two vertices in $V_1 \setminus S$ and two vertices in $V_2 \setminus S$, then these vertices would induce a C_4 in $G - S$, a contradiction. Therefore S must contain all but at most one vertex of V_1 or all but at most one vertex of V_2, so $\text{fvs}(G) \geq \min(|V_1| - 1, |V_2| - 1) = |V_2| - 1$. Let I be a set consisting of $|V_2| - 1$ vertices of V_2. Then I is independent and $G - I$ is a star, so I is an independent feedback vertex set. It follows that $\text{ifvs}(G) \leq |V_2| - 1$. Since $\text{fvs}(G) \leq \text{ifvs}(G)$, we conclude that $\text{ifvs}(G) = \text{fvs}(G)$ in this case.

Now suppose that $k = 3$, so G is obtained from a complete bipartite graph by adding a dominating vertex. Without loss of generality assume that $|V_1| \geq |V_2| \geq |V_3| = 1$. Let S be a feedback vertex set of G. By the same argument as in the $k = 2$ case, S must contain all but at most one vertex of V_1 or all but at most one vertex of V_2. If there is a vertex in $V_i \setminus S$ for all $i \in \{1, 2, 3\}$, then these three vertices would induce a C_3 in $G - S$, a contradiction. Therefore S must contain every vertex in V_i for some $i \in \{1, 2, 3\}$. Since $|V_1| \geq |V_2| \geq |V_3| = 1$, it follows that $|S| \geq \min(|V_2| - 1 + |V_3|, |V_2|) = |V_2|$. Therefore $\text{fvs}(G) \geq |V_2|$. Now V_2 is an independent set and $G - V_2$ is a star, so V_2 is an independent feedback vertex set. It follows that $\text{ifvs}(G) \leq |V_2|$. Since $\text{fvs}(G) \leq \text{ifvs}(G)$, we conclude that $\text{ifvs}(G) = \text{fvs}(G)$. ◀

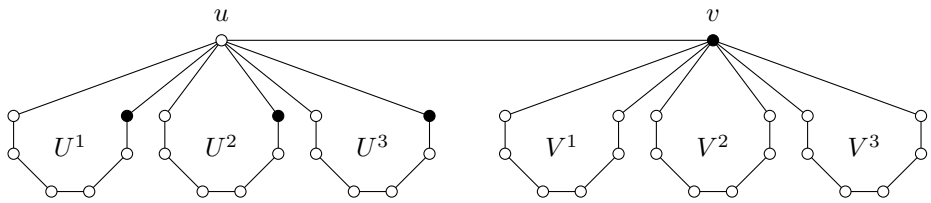


Figure 2 The graph S_3^3.

▶ **Lemma 8.** *If $r \geq 1$ and G is a $K_{1,r}$-free near-bipartite graph, then* $\text{ifvs}(G) \leq (2r^2 - 5r + 3)\,\text{fvs}(G)$.

Proof. Fix integers $k \geq 0$ and $r \geq 1$. Suppose G is a $K_{1,r}$-free near-bipartite graph with a feedback vertex set S such that $|S| = k$. Since G is near-bipartite, $V(G)$ can be partitioned into an independent set V_1 and a set $V(G) \setminus V_1$ that induces a forest in G. Since forests are bipartite, we can partition $V(G) \setminus V_1$ into two independent sets V_2 and V_3.

Suppose $x \in V_i$ for some $i \in \{1,2,3\}$. Then x has no neighbours in V_i since V_i is an independent set. For $j \in \{1,2,3\} \setminus \{i\}$, the vertex x can have at most $r-1$ neighbours in V_j, otherwise G would contain an induced $K_{1,r}$. It follows that $\deg(x) \leq 2(r-1)$ for all $x \in V(G)$.

Let $S' = S$. Let $F' = V(G) \setminus S'$, so $G[F']$ is a forest. To prove the lemma, we will iteratively modify S' until we obtain an independent feedback vertex set S' of G with $|S'| \leq (2r^2 - 5r + 3)|S|$. Every vertex $u \in S'$ has at most $2r-2$ neighbours in F'. Consider two neighbours v, w of u in F'. As F' is a forest, there is at most one induced path in F' from v to w, so there is at most one induced cycle in $G[F' \cup \{u\}]$ that contains all of u, v and w. Therefore $G[F' \cup \{u\}]$ contains at most $\binom{2r-2}{2} = \frac{1}{2}(2r-2)(2r-2-1) = 2r^2 - 5r + 3$ induced cycles. Note that every cycle in G contains at least one vertex of V_1. Therefore, if $s \in S' \cap (V_2 \cup V_3)$, then we can find a set X of at most $2r^2 - 5r + 3$ vertices in $V_1 \setminus S'$ such that if we replace s in S' by the vertices of X, then we again obtain a feedback vertex set. Repeating this process iteratively, for each vertex we remove from $S' \cap (V_2 \cup V_3)$, we add at most $2r^2 - 5r + 3$ vertices to $S' \cap V_1$. We stop the procedure once $S' \cap (V_2 \cup V_3)$ becomes empty, at which point we have produced a feedback vertex set S' with $|S'| \leq (2r^2 - 5r + 3)|S|$. Furthermore, at this point $S' \subseteq V_1$, so S' is independent. It follows that $\text{ifvs}(G) \leq (2r^2 - 5r + 3)\,\text{fvs}(G)$. ◀

Note that all near-bipartite graphs are 3-colourable (use one colour for the independent set and the two other colours for the forest). We prove the following lemma.

▶ **Lemma 9.** *Let $k \geq 3$. The class of C_k-free near-bipartite graphs is* ifvs*-unbounded and* ioct*-unbounded.*

Proof. For $r, s \geq 2$, let S_s^r denote the graph constructed as follows (see also Figure 2). Start with the graph that is the disjoint union of $2s$ copies of P_{2r}, and label these copies $U^1, \ldots, U^s, V^1, \ldots, V^s$. Add a vertex u adjacent to both endpoints of every U^i and a vertex v adjacent to both endpoints of every V^i. Finally, add an edge between u and v.

Every induced cycle in S_s^r is isomorphic to C_{2r+1}, which is an odd cycle. Thus a set $S \subseteq V(S_s^r)$ is a feedback vertex set for S_s^r if and only if it is an odd cycle transversal for S_s^r. It follows that $\text{fvs}(S_s^r) = \text{oct}(S_s^r)$ and $\text{ifvs}(S_s^r) = \text{ioct}(S_s^r)$.

Now $\{u, v\}$ is a minimum feedback vertex set of S_s^r, so $\text{fvs}(S_s^r) = \text{oct}(S_s^r) = 2$. However, any independent feedback vertex set S contains at most one vertex of u and v; say it does not contain u. Then it must contain at least one vertex of each U^i. It follows that $\text{ifvs}(S_s^r) = \text{ioct}(S_s^r) \geq s + 1$. Since for every $s \geq 2$, $k \geq 3$, the graph S_s^k is C_k-free. This completes the proof. ◀

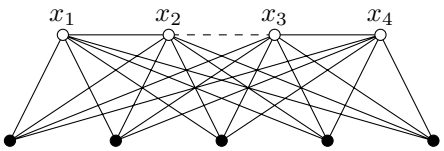


Figure 3 The graphs T_5 and T_5'. The edge x_2x_3 is present in T_5, but not in T_5'.

▶ **Theorem 2 (restated).** *Let H be a graph. The class of H-free near-bipartite graphs is* ifvs*-bounded if and only if H is isomorphic to $P_1 + P_2$, a star or an edgeless graph.*

Proof. If $H = P_1 + P_2$, then the theorem holds by Lemma 7. If H is isomorphic to a star or an edgeless graph, then H is an induced subgraph of $K_{1,r}$ for some $r \geq 1$. In this case the theorem holds by Lemma 8.

Now suppose that the class of H-free near-bipartite graphs is ifvs-bounded. By Lemma 9, H must be a forest. We will show that H is isomorphic to $P_1 + P_2$, a star or an edgeless graph.

We start by showing that H must be $(P_1+P_3, 2P_1+P_2, 2P_2)$-free. Let vertices x_1, x_2, x_3, x_4, in that order, form a path on four vertices. For $s \geq 3$, let T_s be the graph obtained from this path by adding an independent set I on s vertices (see also Figure 3) that is complete to the path and note that T_s is near-bipartite. Then $\{x_1, x_2, x_3\}$ is a minimum feedback vertex set in T_s. However, if S is an independent feedback vertex set, then S contains at most two vertices in $\{x_1, \ldots, x_4\}$. Therefore S must contain at least $s-1$ vertices of I, otherwise $T_s - S$ would contain an induced C_3 or C_4. Therefore $\mathrm{fvs}(T_s) = 3$ and $\mathrm{ifvs}(T_s) \geq s-1$. Note that T_s is $(P_1 + P_3, 2P_1 + P_2, 2P_2)$-free (this is easy to see by casting to the complement and observing that $\overline{T_s}$ is the disjoint union of a P_4 and a complete graph). Therefore H cannot contain $P_1 + P_3$, $2P_1 + P_2$ or $2P_2$ as an induced subgraph, otherwise T_s would be H-free, a contradiction.

Next, we show that H must be P_4-free. For $s \geq 3$ let T_s' be the graph obtained from T_s by removing the edge x_2x_3 (see also Figure 3). Then $\{x_1, x_2, x_3\}$ is a minimum feedback vertex set in T_s', so $\mathrm{fvs}(T_s') = 3$. By the same argument as for T_s, we find that $\mathrm{ifvs}(T_s') \geq s-1$ Now the complement $\overline{T_s'}$ is the disjoint union of a C_4 and a complete graph, so T_s' is P_4-free. Therefore H cannot contain P_4 as an induced subgraph.

We may now assume that H is a $(P_1+P_3, 2P_1+P_2, 2P_2, P_4)$-free forest. If H is connected, then it is a P_4-free tree, so it is a star, in which case we are done. We may therefore assume that H is disconnected. We may also assume that H contains at least one edge, otherwise we are done. Since H is $(2P_1 + P_2)$-free, it cannot have more than two components. Since H is $2P_2$-free, one of its two components must be isomorphic to P_1. Since H is a $(P_1 + P_3)$-free forest, its other component must be isomorphic to P_2. Hence H is isomorphic to $P_1 + P_2$. This completes the proof. ◀

4 Odd Cycle Transversal

Recall that a graph has an independent odd cycle transversal if and only if it is 3-colourable. We show the following two lemmas.

▶ **Lemma 10.** *If G is a P_4-free* 3*-colourable graph, then* $\mathrm{ioct}(G) = \mathrm{oct}(G)$.

Proof. Let G be a P_4-free 3-colourable graph. It suffices to prove the lemma component-wise, so we may assume that G is connected. Note that G cannot contain any induced odd cycles on more than three vertices, as it is P_4-free. Let (V_1, V_2, V_3) be a partition of $V(G)$ into

independent sets. We may assume that G is not bipartite, otherwise $\mathrm{ioct}(G) = \mathrm{oct}(G) = 0$, in which case we are done. As G is connected, P_4-free and contains more than one vertex, its complement $\overline{G}$ must be disconnected. Therefore we can partition the vertex set of G into two parts X_1 and X_2 such that X_1 is complete to X_2. No independent set V_i can have vertices in both X_1 and X_2, so without loss of generality we may assume that $X_1 = V_1$ and $X_2 = V_2 \cup V_3$. Since $G[X_2]$ is a P_4-free bipartite graph, it is readily seen that it is a disjoint union of complete bipartite graphs.

Note that $G - X_1$ is a bipartite graph, so X_1 is an odd cycle transversal of G. Furthermore, X_1 is independent. Now let S be a minimum vertex cover of $G[X_2]$. Observe that $G - S$ is bipartite, so S is an odd cycle transversal of G. Since $G[X_2]$ is the disjoint union of complete bipartite graphs, for every component C of $G[X_2]$, S must contain one part of the bipartition of C, or the other; by minimality of S, it only contains vertices from one of the parts. It follows that S is independent.

We now claim that every minimum odd cycle transversal S of G contains either X_1 or a minimum vertex cover of $G[X_2]$, both of which we have shown are independent odd cycle transversals; by the minimality of S, this will imply that S is equal to one of them. Indeed, suppose for contradiction that there is a vertex $x \in X_1 \setminus S$ and two adjacent vertices $y, z \in X_2 \setminus S$. Then $G[\{x, y, z\}]$ is a C_3 in $G - S$. This contradiction completes the proof. ◀

▶ **Lemma 11.** *If G is a $K_{1,3}$-free 3-colourable graph, then* $\mathrm{ioct}(G) \leq 3\,\mathrm{oct}(G)$.

Proof. Fix an integer $k \geq 0$. Let G be a $K_{1,3}$-free 3-colourable graph with an odd cycle transversal S such that $|S| = k$. Fix a partition of $V(G)$ into three independent sets V_1, V_2, V_3. Without loss of generality assume that $|S \cap V_1| \geq |S \cap V_2|, |S \cap V_3|$, so $|S \cap (V_2 \cup V_3)| \leq \frac{2k}{3}$. Let $S' = S$ and note that $G - S'$ is bipartite by definition of odd cycle transversal. To prove the lemma, we will iteratively modify S' until we obtain an independent odd cycle transversal S' of G with $|S'| \leq 3k$.

Suppose $x \in V_i$ for some $i \in \{1, 2, 3\}$. Then x has no neighbours in V_i since V_i is an independent set. For $j \in \{1, 2, 3\} \setminus \{i\}$, the vertex x can have at most two neighbours in V_j, otherwise G would contain an induced $K_{1,3}$. It follows that $\deg(x) \leq 4$ for all $x \in V(G)$.

As $G - S'$ is a bipartite $K_{1,3}$-free graph, it is a disjoint union of paths and even cycles. Every vertex $u \in S'$ has at most four neighbours in $V(G) \setminus S'$. An induced odd cycle in $G - (S' \setminus \{u\})$ consists of the vertex u and an induced path P in $G - S'$ between two neighbours v, w of u such that $P \cap N(u)$ does not contain any vertices apart from v and w. If u has q neighbours in some component C of $G - S'$, then there can be at most q such paths P that lie in this component. It follows that there are at most four induced odd cycles in $G - (S' \setminus \{u\})$. Note that every induced odd cycle in G contains at least one vertex in each V_i. Therefore, if $s \in S' \cap (V_2 \cup V_3)$, then we can find a set X of at most four vertices in $V_1 \setminus S'$ such that if we replace s in S' by the vertices of X, then we again obtain an odd cycle transversal. Repeating this process iteratively, for each vertex we remove from $S' \cap (V_2 \cup V_3)$, we add at most four vertices to $S' \cap V_1$, so $|S'|$ increases by at most 3. We stop the procedure once $S' \cap (V_2 \cup V_3)$ becomes empty, at which point we have produced an odd cycle transversal S' with $|S'| \leq |S| + 3|S \cap (V_2 \cup V_3)| \leq k + 2 \times \frac{2k}{3} = 3k$. Furthermore, at this point $S' \subseteq V_1$, so S' is independent. It follows that $\mathrm{ioct}(G) \leq 3\,\mathrm{oct}(G)$. ◀

▶ **Lemma 12.** *Let $r, s \geq 1$. Suppose there is a function $f : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ such that* $\mathrm{ioct}(G) \leq f(\mathrm{oct}(G))$ *for every $K_{1,r}$-free 3-colourable graph G. Then* $\mathrm{ioct}(G) \leq \max(\mathrm{oct}(G)r + r^2 + 3rs - 2r, f(\mathrm{oct}(G)))$ *for every $(K_{1,r} + sP_1)$-free 3-colourable graph G.*

Proof. Fix $r, s \geq 1$ and $k \geq 0$. Let G be a $(K_{1,r} + sP_1)$-free 3-colourable graph with a minimum odd-cycle transversal T on k vertices. Fix a partition of $V(G)$ into three independent sets V_1, V_2, V_3. We may assume that $\text{oct}(G) \geq 2$, otherwise $\text{ioct}(G) = \text{oct}(G)$ and we are done. If $|V_i| \leq \max(\text{oct}(G)r + r^2 + 3rs - 2r, f(\text{oct}(G)))$ for some $i \in \{1, 2, 3\}$, then deleting V_i from G yields a bipartite graph, so $\text{ioct}(G) \leq \max(\text{oct}(G)r + r^2 + 3rs - 2r, f(\text{oct}(G)))$ and we are done. We may therefore assume that $|V_i| > \max(\text{oct}(G)r + r^2 + 3rs - 2r, f(\text{oct}(G)))$ for all $i \in \{1, 2, 3\}$. If G is $K_{1,r}$-free, then $\text{ioct}(G) \leq f(\text{oct}(G))$, so suppose that G contains an induced $K_{1,r}$, say with vertex set X. Note that $|X| = r + 1$, and each V_i can contain at most r vertices of X, since every V_i is an independent set.

For every $i \in \{1, 2, 3\}$, there cannot be a set of s vertices in $V_i \setminus X$ that are anti-complete to X, otherwise G would contain an induced $K_{1,r} + sP_1$, a contradiction. For every $i \in \{1, 2, 3\}$, since $|V_i| > \text{oct}(G)r + r^2 + 3rs - 2r \geq r^2 + 3rs$, it follows that $|V_i \setminus X| \geq |V_i| - r > (s-1) + (r+1)(r-1) = (s-1) + |X|(r-1)$. Hence for every $i \in \{1, 2, 3\}$, there must be a vertex $x \in X$ that has at least r neighbours in V_i. Applying this for each i in turn, we find that at least two of the graphs in $\{G[V_1 \cup V_2], G[V_1 \cup V_3], G[V_2 \cup V_3]\}$ contain a vertex of degree at least r; without loss of generality assume that this is the case for $G[V_1 \cup V_2]$ and $G[V_1 \cup V_3]$. Let V_2' and V_3' denote the set of vertices in V_2 and V_3, respectively, that have more than $s-1$ non-neighbours in V_1. By Lemma 4, $|V_2'|, |V_3'| \leq s - 1$.

Suppose a vertex $x \in V_2 \setminus V_2'$ is adjacent to a vertex $y \in V_3 \setminus V_3'$. By definition of V_2' and V_3', the vertices x and y each have at most $s-1$ non-neighbours in V_1. Since $|V_1| - 2(s-1) \geq \text{oct}(G)+1$, it follows that $|N(x) \cap N(y) \cap V_1| \geq \text{oct}(G)+1$ so $N(x) \cap N(y) \cap V_1 \not\subseteq T$. We conclude that at least one of x or y must be in T. In other words, $T \cap ((V_2 \setminus V_2') \cup (V_3 \setminus V_3'))$ is a vertex cover of $G[(V_2 \setminus V_2') \cup (V_3 \setminus V_3')]$, of size at most $\text{oct}(G)$. Therefore $(T \cap ((V_2 \setminus V_2') \cup (V_3 \setminus V_3'))) \cup V_2' \cup V_3'$ is a vertex cover of $G[V_2 \cup V_3]$ of size at most $\text{oct}(G)+2(s-1)$. By Lemma 5, there is an independent vertex cover T' of $G[V_2 \cup V_3]$ of size at most $(\text{oct}(G)+2(s-1))r + rs = \text{oct}(G)r + 3rs - 2r$. Note that by definition of vertex cover, $(V_2 \cup V_3) \setminus T'$ is an independent set, and so $G - T'$ is bipartite. Therefore T' is an independent odd cycle transversal for G of size at most $\text{oct}(G)r + 3rs - 2r$. This completes the proof. ◀

The following result follows immediately from combining Lemmas 11 and 12.

▶ **Corollary 13.** *For $s \geq 1$, $\text{ioct}(G) \leq 3\,\text{oct}(G) + 9s + 3$ for every $(K_{1,3} + sP_1)$-free 3-colourable graph G.*

▶ **Lemma 14.** *The class of $(P_1 + P_4, 2P_2)$-free 3-colourable graphs is* ioct*-unbounded.*

Proof. Let $s \geq 2$. We construct the graph Q_s as follows (see also Figure 4). First, let A, B and C be disjoint independent sets of s vertices. Choose vertices $a \in A$, $b \in B$ and $c \in C$. Add edges so that a is complete to $B \cup C$, b is complete to $A \cup C$ and c is complete to $A \cup B$. Let Q_s be the resulting graph and note that it is 3-colourable with colour classes A, B and C.

Note that $\{a, b\}$ is a minimum odd cycle transversal of Q_s, so $\text{oct}(Q_s) = 2$.

Let S be a minimum independent odd cycle transversal. Then S contains at most one vertex in $\{a, b, c\}$, say S contains neither b nor c. If a vertex $x \in A$ is not in S, then $Q_s[\{x, b, c\}]$ is a C_3 in $Q_s - S$, a contradiction. Hence every vertex of A is in S, and so $\text{ioct}(Q_s) \geq s$.

It remains to show that Q_s is $(P_1 + P_4, 2P_2)$-free. Consider a vertex $x \in A$. Then $Q_s - N[x]$ is an edgeless graph if $x = a$ and $Q_s - N[x]$ is the disjoint union of a star and an edgeless graph otherwise. It follows that $Q_s - N[x]$ is P_4-free. By symmetry, we conclude that Q_s is $(P_1 + P_4)$-free. Now consider a vertex $y \in N(a) \cap B$. Then $Q_s - N[\{a, y\}]$ is empty if $b = y$ and $Q_s - N[\{a, y\}]$ is an edgeless graph otherwise. It follows that $Q_s - N[\{a, y\}]$ is P_2-free. By symmetry, we conclude that Q_s is $2P_2$-free. This completes the proof. ◀

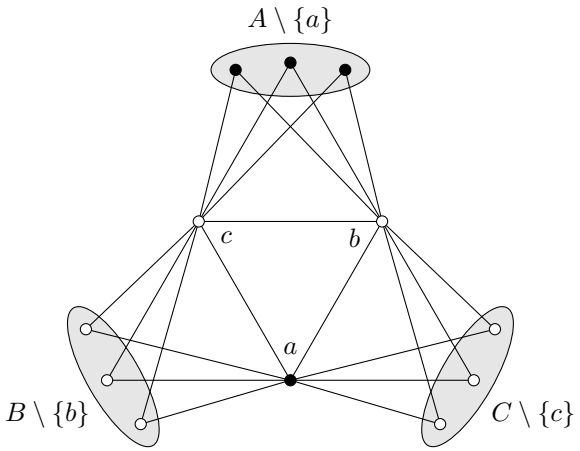


Figure 4 The graph Q_4.

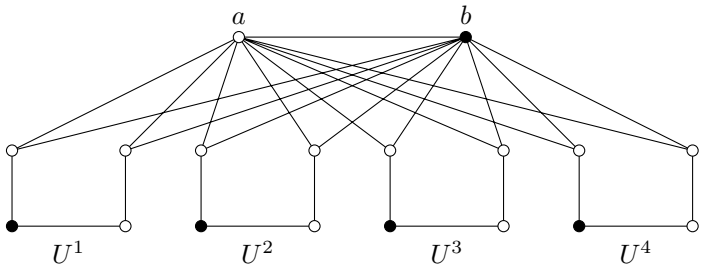


Figure 5 The graph Z_4.

▶ **Lemma 15.** *Let H be a graph with more than one vertex of degree at least 3. Then the class of H-free 3-colourable graphs is* ioct*-unbounded.*

Proof. Let $s \geq 1$. We construct the graph Z_s as follows (see also Figure 5). Start with the disjoint union of s copies of P_4 and label these copies $U^1, \ldots, U^s$. Add an edge ab and make a and b adjacent to the endpoints of every U^i. Let Z_s be the resulting graph and note that Z_s is 3-colourable (colour a and b with Colours 1 and 2, respectively, colour the endpoints of the U^is with Colour 3 and colour the remaining vertices of the U^is with Colours 1 and 2).

Note that $Z_s - \{a, b\}$ is bipartite, so $\{a, b\}$ is a minimum odd cycle transversal and $\mathrm{oct}(Z_s) = 2$. However, any independent odd cycle transversal S contains at most one vertex of a and b; say it does not contain a. For every $i \in \{1, \ldots, s\}$, the graph $Z_s[U^i \cup \{a\}]$ is a C_5. Therefore S must contain at least one vertex from each U^i. It follows that $\mathrm{ioct}(Z_s) \geq s$.

Let H be a graph with more than one vertex of degree at least 3. By Lemma 9, we may assume that H is a forest. It remains to show that Z_s is H-free. Suppose, for contradiction, that Z_s contains H as an induced subgraph and let x and y be two vertices that have degree at least 3 in H. Since H is a forest, x and y must each have three pairwise non-adjacent neighbours in Z_s. The endpoints of each U^i have exactly three neighbours, but two of them (a and b) are adjacent. Without loss of generality we may therefore assume that $x = a$ and $y = b$. Since x has degree at least 3 in H, the vertex x must have a neighbour $z \neq y$ in H and so z must be the endpoint of a U^i. Therefore x, y and z are pairwise adjacent, so $H[\{x, y, z\}]$ is a C_3, contradicting the fact that H is a forest. It follows that Z_s is H-free. This completes the proof. ◀

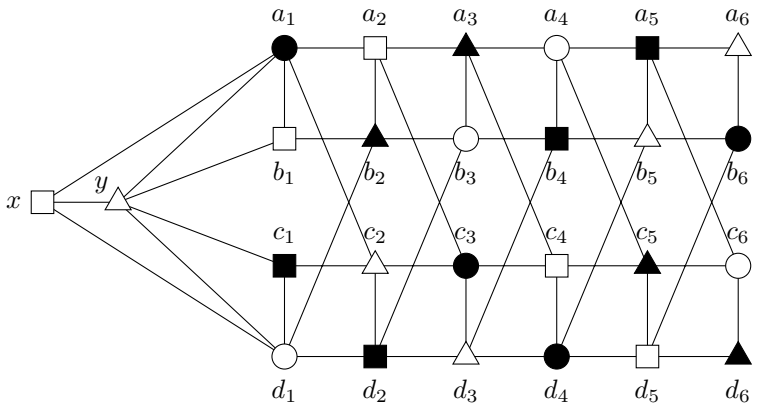


Figure 6 The graph Y_2. Different shapes show the unique 3-colouring of Y_2. Different colours show the 2-colouring of $Y_2 - \{x, y\}$.

▶ **Lemma 16.** *The class of $K_{1,5}$-free 3-colourable graphs is* ioct*-unbounded.*

Proof. Let $s \geq 1$. We construct the graph Y_s as follows (see also Figure 6).

1. Start with the disjoint union of four copies of P_{3s} and label the vertices of these paths $a_1, \ldots, a_{3s}$, $b_1, \ldots, b_{3s}$, $c_1, \ldots, c_{3s}$ and $d_1, \ldots, d_{3s}$ in order, respectively.
2. For each $i \in \{1, \ldots, 3s\}$ add the edges a_ib_i and c_id_i.
3. For each $i \in \{1, \ldots, 3s-1\}$ add the edges a_ic_{i+1} and d_ib_{i+1}.
4. Finally, add an edge xy and make x adjacent to a_1 and d_1 and y adjacent to a_1, b_1, c_1 and d_1.

Let Y_s be the resulting graph.

First note that Y_s is $K_{1,5}$-free. The vertices y, a_1 and d_1 all have degree 5, but their neighbourhood is not independent, so they cannot be the central vertex of an induced $K_{1,5}$. All the other vertices have degree at most 4, so they cannot be the central vertex of an induced $K_{1,5}$ either. Therefore no vertex in Y_s is the central vertex of an induced $K_{1,5}$, so Y_s is $K_{1,5}$-free.

The graph $Y_s - \{x, y\}$ is bipartite with bipartition classes:

1. $\{a_i, c_i \mid 1 \leq i \leq 3s, i \equiv 1 \bmod 2\} \cup \{b_i, d_i \mid 1 \leq i \leq 3s, i \equiv 0 \bmod 2\}$ and
2. $\{a_i, c_i \mid 1 \leq i \leq 3s, i \equiv 0 \bmod 2\} \cup \{b_i, d_i \mid 1 \leq i \leq 3s, i \equiv 1 \bmod 2\}$.

It follows that $\mathrm{oct}(Y_s) = 2$.

Furthermore, Y_s is 3-colourable with colour classes:

1. $\{x\} \cup \{a_i, d_i \mid 1 \leq i \leq 3s, i \equiv 2 \bmod 3\} \cup \{b_i, c_i \mid 1 \leq i \leq 3s, i \equiv 1 \bmod 3\}$
2. $\{y\} \cup \{a_i, d_i \mid 1 \leq i \leq 3s, i \equiv 0 \bmod 3\} \cup \{b_i, c_i \mid 1 \leq i \leq 3s, i \equiv 2 \bmod 3\}$
3. $\{a_i, d_i \mid 1 \leq i \leq 3s, i \equiv 1 \bmod 3\} \cup \{b_i, c_i \mid 1 \leq i \leq 3s, i \equiv 0 \bmod 3\}$

In fact, we will show that this 3-colouring is unique (up to permuting the colours). To see this, suppose that $c : V(Y_s) \to \{1, 2, 3\}$ is a 3-colouring of Y_s. Since x and y are adjacent we may assume without loss of generality that $c(x) = 1$ and $c(y) = 2$. Since a_1 and d_1 are adjacent to both x and y, it follows that $c(a_1) = c(d_1) = 3$. Since b_1 is adjacent to y and a_1, it follows that $c(b_1) = 1$. By symmetry $c(c_1) = 1$.

We prove by induction on i that for every $i \in \{1, \ldots, 3s\}$, $c(a_i) = c(d_i) \equiv i + 2 \bmod 3$ and $c(b_i) = c(c_i) \equiv i \bmod 3$. We have shown that this is true for $i = 1$. Suppose that the claim holds for $i - 1$ for some $i \in \{2, \ldots, 3s\}$. Then $c(a_{i-1}) = c(d_{i-1}) \equiv (i-1) + 2 \bmod 3$ and $c(b_{i-1}) = c(c_{i-1}) \equiv i - 1 \bmod 3$. Since b_i is adjacent to b_{i-1} and d_{i-1}, it follows that $c(b_i) \equiv i \bmod 3$. Since a_i is adjacent to b_i and a_{i-1}, it follows that $c(a_i) \equiv i + 2 \bmod 3$.

By symmetry $c(c_i) \equiv i \bmod 3$ and $c(d_i) \equiv i+2 \bmod 3$. Therefore the claim holds for i. By induction, this completes the proof of the claim and therefore shows that the 3-colouring of Y_s is indeed unique.

Furthermore, note that the colour classes in this colouring have sizes $4s+1$, $4s+1$ and $4s$, respectively. A set S is an independent odd cycle transversal of a graph if and only if it is a colour class in some 3-colouring of this graph. It follows that $\mathrm{ioct}(Y_s) = 4s$. This completes the proof. ◀

We summarise the results of this section in the following theorem.

▶ **Theorem 3 (restated).** *Let H be a graph. The class of H-free* 3*-colourable graphs is* ioct*-bounded:*

- *if H is an induced subgraph of P_4 or $K_{1,3}+sP_1$ for some $s \geq 0$ and*
- *only if H is an induced subgraph of $K_{1,4}^+$ or $K_{1,4}+sP_1$ for some $s \geq 0$.*

Proof. If H is an induced subgraph of P_4 or $K_{1,3}+sP_1$ for some $s \geq 0$, then the class of H-free 3-colourable graphs is ioct-bounded by Lemma 10 or Corollary 13, respectively.

Now suppose that the class of H-free 3-colourable graphs is ioct-bounded. By Lemma 9, H must be a forest. By Lemma 16, H must be $K_{1,5}$-free. Since H is a $K_{1,5}$-free forest, it has maximum degree at most 4. By Lemma 14, H must be $(P_1+P_4, 2P_2)$-free.

First suppose that H is P_4-free, so every component of H is a P_4-free tree. Hence every component of H is a star. In fact, as H has maximum degree at most 4, every component of H is an induced subgraph of $K_{1,4}$. As H is $2P_2$-free, at most one component of H contains an edge. Therefore H is an induced subgraph of $K_{1,4}+sP_1$ for some $s \geq 0$ and we are done.

Now suppose that H contains an induced P_4, say on vertices x_1, x_2, x_3, x_4 in that order and let $X = \{x_1, \ldots, x_4\}$. Since H is a forest, every vertex $v \in V(H) \setminus X$ has at most one neighbour in X. A vertex $v \in V(H) \setminus X$ cannot be adjacent to x_1 or x_4, since H is $2P_2$-free. By Lemma 15, the vertices x_2 and x_3 cannot both have neighbours outside X; without loss of generality assume that x_3 has no neighbours in $V(H) \setminus X$. Since H is (P_1+P_4)-free, every vertex $v \in V(H) \setminus X$ must have at least one neighbour in X, so it must be adjacent to x_2. As H has maximum degree at most 4, it follows that H is an induced subgraph of $K_{1,4}^+$. This completes the proof. ◀

By Lemma 12, for $r, s \geq 1$ the class of $K_{1,r}$-free 3-colourable graphs is ioct-bounded if and only if the class of $(K_{1,r}+sP_1)$-free 3-colourable graphs is ioct-bounded. Therefore, Theorem 3 leaves three open cases, as follows:

▶ **Open Problem 17.** *Is the class of H-free* 3*-colourable graphs* ioct*-bounded when H is:*

1. $K_{1,4}$ *(or equivalently $K_{1,4}+sP_1$ for any $s \geq 1$).*
2. $K_{1,3}^+$ *or*
3. $K_{1,4}^+$.

5 Conclusions

To develop an insight into the price of independence for classical concepts, we have investigated whether or not the size of a minimum independent vertex cover, feedback vertex set or odd cycle transversal is bounded in terms of the minimum size of the not-necessarily-independent variant of each of these transversals for H-free graphs (that have such independent transversals). We have determined the answer for every graph H except for the three missing cases for odd cycle transversal listed in Open Problem 17. While we note that the bounds

we give in some of our results (Lemmas 7 and 10) are tight, in this paper we were mainly concerned with obtaining the dichotomy results on whether there is a bound, rather than trying to find exact bounds.

As results for the price of connectivity implied algorithmic consequences [9, 13], it is natural to ask if our results for the price of independence have similar consequences. The problems Independent Vertex Cover, Independent Feedback Vertex Set and Independent Odd Cycle Transversal ask to determine the minimum size of the corresponding independent transversal. The first problem is readily seen to be polynomial-time solvable. The other two problems are NP-hard for H-free graphs whenever H is not a linear forest [3], just like their classical counterparts Feedback Vertex Set [16, 17] and Odd Cycle Transversal [10] (see also [14, 15]). The complexity of these four problems restricted to H-free graphs is still poorly understood when H is a linear forest. Our results suggest that it is unlikely that we can obtain polynomial algorithms for the independent variants based on results for the original variants, as the difference between $\mathrm{ifvs}(G)$ and $\mathrm{fvs}(G)$ and between $\mathrm{ioct}(G)$ and $\mathrm{oct}(G)$ can become unbounded quickly.

References

1 Rémy Belmonte, Pim van 't Hof, Marcin Kamiński, and Daniël Paulusma. The price of connectivity for feedback vertex set. *Discrete Applied Mathematics*, 217(Part 2):132–143, 2017.

2 Marthe Bonamy, Konrad K. Dabrowski, Carl Feghali, Matthew Johnson, and Daniël Paulusma. Recognizing graphs close to bipartite graphs. *Proc. MFCS 2017, LIPIcs*, 83:70:1–70:14, 2017.

3 Marthe Bonamy, Konrad K. Dabrowski, Carl Feghali, Matthew Johnson, and Daniël Paulusma. Independent feedback vertex set for P_5-free graphs. *Algorithmica*, to appear.

4 Eglantine Camby. Price of independence for the dominating set problem. Manuscript, 2017.

5 Eglantine Camby, Jean Cardinal, Samuel Fiorini, and Oliver Schaudt. The price of connectivity for vertex cover. *Discrete Mathematics & Theoretical Computer Science*, 16:207–224, 2014.

6 Eglantine Camby and Fränk Plein. A note on an induced subgraph characterization of domination perfect graphs. *Discrete Applied Mathematics*, 217(Part 3):711–717, 2017.

7 Eglantine Camby and Oliver Schaudt. The price of connectivity for dominating set: Upper bounds and complexity. *Discrete Applied Mathematics*, 177:53–59, 2014.

8 Jean Cardinal and Eythan Levy. Connected vertex covers in dense graphs. *Theoretical Computer Science*, 411(26–28):2581–2590, 2010.

9 Nina Chiarelli, Tatiana R. Hartinger, Matthew Johnson, Martin Milanič, and Daniël Paulusma. Minimum connected transversals in graphs: New hardness results and tractable cases using the price of connectivity. *Theoretical Computer Science*, 705:75–83, 2018.

10 Thomas Emden-Weinert, Stefan Hougardy, and Bernd Kreuter. Uniquely colourable graphs and the hardness of colouring graphs of large girth. *Combinatorics, Probability and Computing*, 7(04):375–386, 1998.

11 Alexander Grigoriev and René Sitters. Connected feedback vertex set in planar graphs. *Proc. WG 2009, LNCS*, 5911:143–153, 2010.

12 Tatiana R. Hartinger, Matthew Johnson, Martin Milanič, and Daniël Paulusma. The price of connectivity for cycle transversals. *European Journal of Combinatorics*, 58:203–224, 2016.

13 Matthew Johnson, Giacomo Paesani, and Daniël Paulusma. Connected vertex cover for $(sP_1 + P_5)$-free graphs. *Proc. WG 2018, LNCS*, to appear.

14 Daniel Král', Jan Kratochvíl, Zsolt Tuza, and Gerhard J. Woeginger. Complexity of coloring graphs without forbidden induced subgraphs. *Proc. WG 2001, LNCS*, 2204:254–262, 2001.

15 Vadim V. Lozin and Marcin Kamiński. Coloring edges and vertices of graphs without short or long cycles. *Contributions to Discrete Mathematics*, 2(1), 2007.

16 Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, and Saket Saurabh. On parameterized independent feedback vertex set. *Theoretical Computer Science*, 461:65–75, 2012.

17 Andrea Munaro. On line graphs of subcubic triangle-free graphs. *Discrete Mathematics*, 340(6):1210–1226, 2017.

Probabilistic Secret Sharing

Paolo D'Arco
Dipartimento di Informatica, Università of Salerno, Italy
pdarco@unisa.it
https://orcid.org/0000-0002-9271-4240

Roberto De Prisco
Dipartimento di Informatica, Università of Salerno, Italy
robdep@unisa.it
https://orcid.org/0000-0003-0559-6897

Alfredo De Santis
Dipartimento di Informatica, Università of Salerno, Italy
ads@unisa.it
https://orcid.org/0000-0001-8962-1919

Angel Pérez del Pozo
Departamento de Matemática Aplicada, Ciencia e Ingeniería de los Materiales y Tecnología Electrónica, Universidad Rey Juan Carlos, Madrid, Spain
angel.perez@urjc.es
https://orcid.org/0000-0002-8135-9642

Ugo Vaccaro
Dipartimento di Informatica, Università of Salerno, Italy
uvaccaro@unisa.it
https://orcid.org/0000-0003-4085-7300

Abstract

In classical secret sharing schemes a dealer shares a secret among a set of participants in such a way that qualified subsets can reconstruct the secret, while forbidden ones do not get any kind of information about it. The basic parameter to optimize is the size of the shares, that is, the amount of secret information that the dealer has to give to participants. In this paper we formalize a notion of *probabilistic* secret sharing schemes, in which qualified subsets can reconstruct the secret but only with a certain controlled probability. We show that, by allowing a bounded error in the reconstruction of the secret, it is possible to *drastically reduce the size* of the shares the participants get (with respect to classical secret sharing schemes). We provide efficient constructions both for threshold access structures on a finite set of participants and for evolving threshold access structures, where the set of participants is potentially infinite. Some of our constructions yield shares of *constant* size (i.e., not depending on the number of participants) and an error probability of successfully reconstructing the secret which can be made as *close to* 1 as desired.

2012 ACM Subject Classification Security and privacy → Mathematical foundations of cryptography

Keywords and phrases Secret sharing, probabilistic secret sharing, evolving secret sharing

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.64

Acknowledgements The first and fourth authors are partially supported by research project MTM2016-77213-R funded by the Spanish MINECO.

1 Introduction

Secret sharing. A secret sharing scheme is a method through which a *dealer* shares a piece of information (*a secret*) among a set of participants, according to a specific privacy policy. Specifically, each participant, during the sharing phase, receives and securely stores a piece of information, called *share*. Then, in the reconstruction phase, prescribed subsets, called *qualified subsets*, by pooling together their shares or by sending them to a trusted combiner, can recover the secret through the computation of an appropriate reconstruction function. All the other remaining subsets of participants, called *forbidden subsets*, analyzing the shares they have, and applying *any* computation on them, do not get any information whatsoever about the secret. The collection of qualified subsets forms the *access structure* to the secret.

The notion of secret sharing and the first constructions were introduced in 1979, independently by Shamir [34] and Blakley [6]. Both of them considered *threshold* access structures, usually referred to as (k, n)-threshold schemes, where the set of participants has size n, *any* subset of size greater than or equal to k is qualified, and *any* subset of size less than k is forbidden. General access structures were later considered in [22, 23, 23, 4, 35, 25].

Secret sharing schemes have been widely used in cryptographic protocol design and have been extended in many ways in order to exhibit additional properties. For a detailed overview of the field and of the open problems the reader is referred to [2] and to [14].

Visual cryptography (visual secret sharing). Visual cryptography, introduced independently and in different forms in [31, 24], is a sharing technique in which the secret is an image, the shares are images printed on transparencies and the reconstruction is performed by the human visual system by looking at the superposed shares. Notice that, while the image can be seen as a sequence of black and white pixels and thus can encode arbitrary sequences of bits, the reconstruction function is fixed to be the `or` of the shares. Hence, any visual cryptography scheme can be used as a regular secret sharing scheme, i.e., as a scheme that uses the `or` function to reconstruct the secret; on the other hand, the converse is not true, unless the scheme uses the `or` function on the corresponding bits of the shares to reconstruct the secret. In the context of visual cryptography, the notion of probabilistic reconstruction has been widely studied, e.g., [36, 13, 19], since the erroneous reconstruction of a limited number of pixels does not significantly affect the recognizability of the reconstructed secret image.

Evolving access structures. An interesting new variant of secret sharing schemes has been introduced recently in [26, 27]. The authors have considered a setting in which the set of participants is *infinite* and the access structure is defined through a *collection* of access structures, not known at the beginning. More precisely, at any time t, a new participant arrives and new qualified subsets – if any – are added to the existing access structure $\mathcal{A}_{t-1}$, obtaining the new one $\mathcal{A}_t$. The authors of [26, 27] design secret sharing schemes in such a new scenario for threshold access structures, denoted as (k, ∞)-threshold schemes, and for general access structures. The authors of [26, 27] have also shown a nice equivalence between $(2, \infty)$-threshold schemes and prefix-free codes for the integers. Further results have been provided in [32, 28].

Motivations of the present work. Since their introduction, secret sharing schemes have represented a relevant research area in Cryptography. Secret sharing schemes are an important tool by themselves, as well as a building block both in the general solutions for multi-party

computation and in the design of ad-hoc protocols. A central issue in the area has been to provide constructive results and non-existential results towards the possibility of obtaining secret sharing schemes that provide to the users shares *short in size*. For threshold access structures and ramp schemes tight bounds have been provided in [11] and in [9].

Unfortunately, for general access structures what can be done is still an open problem: for many access structures the known constructions deliver to the users shares of exponential size in the number n of participants [5] (see also [30] for recent important improvements). The best lower bound on the size of the shares is instead sublinear as it was proved in [8, 15] using the information theoretic approach introduced in [10]. For the subclass of *linear* general secret sharing schemes recent results of [33] provide exponential lower bounds.

Several papers have considered secret sharing schemes in which the privacy condition has been relaxed, both in a *statistical* sense, i.e., the privacy is not information-theoretic but there is a probability of information leakage, and in a *computational* sense, i.e., the privacy is only guaranteed with respect to computationally bounded adversaries, e.g., [29]. In [16, 20] secret sharing schemes are seen as joint probability distributions of the shares and the secret. This particular view is introduced to study the case of infinite participants and to deal with infinite domains for the shares. The approach leads to a probabilistic notion of security allowing information leakage. So, it is quite different from the approach pursued in this paper. To our knowledge, although the relaxation of the property that qualified subsets can correctly recover the secret has been mentioned in a few papers, no study has focused on the analysis and the design of secret sharing schemes in which the secret can be reconstructed only with a prescribed high probability. Exceptions consist for the specific case of visual cryptography [17] and, notably, the secret sharing scheme of [21]. We remark that the model used in [21], that allows public information, is different from the one considered in this paper.

Hence, it seems natural to ask the following question:

> *Can we reduce the size of the shares held by the participants if we allow a small probability of error in the reconstruction phase?*

In this paper we show that it is possible to give an answer to the above question by providing, among other results, a quantitative trade-off between the probability of a correct reconstruction and the size of the shares.

Our contribution. We proceed as follows: in Section 2, we introduce a formal definition of probabilistic secret sharing schemes that intends to capture the intuition given above. We consider both access structures on a finite set of participants and evolving access structures defined over an infinite set of participants. Then, in Section 3, we discuss probabilistic constructions for the finite case and, in Section 4, we propose a probabilistic construction for a $(2, \infty)$-threshold secret sharing scheme. In Section 5, we describe and analyze some methods to build more general schemes from simpler ones. Our techniques are of general interest since they apply to both deterministic and probabilistic secret sharing schemes. Then, in Section 6, we describe and analyze a direct construction for probabilistic secret sharing schemes, which gives to participants shares of constant size, with respect to the number of participants, and enables the dealer to set the probability of reconstruction as high as he needs. Finally, in Section 7, we provide conclusions and briefly discuss some open problems.

Overview of results and techniques. We define and provide constructions for α-probabilistic secret sharing schemes for threshold access structures, where α denotes *the probability of a correct reconstruction*, both for the finite and the infinite cases, by using a variety of

techniques. Throughout the paper, unless otherwise stated, we assume that the secret is a single bit.

- Focusing on the finite case, where the set of participants is fixed and has size n, we leverage on a nice connection in *visual* secret sharing schemes between deterministic and probabilistic schemes. We show how a probabilistic visual secret sharing scheme can be turned easily into an α-probabilistic secret sharing scheme for the same access structure.
- To deal with evolving access structures, we design a $(2, \infty)$-threshold $\frac{1+p}{2}$-probabilistic secret sharing scheme, where $(p, 1-p)$ is the probability distribution of the secret bit. Our construction is inspired by the $(2, \infty)$-threshold visual cryptography scheme provided in [12], which seems to be the first paper that has considered the problem of constructing a visual secret sharing scheme for an infinite set of participants. It is simple and allows to share a 1-bit secret with shares of 1-bit.
- Successively, in order to construct more general schemes from simpler ones, using a recursive approach, we present two algorithms: the first builds a $(k+1, \infty)$-threshold α-probabilistic secret sharing scheme from a (k, ∞)-threshold β-probabilistic secret sharing scheme. The second builds a $(k+1, \infty)$-threshold α-probabilistic secret sharing scheme from k schemes, i.e., by using, for $j = 2, \ldots, k$, a (j, ∞)-threshold α_j-probabilistic secret sharing scheme. We point out that the algorithms can be used to construct *both* probabilistic schemes and deterministic schemes for evolving access structures. In particular, for both of our methods, if we use the most efficient deterministic $(2, \infty)$-threshold secret sharing scheme provided in [26, 27] and apply to the scheme resulting from the algorithms, the same domain reduction technique which is used in [26, 27], we get deterministic schemes for evolving access structures which achieve the same asymptotic share size of the ones obtained in [26, 27]. Hence, the new schemes can be seen as an *alternative* way for constructing deterministic schemes. On the other hand, for both of our algorithms, starting from our $(2, \infty)$-threshold $\frac{1+p}{2}$-probabilistic secret sharing scheme, we obtain (k, ∞)-threshold α-probabilistic secret sharing schemes which achieve asymptotically the same share size of the deterministic schemes.
- Finally, through a direct construction, which uses the shares of Shamir's scheme, we provide a (k, ∞)-threshold α-probabilistic secret sharing scheme which enables to share a 1-bit secret with shares of *constant size*. Moreover, the scheme exhibits a nice trade-off between the probability α of successfully reconstructing the secret and the size of the underlying field: indeed, α can approach 1 as much as desired by properly choosing the size of $\mathbb{F}_q$. For the case in which k is equal to 2, we explain the differences between this construction and the $(2, \infty)$-threshold $\frac{1+p}{2}$-probabilistic secret sharing scheme presented in Section 4. Finally, we emphasize that our construction also applies to the case of a finite number of participants, and we describe advantages and disadvantages associated with it.

Hence, the probabilistic approach we suggest enables to *beat the lower bounds* on the size of the shares the deterministic threshold schemes are subject to, both for the case of a finite set of participants and for the infinite one.

Related work. Other models for secret sharing schemes have been introduced in the past years in order to reduce the size of the shares held by the participants. More precisely, secret sharing schemes in which the correctness property holds with *no error*, and the privacy property holds with *identical* probability distributions on the shares held by the participants of a forbidden subset, are called *perfect*. *Non-perfect* secret sharing schemes are less restrictive. For example, in a (d, t, n)-ramp scheme [7], the first and most relevant case of non-perfect

secret sharing schemes, forbidden subsets of size less than d do not get any information about the secret, qualified subsets of size greater than or equal to t reconstruct the secret, while subsets whose sizes are *"in between"* get *some* information about the secret. Note that ramp schemes are a *close but different* class of non-perfect schemes, compared to the probabilistic ones we introduce here. Similarly, *computational* secret sharing schemes [29] are another class of non-perfect secret sharing schemes; in such schemes the privacy property requires computationally indistinguishable probability distributions on the shares held by the participants of a forbidden subset. In both cases, it is possible to design schemes with shorter share size compared to the size of the shares in perfect secret sharing schemes.

2 The Models

In this section we introduce the models we work with. We essentially follow and extend the treatment of [2, 26, 27].

Let $\mathcal{P}_n = \{p_1, \ldots, p_n\}$ be a finite set of n participants, and let $2^{\mathcal{P}_n}$ be the set of all the subsets of $\mathcal{P}_n$. A collection of subsets $\mathcal{A} \subseteq 2^{\mathcal{P}_n}$ is *monotone* if, for each subsets $B, C \in \mathcal{P}_n$ such that $B \subseteq C$, the condition $B \in \mathcal{A}$ implies $C \in \mathcal{A}$.

An access structure is defined as follows:

▶ **Definition 1.** An *access structure* $\mathcal{A}$ on the set $\mathcal{P}_n = \{p_1, \ldots, p_n\}$ is a monotone collection of subsets of $\mathcal{P}_n$, i.e., $\mathcal{A} \subseteq 2^{\mathcal{P}_n}$. Subsets in $\mathcal{A}$ are called *qualified*. Subsets not in $\mathcal{A}$ are called *forbidden*.

To avoid overburdening the notation, when it is clear from the context, we use the letter which denotes a subset of participants also to denote the subset of the indices of the participants. We define a probabilistic secret sharing scheme as follows.

▶ **Definition 2.** Let S be a set of secrets such that $|S| \geq 2$, and let α be a positive real value such that $0 < \alpha \leq 1$. An α-probabilistic secret sharing scheme Π for an access structure $\mathcal{A}$ on the set of participants $\mathcal{P}_n$ and set of secrets S consists of a pair of probabilistic polynomial time algorithms $(\mathsf{Share}, \mathsf{Recon})$ where

- Share gets as input a secret $s \in S$ and outputs n shares $sh_1, \ldots, sh_n$
- Recon gets as input the shares of a subset $A \subseteq \mathcal{P}_n$, denoted by $\{sh_i\}_{i \in A}$, and outputs a string

such that the following two requirements are satisfied:

1. **α-correctness:** for every $s \in S$ and every qualified subset $A \in \mathcal{A}$, it holds that

$$Prob[\mathsf{Recon}(\{sh_i\}_{i \in A}) = s] \geq \alpha.$$

2. **perfect privacy:** for every forbidden subset $B \notin \mathcal{A}$ and for every two secrets $s_1, s_2 \in S$, it holds that the probability distributions $\{sh_i^1\}_{i \in B}$ and $\{sh_i^2\}_{i \in B}$, associated to the corresponding secrets, are the same.

▶ Remark. Notice that, when the parameter α is equal to 1, we get the traditional notion of perfect secret sharing scheme, in which the secret is *always* reconstructed by qualified subsets of participants.

▶ Remark. As pointed out in [2], the above definition can be easily relaxed to consider less stringent privacy notions, in which the probability distributions on the set of shares of forbidden subsets are not required to be identical but only statistically close or computationally indistinguishable.

The *share size* of the scheme is the maximum number of bits each participant holds in the worst case, over all participants and all secrets.

In order to introduce evolving schemes, we need to modify the setting and extend some of the previous notions. Basically, we define a sequence of access structures but require that the access structures be monotone: parties are *only added* and qualified subsets *remain qualified in the future*.

Let $\mathcal{P} = \{p_1, p_2, \dots, \}$ be an infinite set of participants.

▶ **Definition 3.** [26, 27] Let $\mathcal{A}$ be an access structure on $\mathcal{P}_n$ and let m be an integer such that $0 < m < n$. We denote by $\mathcal{A}|_m$ the restriction of $\mathcal{A}$ to $\mathcal{P}_m$, i.e., to the first m participants.

The following result holds:

▶ **Claim 4.** *[26, 27] If $\mathcal{A}$ is an access structure on $\mathcal{P}_n$, then $\mathcal{A}|_m$ is an access structure on $\mathcal{P}_m$.*

▶ **Definition 5.** [26, 27] Let $\mathbb{N}$ be the set of the natural numbers. A (possibly infinite) sequence of access structures $\{\mathcal{A}_t\}_{t\in\mathbb{N}}$ is called evolving if, for every $t \in \mathbb{N}$, the following conditions hold:

- $\mathcal{A}_t$ is an access structure over $\mathcal{P}_t$
- $\mathcal{A}_t|_{t-1}$ is equal to $\mathcal{A}_{t-1}$.

At this point, we can extend the definition of a probabilistic secret sharing scheme to evolving access structures:

▶ **Definition 6.** Let S be a set of secrets such that $|S| \geq 2$, and let α be a positive real value such that $0 < \alpha \leq 1$. An α-probabilistic secret sharing scheme Π for an evolving access structure $\{\mathcal{A}_t\}_{t\in\mathbb{N}}$ on the infinite set of participants $\mathcal{P}$ and set of secrets S consists of a pair of probabilistic polynomial time algorithms $(\mathsf{Share}, \mathsf{Recon})$ where

- Share gets as input a secret $s \in S$ and the shares $sh_1, \dots, sh_{t-1}$, generated for participants $p_1, \dots, p_{t-1}$, and outputs the share sh_t for the t-th participant
- Recon gets as input the shares of a subset $A \subseteq \mathcal{P}$, denoted by $\{sh_i\}_{i\in A}$, and outputs a string

such that the following two requirements are satisfied:

1. **α-correctness:** for every $s \in S$, for every $t \in \mathbb{N}$, and for every qualified subset $A \in \mathcal{A}_t$, it holds that

$$Prob[\mathsf{Recon}(\{sh_i\}_{i\in A}) = s] \geq \alpha.$$

2. **perfect privacy:** for every $t \in \mathbb{N}$, for every forbidden subset $B \notin \mathcal{A}_t$ and for every two secrets $s_1, s_2 \in S$, it holds that the probability distributions $\{sh_i^1\}_{i\in B}$ and $\{sh_i^2\}_{i\in B}$, associated to the corresponding secrets, are the same.

The *share size* of the scheme is the maximum number of bits the t-th participant holds in the worst case, over all secrets and previous share assignments.

▶ Remark. When the parameter α is equal to 1, we get the notion of perfect evolving secret sharing scheme, in which the secret is *always* reconstructed by qualified subsets of participants, given in [26, 27]. Notice also that the above definition, when $\mathcal{P}$ is finite and $\mathcal{A}$ is fixed and known at the beginning, i.e., $\mathcal{P} = \mathcal{P}_n$ for a certain n and $\{\mathcal{A}_t\}_{t\in\mathbb{N}} = \mathcal{A}$, it formally *does not* coincide with Definition 2. Indeed, the former generates the shares in *one-shot*, while the latter generates the shares sequentially. However, it is easy to see they are equivalent. Moreover, the considerations we have done on relaxing the privacy notion apply also here.

Notice that in [3] a unified framework for secret sharing schemes has been introduced. It enables to model together perfect, statistically a computationally secure secret sharing schemes. It also models the notion of robustness, i.e., the possibility of reconstructing a secret even if some shares are missing or corrupted, assuming an honest dealer. We point out that the 36 notions the authors have provided do not consider any relaxation of the correctness condition. Moreover, we have preferred to follow [2, 26, 27] in modeling the notion instead of extending [3] because it is easy to work with [2, 26, 27] and, perhaps, results in an abstract easier to read.

3 Probabilistic schemes for the threshold finite case

Visual cryptography schemes are a special type of secret sharing schemes for which, roughly speaking, the Recon algorithm is the `or` function[1]. Several models of visual cryptography have been studied: Kafri and Keren [24] introduced the so-called random grid model, Naor and Shamir [31] coined the term "visual cryptography" providing a deterministic model, Yang [36] introduced the probabilistic model, which actually is equivalent to the model of Kafri and Keren, and Cimato et al. [13] generalized the probabilistic model. Finally, De Prisco and De Santis [19] proved that all these models are related to each other and, thus, they can be seen simply as different ways of looking at the same object. The interested reader is referred to [17] for a recent survey on models, issues, applications and new directions in visual cryptography and to the references therein quoted.

The particular "view" that is of interest in the context of this paper is the probabilistic one: in that model we already have the notion of a probabilistic reconstruction, where a secret pixel is correctly reconstructed only with a given probability. Nevertheless, notice that in the context of visual cryptography the error is less critical than in the context of general secret sharing: indeed, the whole secret which is visually reconstructed consists of many pixels, often thousands, and, even if some of them are incorrectly reconstructed, the only tangible effect is that the secret is reconstructed on a different, usually darker, background.

Anyway, being visual cryptography a special type of secret sharing, we can use visual cryptography schemes as secret sharing schemes. Since there is basically no research on probabilistic secret sharing, it is useful to start by taking probabilistic visual secret sharing scheme and "translating" them into probabilistic secret sharing schemes. We said "translating" instead of "using" because an obvious improvement that can be made is that of using a different Recon function, since in regular secret sharing we are not constrained to using the `or` performed by the human visual system in the reconstruction process.

In [19] it has been proved that any deterministic visual cryptography scheme can be transformed into a probabilistic visual cryptography scheme. By using this result we can transform any deterministic visual cryptography scheme into a probabilistic secret sharing scheme. We explain the technique using an example.

[1] In visual cryptography schemes, the bits of the shares are pixels printed on transparencies and the reconstruction consists in superposing the transparencies; the human visual system performs the `or` operation on the "bits" in the corresponding positions.

Consider a $(3, 4)$-threshold scheme. The following deterministic scheme has been presented in [1]. The scheme is described by two base matrices, B_0 and B_1:

$$B_0 = \begin{bmatrix} 000111 \\ 001011 \\ 001101 \\ 001110 \end{bmatrix} \qquad B_1 = \begin{bmatrix} 000111 \\ 100110 \\ 010110 \\ 001110 \end{bmatrix}$$

The two base matrices define the collections $\mathcal{C}_0$ and $\mathcal{C}_1$, where $\mathcal{C}_b$, $b = 0, 1$, contains all the matrices that can be obtained by permuting in all possible ways the columns of B_b. In order to share the secret the dealer (randomly) chooses a specific permutation and gives to each participant a row of the corresponding matrix of the collection $\mathcal{C}_b$, where b is the secret bit. More specifically, participant i gets the i^{th} row of the selected matrix (permutation).

For example, considering, for simplicity, the identity permutation, if the secret pixel is $b = 0$, that is a white pixel, then participant 1 gets a share where the first three subpixels are white and the last three are black, described with the binary string 000111, while participant 2 gets a share described by 001011, that is, the subpixels are white-white-black-white-black-black; etc. Superposing three shares for the reconstruction, it is guaranteed that, if the secret pixel is white, then the reconstructed version has 4 black subpixels out of 6; while, if the secret pixel is black, then the reconstructed version has 5 black subpixels out of 6. The fact that all permutations are considered ensures security.

A probabilistic visual cryptography scheme can be easily derived from the above deterministic scheme (see [19] for more details). In such a scheme, the secret pixel (bit) is shared by giving to each participant a white (0) or black (1) pixel (bit) according to the following sets[2]:

$$\mathcal{C}_0 = \left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \right\},$$

$$\mathcal{C}_1 = \left\{ \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right\}.$$

More precisely, if the secret pixel is b, with $b \in \{0, 1\}$, the 4 participants are given shares according to one of the vectors in the set $\mathcal{C}_b$, where the specific vector is chosen *uniformly at random* in the set. The reconstruction function is the `or` of 4 bits. Looking at this scheme as a visual cryptography scheme, we have that a secret white (0) pixel is reconstructed correctly 1/3 of the times, while a secret black (1) pixel is reconstructed correctly 5/6 of the times. Assuming a uniform distribution of the secret pixel, the reconstruction is correct with probability 7/12.

On the other hand, using the `xor` function as Recon we have that in both cases the secret pixel is reconstructed correctly 5/6 of the times. This provides a 5/6-probabilistic scheme for sharing one bit, regardless of the distribution of the secret bit.

[2] Notice that these sets are easily constructed from the base matrices since each member of the set corresponds to a column of the base matrix.

The above "transform" of a visual cryptography scheme into a probabilistic secret sharing scheme can be applied to any visual cryptography scheme. In particular, it is possible to construct schemes for threshold and general access structures defined on a finite set of participants.Unfortunately, the constructions *do not extend* to the evolving case. So, we would like to find schemes that allow to manage an *evolving* set of participants.

As a side note, we point out an interesting fact: in the recent years, it has been shown that visual cryptography schemes can be used for secure computation in non standard settings, in which it is desirable to reduce the trust in digital devices [19, 18]. Here, we are going the other way around: we are using visual cryptography schemes for building probabilistic secret sharing schemes which can be used in standard digital computation.

4 A probabilistic $(2, \infty)$-threshold construction

In this section we present a construction of a probabilistic threshold scheme for the case of $k = 2$. The construction works both for a finite set of participants and for an infinite set of participants; the sharing phase is similar to the one described in Algorithm 2 of [12]. We present it as a $(2, \infty)$-threshold scheme, although the same construction works for a $(2, n)$-threshold scheme, when n is fixed in advance.

Let s be a secret bit. Let p_i be the i-th participant, with i potentially growing to ∞ (or up to a fixed n).

▶ **Construction 7.** *The construction works as follows.*

- *The* Share *algorithm takes as input the index i and produces a share for p_i as follows: for the first participant, that is for $i = 1$, the share is a random bit b_1; for the other participants, that is, for $i > 1$, if the secret is $s = 0$ then the share is b_1, the same bit given to p_1, whereas if the secret is $s = 1$ the share is a new random bit b_i. In other words, if the secret is $s = 0$, then only one random bit is produced and all participants get that random bit as their share, while if the secret is 1 then each participant gets a new random bit.*
- *The* Recon *algorithm takes in input two shares, that is two bits b_i and b_j and outputs 0 if $b_i = b_j$ (the two shares are equal), and 1 if $b_i \neq b_j$ (the two shares are different).*

▶ **Theorem 8.** *Construction 7 builds a $\frac{1+p}{2}$-probabilistic $(2, \infty)$-threshold scheme, where $(p, 1-p)$ is the distribution of the secret bit.*

Proof. We start by proving that the scheme is secure, that is any forbidden set does not have information about the secret bit. Indeed, let $B = \{p_i\}$ be a forbidden set, a set with a single participant, and let $s_1, s_2 \in S$ be two secret bits. The probability distribution of the share that p_i gets for s_1 is the same as the one that p_i gets for s_2. Indeed the share is always a random bit, so in both cases the probability distribution is $(0.5, 0.5)$ over the two possible values.

Next we prove that a qualified subset can reconstruct the secret with $(1+p)/2$-correctness. Let A be a qualified subset. In order to compute $Prob[\mathsf{Recon}(\{sh_i\}_{i\in A}) = s]$, consider the two possible cases, $s = 0$ and $s = 1$. The first one occurs with probability p and the second one with probability $(1-p)$. Thus, we have that:

$$\begin{aligned} Prob[\mathsf{Recon}(\{sh_i\}_{i\in A}) = s] \quad = \quad & p \cdot Prob[\mathsf{Recon}(\{sh_i\}_{i\in A}) = 0 | s = 0] + \\ & (1-p) \cdot Prob[\mathsf{Recon}(\{sh_i\}_{i\in A}) = 1 | s = 1]. \end{aligned}$$

When $s = 0$, all the shares are equal, thus Recon gives 0 as output and the reconstruction is always correct. Hence, $Prob[\mathsf{Recon}(\{sh_i\}_{i\in A}) = 0 | s = 0] = 1$. On the other hand, when

$s = 1$, all the shares are independent random bits, thus Recon gives in output 1 only half of the times, which means that $Prob[\mathsf{Recon}(\{sh_i\}_{i\in A}) = 1|s = 1] = 1/2$.

Therefore, we have that:

$$Prob[\mathsf{Recon}(\{sh_i\}_{i\in A}) = s] = p \cdot 1 + (1-p) \cdot \frac{1}{2} = \frac{1+p}{2}. \quad \blacktriangleleft$$

For $p = 0.5$, we get a scheme with $\frac{3}{4}$-correctness. Notice also that, when $p \neq 0.5$, since we can easily swap 0 and 1, we can always change p to be $1 - p$ to get a better correctness probability.

Finally, it is also easy to see that the above protocol can be strengthened, by using shares of 2 or more bits, constructed along the same line of the former, in order to make the probability of error as low as desired. Precisely, for sharing 0, the dealer chooses c bits uniformly at random and gives them to all participants. On the other hand, for sharing 1, to each new participant are provided c bits, chosen uniformly at random each time.

5 Transforms for general schemes from simpler ones

Once we have provided a probabilistic $(2, \infty)$-threshold construction it is a natural problem to extend it to the probabilistic (k, ∞)-threshold case. In this section we provide two different constructions for the general threshold case. The first one builds on an auxiliary probabilistic (k, ∞)-threshold construction which is used in a black-box way to build a probabilistic $(k + 1, \infty)$-threshold scheme. The probability α of correct reconstruction is preserved. For the second construction we use a family of probabilistic (j, ∞)-threshold schemes, for $j \leq k$, to obtain a probabilistic $(k+1, \infty)$-threshold scheme. The new scheme correctly reconstructs with probability at least the worse reconstruction probability of the schemes from the family.

The interesting aspect of the second construction lies in the fact that parties are grouped in generations of increasing size. The sizes of these generations are left as parameters of the construction and choosing them carefully leads to improvements in the share size with respect to the first construction.

Note that, if all auxiliary schemes provide 1-correctness, then the compiled constructions are also 1-correct; therefore, we are providing alternative constructions for the deterministic evolving (k, ∞)-threshold schemes presented in [26, 27]. In addition, if we apply to our constructions the same domain reduction technique which is used in [26, 27] to reduce the size of the shares, we get deterministic schemes for evolving access structures which achieve the same asymptotic share size of the ones obtained in [26, 27].

We denote by $[n]$ the set $\{1, \ldots, n\}$ and, for consistency of some formulas, $[0]$ denotes the empty set.

5.1 From (k, ∞)-threshold to $(k + 1, \infty)$-threshold

Let $s \in \{0, 1\}$ be the secret. The idea behind this construction is that at the arrival of party t, he receives the value $s \oplus r$, where r is a freshly and uniformly random generated bit. The bit r is then shared by using the (k, ∞) scheme with every party arriving after that moment.

More precisely, let Π denote the auxiliary (k, ∞)-threshold scheme and let Λ be the $(k + 1, \infty)$-threshold we are about to construct. At the time of arrival of party t, the share sh_t for the new scheme Λ is constructed in the following way:

1. A bit $r_t \in \{0,1\}$ is chosen uniformly at random.
2. For every $j \in [t-1]$ a new share $w_{t,j}$ of r_j is computed with Π.
3. The share of party t for the scheme Λ is the following (ordered) set of values:

$$sh_t = \{s \oplus r_t\} \cup \{w_{t,j}\}_{j \in [t-1]}$$

For the Recon algorithm of Λ, assume that $k+1$ parties $P_{t_0}, P_{t_1}, \ldots, P_{t_k}$, which are chronologically ordered, want to reconstruct. Then, the last k parties run the Recon algorithm of Π with inputs $(w_{t_1,t_0}, w_{t_2,t_0}, \ldots, w_{t_k,t_0})$ in order to recover the value r_{t_0}. Once r_{t_0} is recovered it is xored with the value $s \oplus r_{t_0}$, held by P_{t_0}, to recover s.

▶ **Theorem 9.** *If Π is an α-probabilistic scheme for the (k,∞)-threshold access structure then Λ is an α-probabilistic scheme for the $(k+1,\infty)$-threshold access structure.*

Proof. First we prove the security of Λ. Let $\mathcal{F} = \{P_{t_1}, \ldots, P_{t_l}\}$ be a set of $l \leq k$ parties which is chronologically ordered. The only value that party P_{t_j} holds which is related to the secret is $s \oplus r_{t_j}$. But then, as r_{t_j} has only been shared with players arriving later than P_{t_j} and there are at most $k-1$ of them, the value r_{t_j} is a uniformly distributed random bit from the perspective of $\mathcal{F}$ due to the perfect privacy of Π. Therefore, the value $s \oplus r_{t_j}$ is also a uniformly distributed random bit for the set $\mathcal{F}$. Thus, we have shown perfect privacy for Λ.

Next, for α-correctness, notice that the reconstruction algorithm of Π is used precisely once when reconstructing with Λ. If the value r_{t_0} is correctly reconstructed so it is the secret s. Therefore, if r_{t_0} is correctly reconstructed with probability at least α, then the same holds for s. Our result for α-correctness follows from this fact. ◀

▶ Remark. Note that for $\alpha = 1$ we get a transform for regular (i.e. non probabilistic) evolving schemes.

Share size. Let $size_\Pi(sh_t)$ denote the bitlength of the t-th share of the scheme Π. For the scheme Λ the size of the t-th share verifies

$$size_\Lambda(sh_t) = 1 + \sum_{j=1}^{t-1} size_\Pi(sh_j) \leq 1 + (t-1) \cdot size_\Pi(sh_{t-1})$$

where the last inequality holds if we assume that the shares are increasing in size, which is usually the case.

5.2 From $\{(j,\infty)$-threshold$\}_{j=2,\ldots,k}$ to $(k+1,\infty)$-threshold

Let $k \geq 2$ be an integer number. Assume that, for any $j \in \{2,\ldots,k\}$, Π_j is an auxiliary (j,∞)-threshold scheme, and let Λ be the $(k+1,\infty)$-threshold scheme we are about to construct. For consistency of notation, assume that Π_1 is a scheme that provides the secret to every participant. Let $s \in \{0,1\}$ be the secret. In this construction parties are grouped together in *generations*. For every integer $i \geq 1$, generation G_i is a set consisting of g_i consecutive participants. G_1 starts with P_1 while each subsequent generation starts with the participant following the last participant of the previous generation. For generation sizes we only require that they are an increasing sequence and all greater or equal than the target threshold, that is, $k < g_1 \leq g_2 \leq g_3 \leq \ldots$.

When a generation G_m starts, the following values are computed:

1. k random bits $r_1^{(m)}, \ldots, r_k^{(m)}$ are chosen.
2. For every $j \in [k]$, the value $s \oplus r_j^{(m)}$ is shared by using a regular (j, g_m)-threshold scheme, e.g., Shamir's scheme. Let $u_{j,l}^{(m)}$ denote the l-th share.
3. The secret s is shared by using a $(k+1, g_m)$-threshold scheme. Let $u_{k+1,l}^{(m)}$ denote the l-th share.

When the player P_t, which is the l-th player of generation G_m, arrives, the following values are computed:

4. For each $i \in [m-1]$ and $j \in [k]$ a new share for the random bit $r_j^{(i)}$ is computed by using the $(k+1-j, \infty)$-threshold scheme Π_{k+1-j}. Note that it takes as inputs all the shares of $r_j^{(i)}$ previously computed. Let $v_{j,l}^{(i)}$ denote this share.
5. The share of party t is the following (ordered) set of values:

$$\Lambda_t^{(s)} = \{u_{j,l}^{(m)}\}_{j \in [k+1]} \cup \{v_{j,l}^{(i)}\}_{j \in [k], i \in [m-1]}$$

For the Recon algorithm of Λ, assume that a set $\mathcal{F}$ consisting of $k+1$ parties want to reconstruct. Moreover, assume that m is the first index such that a party from G_m is in $\mathcal{F}$. Let split $\mathcal{F}$ in two parts, $\mathcal{F}_0 = \mathcal{F} \cap G_m$ and $\mathcal{F}_1 = \mathcal{F} \setminus \mathcal{F}_0$, that is, $\mathcal{F}_0$ consists of the parties from $\mathcal{F}$ which are in generation G_m and $\mathcal{F}_1$ consists of parties from subsequent generations. Let $k_0 > 0$ be the cardinal of $\mathcal{F}_0$ and $k_1 \geq 0$ the cardinal of $\mathcal{F}_1$. Note that $k_0 + k_1 = k + 1$. Now there are two different cases:

1. If $k_1 = 0$, that is, all players are in generation G_m, they use their $\{u_{k+1,l}^{(m)}\}_l$ shares from the $(k+1, g_m)$-threshold scheme to recover s.
2. If $k_1 > 0$, then the players in $\mathcal{F}_0$ use their $\{u_{k_0,l}^{(m)}\}_l$ shares from the (k_0, g_m)-threshold scheme to recover $s \oplus r_{k_0}^{(m)}$. On the other hand, the players in $\mathcal{F}_1$ use their $\{v_{k_1,l}^{(m)}\}_l$ shares from the (k_1, ∞)-threshold scheme Π_{k_1} to recover $r_{k_0}^{(m)}$. Then, the two values $s \oplus r_{k_0}^{(m)}$ and $r_{k_0}^{(m)}$ are xored together and the secret is recovered.

▶ **Theorem 10.** *If, for every $j \in \{2, \ldots, k\}$, Π_j is an α_j-probabilistic scheme for the (j, ∞)-threshold access structure, then Λ is an α-probabilistic scheme for the $(k+1, \infty)$-threshold access structure, where $\alpha = min_{j=2,\ldots,k}\{\alpha_j\}$.*

Proof. The result follows from a similar analysis than the one for the proof of Theorem 9. For α-correctness note that at most one of the schemes Π_j is used for reconstruction, while the other steps provide 1-correctness. ◀

Share size. The share of party t, which is the l-th participant from generation m includes $k+1$ different shares $\{u_{j,l}^{(m)}\}_{l \in [k+1]}$ for a (j, g_m)-threshold scheme. If instantiated with Shamir's scheme each of them is of size $\lfloor log_2(g_m) \rfloor$. Therefore we have:

$$size_\Lambda(sh_t) = (k+1)\lfloor log_2(g_m) \rfloor + \sum_{i=1}^{m-1} \sum_{j=1}^{k} size_{\Pi_j}(v_{j,l}^{(i)})$$

6 A probabilistic (k, ∞)-threshold construction with constant share size

It is possible to construct probabilistic (k, ∞)-threshold schemes starting from our construction provided in Section 4. In order to do so, we need to first apply iteratively the transforms from Section 5, and then apply the same domain reduction technique of [26, 27]. However, the

probabilistic (k, ∞)-threshold constructions obtained in this way have the same asymptotic share size as the ones in [26, 27]. Thus, it might seem that there is no gain in moving from the deterministic to the probabilistic scenario. However, in the following, we show how to construct schemes with better share size.

More precisely, we propose a probabilistic (k, ∞)-threshold construction with constant size shares, which is known to be impossible in the deterministic scenario, due to the lower bound presented in [26, 27]. Moreover, by choosing the parameters appropriately, the scheme can be made α-correct, with α as closer to 1 as desired, at the expenses of an increase in the size of the underlying field.

Let $q > k$ be a prime power and let $\mathbb{F}_q$ be a finite field with q elements. The idea behind this construction is giving each player an independently chosen at random Shamir share for a (k, q)-threshold scheme. In this case the secret s is an arbitrary element of $\mathbb{F}_q$, that is, $s \in \mathbb{F}_q$.

The scheme works as follows: at the beginning of the execution, a $k-1$ degree polynomial $p(x) \in \mathbb{F}_q$ is chosen as in Shamir's scheme, that is, such that $p(0) = s$. Upon arrival of participant t, a value r_t is chosen uniformly at random from $\mathbb{F}_q \setminus \{0\}$. The share of participant t is the pair $(r_t, p(r_t))$.

For reconstruction, a group of m participants checks if they hold at least k different values from $\mathbb{F}_q$ as the first component of their shares. If this is the case they recover polynomial $p(x)$ by interpolation and output $p(0)$. Otherwise, they output a random value in $\mathbb{F}_q$.

▶ **Theorem 11.** *The previous construction is an α-probabilistic scheme for the (k, ∞)-threshold access structure, where*

$$\alpha = \frac{1}{(q-1)^k} \prod_{i=1}^{k} (q-i)$$

Proof. For security, take into account that $m < k$ participants hold m shares for the Shamir's (k, q) threshold scheme and perfect privacy trivially follows.

For α-correctness, the probability of k different participants getting k different shares and thus being able to correctly reconstruct the secret equals the probability of getting k different items when choosing k times, independently and uniformly at random, from $q-1$ different items, which equals

$$\prod_{i=1}^{k} \frac{q-i}{q-1} = \frac{1}{(q-1)^k} \prod_{i=1}^{k} (q-i)$$

Note also that if $m \geq k$ participants are present upon reconstruction, the probability of getting k different values only increases. The value of α follows from these facts. ◀

Share size. A single share is a pair of two elements of $\mathbb{F}_q$, thus its size equals $2(\lfloor \log q \rfloor + 1)$.

▶ Remark. For a fixed value of k, the probability of correct reconstruction α approaches to 1 when $q \to \infty$. Therefore, it is possible to get a scheme with α as closer to 1 as desired by appropriately choosing the value of q. Of course, increasing the value of q produces an increase in the share size.

▶ Remark. As pointed out in the proof, the participation of more than k parties in the reconstruction phase makes the probability of correctly reconstructing the secret to increase.

Comparison with the scheme from Section 4. Next we provide a brief comparative analysis between our constructions from Sections 4 and 6, when both are used to share secrets of the same size for the $(2, \infty)$-threshold access structure. Assume that $s \in \{0,1\}^l$ is a bitstring of

length l. When using the construction from Section 4 in a direct way, l independent instances are needed in order to share s. Thus, the share size equals l and the probability of perfect reconstruction, for example, assuming the uniform distribution on the secret space, equals $(3/4)^l$. On the other hand, when using the construction from this section with $k = 2$, we can choose q as the smaller prime power such that $q > 2^l$. Then, the share size equals $\lfloor \log q \rfloor$ which will be equal or very close to l. And the probability of correct reconstruction equals

$$\frac{q-1}{q-1} \cdot \frac{q-2}{q-1} = \frac{q-2}{q-1}$$

Note that both constructions have almost the same share size, but the probability α of correct reconstruction for the former scheme tends to 0, while for the latter scheme tends to 1 as l grows. We can conclude, looking at concrete numbers, that our construction from this section performs much better than the one from Section 4, even for small values of l.

7 Conclusions and open problems

We have introduced the notion of α-probabilistic secret sharing schemes and provided two efficient constructions for threshold access structures and for evolving threshold access structures with shares of constant size, with respect to the number of participants, and probability of reconstruction as close to 1 as desired.

Many questions arise from the above study. We point out just two main challenging problems: the first one is how to design efficient probabilistic secret sharing schemes for *general access structures* for an infinite set of participants and which gain (if any) we can get, compared to perfect secret sharing schemes, in terms of share size. The second one is related to the *power* of probabilistic secret sharing schemes. Indeed, in [2] (Section 6), results, attributed to Rudich, show that it is unlikely to obtain efficient secret sharing schemes for certain access structures unless $NP = \text{co-}NP$. The proof uses the *perfect* correctness of the secret sharing schemes. The question is whether or not we can overcome the impossibility results by Rudich with probabilistic secret sharing schemes.

References

1. G. Ateniese, C. Blundo, A. De Santis, and D. R. Stinson. Visual cryptography for general access structures. *Inf. Comput.*, 129(2):86–106, 1996. doi:10.1006/inco.1996.0076.
2. A. Beimel. Secret-sharing schemes: A survey. In *Proceedings of the Third International Conference on Coding and Cryptology*, LNCS 6639, pages 11–46, Berlin, Heidelberg, 2011. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=2017916.2017918.
3. M. Bellare and P. Rogaway. Robust computational secret sharing and a unified account of classical secret-sharing goals. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 172–184, New York, NY, USA, 2007. ACM. doi:10.1145/1315245.1315268.
4. J. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *Proceedings on Advances in Cryptology*, LNCS 403, pages 27–35, Berlin, Heidelberg, 1990. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=88314.88328.
5. J. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *Proceedings on Advances in Cryptology (CRYPTO '88)*, LNCS 403, pages 27–35, Berlin, Heidelberg, 1990. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=88314.88328.
6. G. R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the 1979 AFIPS National Computer Conference*, pages 313–317. AFIPS Press, 1979.

7 G. R. Blakley and C. Meadows. Security of ramp schemes. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 242–268, New York, NY, USA, 1985. Springer-Verlag New York, Inc. URL: http://dl.acm.org/citation.cfm?id=19478.19498.

8 C. Blundo, A. De Santis, R. De Simone, and U. Vaccaro. Tight bounds on the information rate of secret sharing schemes. *Designs, Codes and Cryptography*, 11(2):107–110, May 1997. doi:10.1023/A:1008216403325.

9 A. Bogdanov, S. Guo, and I. Komargodski. Threshold secret sharing requires a linear size alphabet. In *Proceedings, Part II, of the 14th International Conference on Theory of Cryptography*, LNCS 9986, pages 471–484, Berlin, Heidelberg, 2016. Springer-Verlag. doi:10.1007/978-3-662-53644-5_18.

10 R. M. Capocelli, A. De Santis, L. Gargano, and U. Vaccaro. On the size of shares for secret sharing schemes. *J. Cryptol.*, 6(3):157–167, 1993. doi:10.1007/BF00198463.

11 I. Cascudo, R. Cramer, and C. Xing. Bounds on the threshold gap in secret sharing and its applications. *Information Theory, IEEE Transactions on*, 59:5600–5612, 09 2013.

12 S.-K. Chen and S.-J. Lin. Optimal $(2, n)$ and $(2, \infty)$ visual secret sharing by generalized random grids. *J. Vis. Commun. Image R.*, 23:677–684, 2012.

13 S. Cimato, R. De Prisco, and A. De Santis. Probabilistic visual cryptography schemes. *Comput. J.*, 49(1):97–107, 2006. doi:10.1093/comjnl/bxh152.

14 R. Cramer, I. Damgard, and J. B. Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, New York, NY, USA, 1st edition, 2015.

15 L. Csirmaz. The size of a share must be large. *Journal of Cryptology*, 10(4):223–231, Sep 1997. doi:10.1007/s001459900029.

16 László Csirmaz. Probabilistic infinite secret sharing. *IACR Cryptology ePrint Archive*, 2012:412, 2012.

17 P. D'Arco and R. De Prisco. Visual cryptography - models, issues, applications and new directions. In *Innovative Security Solutions for Information Technology and Communications - 9th International Conference, SECITC 2016, Bucharest, Romania, June 9-10, 2016, Revised Selected Papers*, LNCS 10006, pages 20–39, 2016. doi:10.1007/978-3-319-47238-6_2.

18 P. D'Arco, R. De Prisco, and Y. Desmedt. Private visual share-homomorphic computation and randomness reduction in visual cryptography. In *Information Theoretic Security - 9th International Conference, ICITS 2016, Tacoma, WA, USA, August 9-12, 2016, Revised Selected Papers*, LNCS 10015, pages 95–113, 2016. doi:10.1007/978-3-319-49175-2_5.

19 R. De Prisco and A. De Santis. On the relation of random grid and deterministic visual cryptography. *IEEE Trans. Information Forensics and Security*, 9(4):653–665, 2014. doi:10.1109/TIFS.2014.2305574.

20 A. Dibert and L. Csirmaz. Infinite secret sharing - examples. *J. Mathematical Cryptology*, 8(2):141–168, 2014.

21 Y. Ishai, H. K. Maji, A. Sahai, and J. Wullschleger. Single-use ot combiners with near-optimal resilience. In *2014 IEEE International Symposium on Information Theory*, pages 1544–1548, June 2014. doi:10.1109/ISIT.2014.6875092.

22 M. Ito, A. Saio, and T. Nishizeki. Secret sharing schemes realizing general access structure. In *Proc. of the IEEE Global Telecommunication Conf.*, Globecom '87, pages 99–102. IEEE, 1987.

23 M. Ito, A. Saio, and T. Nishizeki. Multiple assignment scheme for sharing secret. *J. Cryptology*, 6(1):15–20, 1993. doi:10.1007/BF02620229.

24 O. Kafri and E. Keren. Encryption of pictures and shapes by random grids. *Optics letters*, 12:377–379, 1987.

25 M. Karchmer and A. Wigderson. On span programs. In *Proc. of the 8th IEEE Structure in Complexity Theory*, pages 102–111, 1993. doi:10.1109/SCT.1993.336536.

26 I. Komargodski, M. Naor, and E. Yogev. How to share a secret, infinitely. In *Procc. of TCC 2016*, LNCS 9986, pages 485–514. Springer, 2016.

27 I. Komargodski, M. Naor, and E. Yogev. How to share a secret, infinitely. *IEEE Transactions on Information Theory*, 64(6):4179–4190, June 2018. doi:10.1109/TIT.2017.2779121.

28 I. Komargodski and A. Paskin-Cherniavsky. Evolving secret sharing: Dynamic thresholds and robustness. In *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II*, LNCS 10678, pages 379–393, 2017. doi:10.1007/978-3-319-70503-3_12.

29 H. Krawczyk. Secret sharing made short. In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '93)*, LNCS 773, pages 136–146, Berlin, Heidelberg, 1994. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=646758.705700.

30 T. Liu and V. Vaikuntanathan. Breaking the circuit-size barrier in secret sharing. Cryptology ePrint Archive, Report 2018/333, 2018. URL: https://eprint.iacr.org/2018/333.

31 M. Naor and A. Shamir. Visual cryptography. In *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, LNCS 950, pages 1–12, 1994. doi:10.1007/BFb0053419.

32 A. Paskin-Cherniavsky. How to infinitely share a secret more efficiently. *IACR Cryptology ePrint Archive*, 2016:1088, 2016.

33 R. Robere, T. Pitassi, Rossman B., and S. A. Cook. Exponential lower bounds for monotone span programs. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 406–415, Oct 2016. doi:10.1109/FOCS.2016.51.

34 A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

35 J. G. Simmons, W. Jackson, and K. Martin. The geometry of shared secret schemes. *Bulletin of the Institute of Combinatorics and its Applications*, 1, 01 1991.

36 C.-N. Yang. New visual secret sharing schemes using probabilistic method. *Pattern Recogn. Lett.*, 25(4):481–494, mar 2004. doi:10.1016/j.patrec.2003.12.011.

Extra Space during Initialization of Succinct Data Structures and Dynamical Initializable Arrays

Frank Kammer
THM, University of Applied Sciences Mittelhessen, Germany
frank.kammer@mni.thm.de

Andrej Sajenko[1]
THM, University of Applied Sciences Mittelhessen, Germany
andrej.sajenko@mni.thm.de

Abstract

Many succinct data structures on the word RAM require precomputed tables to start operating. Usually, the tables can be constructed in sublinear time. In this time, most of a data structure is not initialized, i.e., there is plenty of unused space allocated for the data structure. We present a general framework to store temporarily extra buffers between the user defined data so that the data can be processed immediately, stored first in the buffers, and then moved into the data structure after finishing the tables. As an application, we apply our framework to Dodis, Pătraşcu, and Thorup's data structure (STOC 2010) that emulates c-ary memory and to Farzan and Munro's succinct encoding of arbitrary graphs (TCS 2013). We also use our framework to present an in-place dynamical initializable array.

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

Keywords and phrases space efficiency, succinct c-ary memory, dynamic graph representation

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.65

1 Introduction

Small mobile devices, embedded systems, and big data draw the attention to space- and time-efficient algorithms, e.g., for sorting [4, 23], geometry [1, 3, 10] or graph algorithms [2, 6, 7, 9, 12, 14, 15, 16]. Moreover, there has been also an increased interest in succinct (encoding) data structures [8, 11, 12, 20, 18, 24, 22].

On a word RAM, succinct data structures often require precomputed tables to start operating, e.g., Dodis, Pătraşcu, and Thorup's data structure [8]. It emulates c-ary memory, for an arbitrary $c \geq 2$, on standard binary memory almost without losing space. Before we can store any information in the data structure, suitable lookup tables have to be computed. Hagerup and Kammer [12, Theorem 6.5] showed a solution that allows us to store the incoming information in an extra buffer and read and write values immediately. However, their solution scrambles the data so that extra mapping tables have to be built and used forever to access the data, even after the lookup tables are built. Moreover, Farzan and Munro [11] described a succinct encoding of arbitrary graphs. To store a dense graph, an adjacency matrix is stored in a compact form by decomposing the matrix in tiny submatrices. Each submatrix is represented by an index and the mapping is done via a lookup table. Assume that the tables are not ready, but the information of the graph already arrives in a

[1] Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 379157101.

stream. One then can use an extra buffer and store the small matrices in a non-compact way and operate on these matrices until the tables are ready. At the end, the non-compact matrices can be moved from the buffers to the data structure that stores the encoded graph.

Usually, the tables can be constructed in sublinear time. In this time, most of the data structure is not initialized, leaving plenty of allocated space unused. We present a general framework to store a temporarily extra buffer within the initial memory. Even if the initialized space changes, the content in the extra buffers do not change for the user. Intuitively, we describe a way how to use the uninitialized space to store a usual array with random access. In particular, if the buffer is not needed any more and thus is removed, our implementation "leaves" the data structure as if the extra buffer has never existed.

As another application of our framework, we show that in-place initializable arrays can be made dynamic and with every increase of the array size, the new space is always initialized. Our model of computation is the word RAM model with a word length w that allows us to access all input words in $O(1)$ time. Thus, to operate on a dynamic initializable array of maximum size $n_{\max}$ we require that the word size $w = \Omega(\log n_{\max})$.

To obtain a dynamic initializable array, we additionally assume that we can expand and shrink an already allocated memory. The memory can come from an operating system or can be user controlled, i.e., the user can shrink an initializable array temporarily, use the space for some other purpose, and can increase it again if it is needed later.

1.1 Previous Array Implementation

The folklore algorithm [5, 19], which uses two arrays with pointers pointing to each other and one array storing the data, uses $O(nw)$ bits of extra space. Navarro [21] showed an implementation that requires $n + o(n)$ bits of extra space. Recently, Hagerup and Kammer [13] showed that $\lceil (n(t/(2w))^t) \rceil$ extra bits for an arbitrary t suffice if one wants to support access to the array in $O(t)$ time. In particular, by choosing $t = O(\log n)$, the extra space is only one bit. Very recently, Katoh and Goto [17] showed that also constant access time is possible with one extra bit. All these array implementations are designed for static array sizes.

1.2 Our Contributions

First, we present our framework to extend a data structure by an extra buffer. We then apply the framework to two known algorithms [8, 11]. Finally, we make Katoh and Goto's in-place initializable arrays dynamic. For this purpose, we identify problems that happen by a simple increase of the memory used for the array and stepwise improve the implementation to make our array dynamic. Our best solution supports operations READ, WRITE, and INCREASE in constant time and an operation SHRINK in amortized constant time.

Why do we support the shrink operation only in amortized time? The rest of this paragraph is not a precise proof, but gives some intuition what the problems are if the goal is to support constant non-amortized time for initialization, writing and shrinking. To support constant-time initialization of an array we must have knowledge of the regions that are completely written by the user and these regions must store the information which words are written. If we now allow arbitrary shrink operations, then we do not have the space to keep all information. Instead, we must clean the information, but keep this information of the written words that still belong to the dynamic array. If we want to run the shrink operation in non-amortized constant time, the information must be sorted in such a way that the cleaning can be done by simply cutting away a last part of the information. This means

the writing operation must take care that the information is stored "almost sorted". Since we can not sort a stream of elements in $O(1)$ time per element in general, it seems plausible that the writing operation takes $\omega(1)$ time.

The remainder of this paper is structured as follows. We begin with summarizing the important parts of Katoh and Goto's algorithm in Section 2. In Section 3 we present our framework to implement the extra buffer stored inside the unused space of a data structure without using extra space. In Section 4, we extend our framework to dynamic arrays. As an application, we show in Section 5.1 how to increase the array size in constant time. In Sections 5.2 and 5.3, we present the final implementation that also allows us to decrease the array size.

2 In-Place Initializable Array

Katoh and Goto [17] introduced the data structure below and gave an implementation that uses $nw + 1$ bits and supports all operations in constant time.

▶ **Definition 1.** An initializable array $\mathcal{D}$ is a data structure that stores $n \in \mathbb{N}$ elements of the universe $Z \subseteq \mathbb{N}$ and provides the following operations:

- READ(i) $(i \in \mathbb{N})$: Returns the i-th element of $\mathcal{D}$.
- WRITE(i, x) $(i \in \mathbb{N}, x \in Z)$: Sets the i-th element of $\mathcal{D}$ to x.
- INIT(x) $(x \in Z)$: Sets all elements of $\mathcal{D}$ to initv $:= x$.

Let $D[0, \ldots, n]$ be an array storing $\mathcal{D}$. $D[0]$ is used only to store one bit to check if the array $\mathcal{D}$ is fully initialized. If $D[0] = 1$, $\mathcal{D}$ is a normal array, otherwise the following rules apply.

The idea is to split a standard array into blocks of $b = 2$ words and group the blocks into two areas: The blocks before some threshold $t \in \mathbb{N}$ are in the *written area*, the remaining in the *unwritten area*. Moreover, they call two blocks *chained* if the values of their first words point at each others position and if they belong to different areas. In the following, we denote by $d(B)$ the block chained with a block B. Note that $d(d(B)) = B$.

In our paper, we also use chains. Therefore, we shortly describe the meaning of a (un)chained block B in [17]. Compare the following description with Figure 1. If B is in the written area and chained, then B is used to store a value of another initialized block and is therefore defined as uninitialized. If B is in the unwritten area and chained, B contains user written values and the words of B are divided among B and $d(B)$. In this case $d(B)$ was not written by the user. In detail, the second word of B is stored in this word of B, but the first word is stored in the second word of $d(B)$ since the first word is used to store a pointer. If B is in the written area and unchained, B contains user written (or initial) values. If B is in the unwritten area and unchained, B has never been written by the user. Every time the user writes into a block for the first time, i.e., into a word of an (un)chained block in the (un)written area, the written area increases by moving the first block of the unwritten area into the written area and by possibly building or correcting chains.

3 Extra Space during Initialization of Succinct Data Structures

We now consider an arbitrary data structure $\mathcal{D}^*$. The only assumption that we make is that $\mathcal{D}^*$ accesses the memory via a data structure $\mathcal{D}$ realizing an n-word array. For the time being, assume $\mathcal{D}$ is the data structure as described in Section 2 with $D[0, \ldots, n]$ being an array storing $\mathcal{D}$. We define $|U|$ as the number of blocks of the unwritten area. As long as

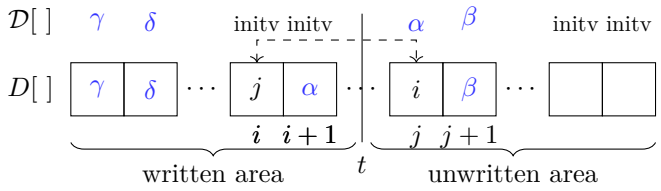


Figure 1 The different states of blocks inside the written and unwritten area. $\mathcal{D}$ represents the users view on the data with *initv* being the initial value, α, β, γ and δ as user written values. D represents the internal view with i and j as indices of D.

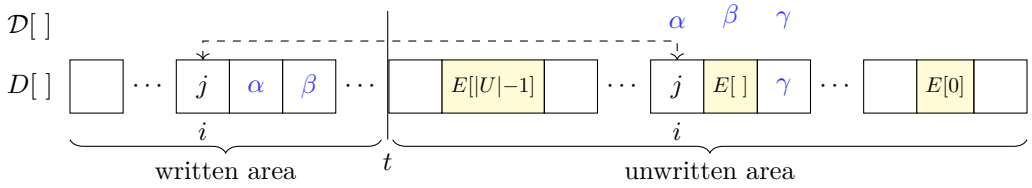


Figure 2 The block size $b = 3$ allows us to move α and β to the chained block such that each block of the unwritten area has one unused word to store the extra data of E.

$\mathcal{D}$ is not fully initialized, i.e, not all array locations have been written since the last INIT operation, and only $m \in \mathbb{N}$ with $m < n$ words are written in $\mathcal{D}$, Theorem 2 shows that the unused storage allocated for $\mathcal{D}$ allows us to store information of an extra array E inside D by increasing the block size to $b > 2$ (Figure 2). The size of E depends on $|U|$ and thus on the current grade of initialization of $\mathcal{D}$. In contrast to $\mathcal{D}$, E is an uninitialized array. Clearly, we can build an initialized array on top of E.

The array size n is not always a multiple of the block size b. By decreasing n and handling less than b words of $\mathcal{D}$ separately, we assume in this paper that n is a multiple of b.

▶ **Theorem 2.** *We can store an extra array E of $(n/b - m)(b-2) \leq |U|(b-2)$ words inside the unused space of an initializable array $\mathcal{D}$ of size n, split into blocks of size b where m is the number of writing operations since the last* INIT *operation. E dynamically shrinks from the end whenever the number of user defined values m increases.*

Proof. Let us consider a block B inside the unwritten area. B and $d(B)$ together provide space for $2b$ words, but the chain between them exists only because the user wrote inside one block (block B, but not in $d(B)$), and thus we need to store b words of user data. Furthermore, B and $d(B)$ each needs one word to store a pointer that represents the chain. By storing as much user data as possible of B in $d(B)$, we have $b-2$ unused words in B – say, words 2 to $b-1$ are unused. Note that, if a block B is unchained, then it contains no user values at all and we have even b unused words. If the user wrote m different words in $\mathcal{D}$, then the threshold t of $\mathcal{D}$ (t is the number of blocks in the written area) is expanded at most m-times. The written area consists of at most m blocks of a total of n/b blocks. Consequently, the unwritten area has $(n/b - m) \leq |U|$ blocks with each having $(b-2)$ unused words. If we start storing $E[0], E[1], ...$ in the unwritten area strictly behind t, the indices of E will shift every time the unwritten area shrinks. Therefore, we store E in reverse and E loses with every shrink the last $b-2$ words, but get static indices. ◀

Note that values like the threshold t and the initial value can be easily stored at the beginning of E. However, we require the size of $\mathcal{D}$ to determine the beginning of E. To store the size we therefore move $D[1]$ also into E and store the size of the array in $D[1]$. During

the usage of D we have to take this into account, but for simplicity we ignore this fact. If the array is fully initialized and thus $D[0] = 1$, we can not store the size anymore, but in this case we do not need to know it.

3.1 Application: c-ary Memory

To use Dodis et al.'s dictionary [8] on n elements, a lookup table Y with $O(\log n)$ entries consisting of $O(\log n)$ bits each must be constructed, which can be done in $O(\log n)$ time. Using tables of the same kind to store powers of c, we can assume that $c = \Omega(n)$ by combining consecutive elements of the input into one element.

Hagerup and Kammer extended Dodis et al.'s dictionary to support constant-time initialization by storing the $k = \Theta(\log n)$ elements that are added first to the dictionary in a trie D^{T} of constant depth $d \geq 4$ and out degree $n^{1/d}$ using $O(n^{\epsilon+1/d} + \log n \log c)$ bits for any $\epsilon > 0$. Interleaved with the first k operations on the dictionary, first the table Y is built and afterwards, the elements in D^{T} are moved to the dictionary.

Since D^{T} is required only temporarily, it is stored within the memory allocated for the dictionary – say in the last part of the memory. The problem that arises is that the last part of the memory can not be used as long as it is still used by D^{T}. This problem is solved by partitioning the memory allocated for the dictionary into sectors and using a complicated mapping function that scrambles the elements to avoid the usage of the last part of the memory. Unfortunately, even after computing the table Y and moving all elements from D^{T} over to the dictionary, the data is still scrambled and each access to the dictionary has to start evaluating the mapping function.

With Theorem 2 as an underlying data structure it is easily possible to implement the dictionary with constant initialization time. Use the extra array E to store temporarily the table D^{T}. Even if we use block size $b = 3$, E has enough space (at least $(n(\log c) - k \log n)/3 \geq (n - k)(\log n)/3$ bits at the end of the construction of Y) to store D^{T}. The mapping function becomes superfluous because the data is not scrambled by the usage of Theorem 2. Furthermore, working on a copied and slightly modified algorithm after the full initialization of $\mathcal{D}$, we can avoid checking the extra bit in $D[0]$.

3.2 Application: Succinct Encoding of Dense Graphs

Farzan and Munro [11] showed a succinct encoding of an $n \times n$-matrix that represents an arbitrary graph with n vertices and m edges. Knowing the number of edges they distinguish between five cases: An almost full case, where the matrix consists of almost only one entries, an extremely dense case, a dense case, a moderate case and an extremely sparse case. For each case, they present a succinct encoding that supports the query operations for adjacency and degree in constant time and iteration over neighbors of a vertex in constant time per neighbor.

We consider only the dense case where table lookup is used. Dense means that $\exists \delta > 0 : n^2 / \log^{1-\delta} n \leq m \leq n^2(1 - 1/(\log^{1-\delta} n)$. As shown in [11], a representation in that case requires $\log \binom{n^2}{m} + O(n^2(\log^{1-\delta} n)) = \log \binom{n^2}{m} + o(\log \binom{n^2}{m})$ bits of memory. To encode the matrix of a graph Farzan and Munro first divide the matrix into small *submatrices* of size $\log^{1-\delta} n \times \log^{1-\delta} n$ for a constant $0 < \delta \leq 1$. For each row and column of these smaller submatrices they calculate a *summary bit* that is 1 if the row and column, respectively, of the submatrix contains at least one 1. The summary bits of a row and column, respectively, of the whole matrix are used to create a *summary vector*. On top of each summary vector they build a rank-select data structure [24] that supports queries on 0's and 1's in constant time. They also build a lookup table to map between possible submatrices and indices as well as to

answer all queries of interest in the submatrix in $O(1)$ time. In a further step, they replace each submatrix by its index. By guaranteeing that the number of possible submatrices is $o(\log n)$, the construction of the lookup table can be done in $O(n)$ time. To simply guarantee this, we restrict δ to be larger than $1/2$. We generalize the result to dynamic graphs by replacing the rank-select data structures with choice dictionaries [12, Theorem 7.6] as follows. We assume that the graph has initially no edges and that there is a stream that consists of edge updates and query operations. With each edge update, the index of the submatrix with the edge changes. To realize the index transition we use a translation table that maps both, the current index of the submatrix and the edge update made, to the new index. We store for each summary vector an *edge counter*, i.e., the number of 1's that was used to calculate each summary bit of it. Whenever an edge is created between two nodes we set the corresponding bit of the row and column, respectively, inside the summary vector to 1 and increment the edge counter. If an edge is removed we decrement the edge counter and determine, using a lookup table, if the submatrix containing this edge has still 1's in the updated row and column, respectively. If it does not, we set the corresponding summary bit inside the summary vector to 0.

If the lookup tables for the submatrices are computed, we can easily answer queries to the current graph similar to [11] since we still use the summary vectors. We can ask for membership to answer adjacency queries. To iterate over the neighbors of a vertex (i.e., over the 1's in the summary vector), we can use the iterator function of the choice dictionary instead of using the select function of the rank-select data structure. The degree query is answered by returning the edge counter.

In the rest of this subsection, the goal is to extend the data structure such that it supports the queries without a delay for the construction of the lookup tables. The idea is to store a submatrix with a 1 entry in the usual, non-compact way in the extra buffer and to add a pointer from the submatrix to a place in the buffer where it is stored. To reduce the space used for the pointers, we group $\log^{\delta} n$ submatrices together to get a *group matrix* of size $\log^{1-\delta} n \times \log n$. Moreover, we use an array A in which we store a pointer of $O(\log n)$ bits for each group. With the first writing operation to one submatrix in a group, the group matrix is stored in the usual, non-compact way in an initialized array, which is stored in the extra buffer (Theorem 2). In the array A, we update the pointer for the group. In addition, we build a choice dictionary on top of each non-empty column and each non-empty row of the group matrix. With each choice dictionary, we also count the number of 1 entries.

Using the counts we can update the summary vectors and the non-compact submatrix allows us to answer adjacency queries. Both can be done in constant time without using lookup tables. To support the neighborhood query, we use choice dictionaries on top of the group matrices. In detail, whenever updating an edge, update the summary vector, add or follow the pointer to the non-compact representation, update it including the choice dictionary and the degree counter.

After $O(n)$ operations the lookup table is computed and, in the same time, the entries in the non-compact matrices are moved to the compact submatrices.

The array A requires $n^2(\log n)/((\log^{1-\delta} n)(\log n)) = O(n^2/\log^{1-\delta} n)$ bits, which is negligible. Moreover, a group matrix with the choice dictionaries requires $O(\log^2 n)$ bits to be stored. Thus, after $O(n)$ operations, the total extra space usage is $O(n \log^2 n)$ bits, which easily fits in the extra buffers of size $\Omega(\log \binom{n^2}{m} - n \log n) = \Omega(n^2/\log^{1-\delta} n - n \log n)$ bits after $O(n)$ operations.

▶ **Theorem 3.** *A graph with n vertices can be stored with $log_2\binom{n}{m} + O(n^2/\log^{1-\delta} n)$ bits supporting edge updates as well as adjacency and degree queries in constant time and iteration over neighbors of a vertex in constant time per neighbor where $m = n^2/\log_2^{1-\delta} n$ and $\delta > 1/2$. A startup time for constructing tables is not necessary.*

Note that, if the whole memory of a graph representation must be allocated in the beginning, i.e., if we do not allow a change of the space bound during the edge updates, then our representation of the dynamical graph (with possibly $n^2/\log_2^{1-\delta} n$ edges) is also succinct.

4 Extra Space for Dynamic Arrays

We now extend our framework to dynamic arrays. The data structure $\mathcal{D}$ (Definition 1) of Katoh and Goto is not dynamic, but assume it is. Then the unwritten area of $\mathcal{D}$ may become larger and smaller and with it the size of the array extra array E (Theorem 2).

Since we start indexing the words belonging to E from the end of $\mathcal{D}$, changing the size of $\mathcal{D}$ will change the index of the words in E. If we start indexing E from the beginning of the unwritten area, all indices change every time the unwritten area shrinks.

To realize an indexed array we introduce a fixed-length array F. To handle the shrink operation, we store F strictly behind the written area so that there is nothing to do if we shrink or expand the size of $\mathcal{D}$. Recall that b is the size of a block words and m is the number of write operations. If the written area grows, we lose $b-2$ unused words of the unwritten area and therefore $b-2$ used words in F. As long as there are unused words in the unwritten area behind F we move the $b-2$ words that we would lose to the first unused words behind F. This will rotate F inside the unwritten area of D.

▶ **Lemma 4.** *We can store a fixed-length array F consisting of ℓ words in the unused space of $\mathcal{D}$ as long as $\ell \leq (n/b - m - 1)(b-2) \leq (|U|-1)(b-2)$. If the condition is violated, then F is destroyed.*

Proof. Let the size of F be smaller than the unused space of $\mathcal{D}$. Whenever the user writes an additional word in $\mathcal{D}$, m increases and the number of the unused words becomes less. Therefore, before extending the written area by one block, we move the $b-2$ words in that block to the unused space of $\mathcal{D}$ behind F. We use the last block in the unwritten area to store the size and a counter to calculate the rotation. Thus, we have one block less in contrast to Theorem 2. F can be of size $\ell \leq (n/b - m - 1)(b-2) \leq (|U|-1)(b-2)$. ◀

For some applications it is interesting to have a dynamic extra storage even if $\mathcal{D}$ is dynamic. We introduce a dynamic set in Lemma 5 that can be stored inside the unwritten area of $\mathcal{D}$ and supports the operations ADD, REMOVE, and ITERATE. The last operation returns a list of all elements in the data structure. The size of dynamic extra storage is dynamically upper bounded by $|U|$.

▶ **Lemma 5.** *We can store a dynamic (multi-)set of maximal $\ell \leq (|U|-1)(b-2)$ elements in the unused space of $\mathcal{D}$. If the condition is violated by write operations of $\mathcal{D}$, elements of the set are removed until the condition holds again.*

Proof. We store a dynamic list strictly behind the written area and shift it cyclically as in the proof of Lemma 4. The difference here is that we have no fixed order so that we can store new elements simply at the beginning of the unused words of $\mathcal{D}$ and increase the size ℓ. The removal of an element may create a gap that can be filled by moving an element and decreasing ℓ. ◀

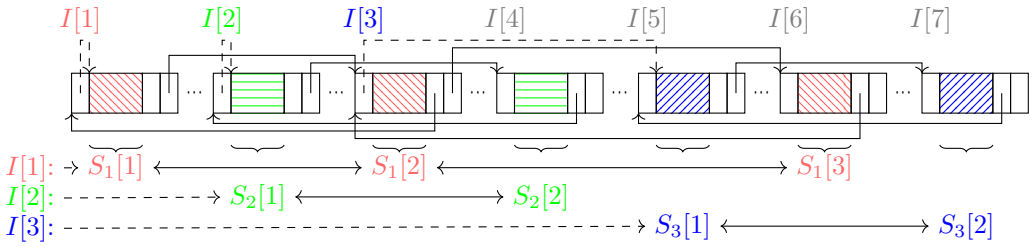


Figure 3 The figure shows a dynamic family consisting of three dynamic sets. The first element of each set can be found in section to which $I[1]$, $I[2]$ and $I[3]$, respectively, points.

For our implementation described in Section 5 we require a (multi-)set with *direct access* to the elements. We can use the position in D as an address for direct access, but this requires that the user implements a *notification function*, which is called by our data structure whenever an element in the list changes its position to inform the user.

▶ **Fact 6.** *Having two chained blocks B and $d(B)$ and a suitable block size b, we can rearrange the user data in the two blocks such that we can store $O(1)$ extra words (information as, e.g., pointers) in both B and in $d(B)$.*

We next show that an algorithm can use several extra data structures in parallel.

▶ **Lemma 7.** *For a $b \geq 2$, $\mathcal{D}$ can have $k \in \mathbb{N}$ extra data structures in parallel where each is of size at most $|U|\lfloor (b-2)/k \rfloor$.*

Proof. Every block of the unwritten area has $b-2$ unused words and every unused word can be used for another data structure. ◀

As we see later, it is useful to have several dynamic sets of elements that are all the same size. The sets can also be (multi-)sets.

▶ **Corollary 8.** *For $b > 2$, $\mathcal{D}$ can store a dynamic family $\mathcal{F}$ consisting of dynamic sets where the sets have in total at most $\lfloor (|U|-3)(b-2)/(s+3) - 1 \rfloor$ elements each of size s.*

Proof. We use the unused words in the unwritten area of $\mathcal{D}$ and partition it into sections of $s+3$ words. Every first word of a section is used for an array I, then s words are used to store an element of a set, and the last two words store pointers of a doubly linked list that connect the element in a doubly-linked list with all other elements in the set. $I[i]$ points to the an element of the ith set. The element is stored in one section. Using the pointers at the end of each section, we can find the next element of the set. The realization is also sketched in Figure 3. The unused words in the last 3 blocks (possibly, fewer blocks suffice) of the unwritten area are used to store some constants: the size t of the written area, the initial value of $\mathcal{D}$, some parameter $p \in \mathbb{N}$, the number of sets in $\mathcal{S}$, and the total number q of items over all sets. Using q we can simply find the section of a new element.

If the written area of $\mathcal{D}$ increases, we have to start moving the first section to a new unused section. We store the position of the new section in p. By knowing t we know how much of the first section has been already moved and where to find the information of the first section. ◀

5 Dynamical Initializable Array

In the next three subsections, we provide implementations to make the in-place initializable array dynamic. Therefore, we extend the initializable array $\mathcal{D}$ by the following two functions to change the current size of the array.

- INCREASE$(n_{\text{old}}, n_{\text{new}}, \text{initv})$ $(n_{\text{old}}, n_{\text{new}} \in \mathbb{N})$: Sets the size of $\mathcal{D}$ from n_{old} to n_{new}. All the elements behind the n_{old}th element in $\mathcal{D}$ are initialized with initv, the initial value of $\mathcal{D}$ defined by the last call of INIT.
- SHRINK$(n_{\text{old}}, n_{\text{new}})$ $(n_{\text{old}}, n_{\text{new}} \in \mathbb{N})$: Sets the size of $\mathcal{D}$ from n_{old} to n_{new}.

In our implementation we handle the array $\mathcal{D}$ differently, according to its size. Recall that w is the word size. As long as $\mathcal{D}$ consists of $n' = O(w)$ elements, we use a bit vector stored in $O(1)$ words to know which words are initialized. On the word RAM we can manipulate it in constant time. To have immediate access to the bit vector we store it in the beginning of the array and move the data located there to the first unused words. If we increase the size of $\mathcal{D}$ to more than $\omega(w)$ elements, we keep the bit vector in some unused words until the first n' words of $\mathcal{D}$ are completely initialized. Having such a bit vector we assume that this is taken into account whenever there is a reading or writing operation, but do not mention it explicitly in the rest of the paper.

If the array $\mathcal{D}$ consists of at least $\omega(w)$ elements, we can not use the solution with the bit vector. Instead, we use chains so that we can distinguish between initialized and uninitialized blocks, even after several shrink and increase operations. We assume in the following that $\mathcal{D}$ consists of $\omega(w)$ elements.

5.1 Increasing the size of $\mathcal{D}$

An increase of the size of $\mathcal{D}$ means allocating additional memory that gives us several new blocks inside the unwritten area. These blocks may contain arbitrary values and these values can point at each other such that they create a so-called unintended chain between a block B_w (written area) and a block B_u (unwritten area). Katoh and Goto *break such unintended chains* by creating a self-chain (pointer at its own position) with B_u whenever they write a value to a block of the written area. Because the increase of $\mathcal{D}$ may be arbitrary, we can not destroy such chains in constant time.

To support large increases of $\mathcal{D}$, we eliminate the possibility of unintended chains by introducing another kind of chain, called *verification chain*. To distinguish the two kinds of chains we name the chain introduced in Section 2 a *data chain*. We define a chain between two blocks B_w and B_u as *intended* exactly if B_w has also a verification chain, otherwise as *unintended*.

To use the verification chain, we first distribute the user data among two chained blocks such that each block of the written area has an unused word (Fact 6). Let L be a dynamic list with direct access stored in $\mathcal{D}$ (Lemma 5). Whenever the block B in the written area has a data chain with a block $d(B)$, we additionally require that it also must have a verification chain with an element of the list L (Figure 4). We denote by $v(B)$ a pointer to the element in L chained with B and use an unused word of B to store $v(B)$. Recall that, whenever the unwritten area shrinks, some elements in L change their position in D. To ensure the validity of the verification chains, we use the notification function of L to update the pointer $k \in \mathbb{N}$ in the effected blocks. The verification pointers in L, stored behind the written area, are not affected by increasing the size of $\mathcal{D}$.

Finally, note that L has enough space to store all verification pointers since only one verification chain is required for each block in the unwritten area, i.e, L is of size $O(|U|)$.

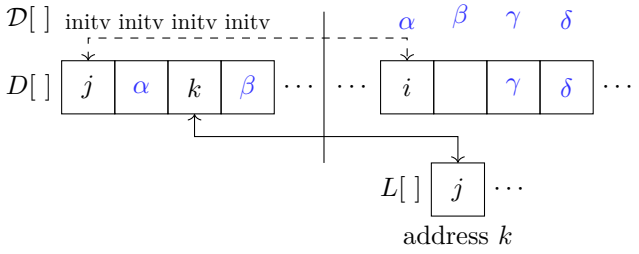


Figure 4 The block B_w (left) has a verification chain with an element of a dynamic list L to verify that the data chain between block B_w and B_u (right) is intended.

5.2 Shrinking the Size of $\mathcal{D}$ at most $O(w)$ Times

Shrinking the size of $\mathcal{D}$ means to free some memory that may be used to store information. We distinguish between two cases. In the first case we shrink the size of $\mathcal{D}$ so much that L is destroyed. In this case we make a *full initialization* of $\mathcal{D}$ as follows.

Since we lose the ability to check for unintended data chains, we first destroy them by iterating over the verification list L, following its pointer to a block B_w, checking if the block $B_u = d(B_w)$ is behind the new size of $\mathcal{D}$. If it is, we initialize B_w with the initial value. Writing the initial value into B_w may created an unintended data chain, which we break. Now the verification list is superfluous and we can iterate over the unwritten area and check for a data chain. If there is one, we move the user values from $d(B)$ into B and initialize $d(B)$, otherwise we initialize B. After the iteration, we set $D[0] = 1$.

Since L will be destroyed by the shrinking operation, the size $|U|$ of the unwritten area behind the shrinking is bounded by $|L|$. Thus, by using the potential function $\psi = (c \cdot \text{length of } L)$ for some constant $c \geq 2$, the amortized cost of the WRITE operation increases by at most $c = O(1)$ and we can easily pay for the full initialization.

In the second case, we reduce the size of $\mathcal{D}$ such that the verification list L is still completely present in $\mathcal{D}$. In this case, all blocks of the written area remain and they still have a verification chain. However, the data chain can point outside of $\mathcal{D}$. If we subsequently re-increase $\mathcal{D}$, some blocks may still have a data chain and also a verification chain. This violates our definition of the INCREASE operation.

To resolve this problem we invalidate the data chains of all blocks outside $\mathcal{D}$ as follows. As long as there is no shrinking operation, we store the same *version number* $v \in \mathbb{N}$ to all data chains. More exactly, let B be a block in the unwritten area chained with a block $d(B)$ in the written area. Then we store the version inside an unused word of $d(B)$ using Fact 6.

Before we execute a shrink operation, we set n_v to the current size of $\mathcal{D}$ and remember it as the size for the current version v. With the shrink operation, we increment the version number by one. Let v' be the version of the chain between B and $d(B)$. We call the chain *valid* if B is before the boundary $n^*_{v'} = \min\{n_v | v \geq v'\}$. Note that $n^*_{v'}$ is the minimal boundary that $\mathcal{D}$ ever had after introducing version number v'.

We obtain the boundaries n^*_v from a data structure M that provides an operation to add the new size of $\mathcal{D}$ after a shrink operation and another operation MINBOUND to check if a chain is valid, i.e., if one endpoint of the chain is or was outside of $\mathcal{D}$. For later usage, M also supports an operation REMOVE.

- ADD(n) ($n \in \mathbb{N}$): Increments an internal version counter v by one and set the boundary $n^*_v = n$. All boundaries n^*_i ($i \in \{1 \dots v-1\}$) larger than n are overwritten by n.
- REMOVE(j) ($0 \leq j \leq v$): Decrements v by j and removes the boundaries of the largest j versions.
- MINBOUND(v) ($0 \leq v < w$): Returns n^*_v, the minimal boundary for v.

▶ **Lemma 9.** *M can be implemented such that it uses $O(w)$ words,* ADD *runs in amortized constant time, and* MINBOUND *and* REMOVE *run in constant time as long as there are only $O(w)$ versions.*

Proof. We have a version counter v^* and a table T where we store the initial boundary for each version. Moreover, we use a stack S that additionally allows us to access the elements of the stack directly and the data structure R from Pătraşcu and Thorup [25]. All three data structures can be implemented using a fixed size array and stored in extra buffers (Lemmas 4 and 7). R consists of all versions v with $n_v = n_v^*$. The stack S store the boundaries of the versions in R in ascending order from the bottom to the top. For a technical reason, S stores also the version with each boundary.

To answer the MINBOUND operation for a version v, we first determine the successor v' of v in R and then return $n_v^* = T[v']$.

Assume that a new boundary $n_v = n$ is added to M for a next version v. Set $T[v] = n$. Before adding a new boundary n_v to S we remove all boundaries from S as long as the top of S is larger than n_v. Whenever removing such a boundary, we remove the corresponding version from R. Finally, we add the new boundary to S and to R. By doing this, we remove all boundaries that are larger than n_v. Now all the versions of these boundaries get n_v as their new boundary.

Finally, it remains to check the running time. We use the potential function $\phi = |S|$ with $|S| \leq w$ being the current size of S. Determining the boundary for a specific version requires looking into R and to check one word in S and T. This does not change ϕ and runs in $O(1)$ time. If the running time for adding or removing versions is not constant, then in all except a last iteration, we remove an element from the stack. Thus, the decrease of ϕ pays for this. ◀

▶ **Corollary 10.** *For every data chain, we can check in constant time if it is valid or not.*

Proof. A chain between some blocks B and $d(B)$ is valid exactly if $d(B) \leq$ MINBOUND(v) is true, where v is the version of the chain and $d(B)$ is the position of the block that belongs to the unwritten area. ◀

Whenever a block in the unwritten area has an invalid chain (caused by shrinking and re-increasing $\mathcal{D}$; determined by MINBOUND), we return the initial value for all words of the block. We want to remark that the structure M can maintain only $O(w)$ versions since it uses a dictionary from Pătraşcu and Thorup [25]. The dictionary is dynamic and supports modification and access in constant time, but only for $O(w)$ entries.

We have to make sure that we have only $O(|U|)$ chains (counting both valid and invalid chains) so that we are able to store them. The problem is that a shrink can invalidate many chains. Therefore, we have to *clean-up* our data structure $\mathcal{D}$ as follows: Whenever we add a chain, we check the validness of three old chains, i.e., chains with an old version number that are stored at the beginning of the unwritten area in L. Invalid chains are removed whereas valid chains are assigned to the current version. To have the time for the clean-up, we also modify the shrink operation such that, after each shrink operation, we make sure that we store at most $|U|/2$ chains. If this is not the case, we run a full initialization. By assuming that every insert into $\mathcal{D}$ pays a coin, this can be done in amortized constant time since the number of chains that we have is bounded by the number of insert operations.

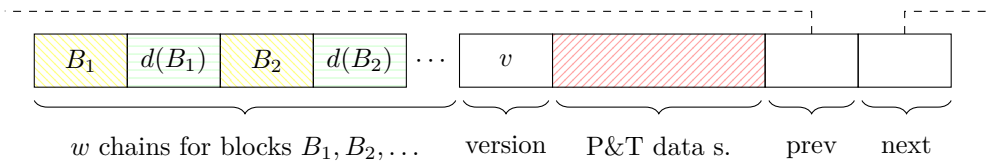


Figure 5 A subgroup that is embedded as an element of the dynamic doubly linked list containing several block positions of a version v that are indexed with a dictionary.

5.3 The General Case

The goal in this section is to limit the number of versions to $O(w)$ such that the data structure M from Lemma 9 can be always used. We achieve this goal by *purifying* the chain information, i.e., for some old version v', we iterate through the chains with a version larger than v', remove invalid data chains and store the remaining chains under v'. Finally, we take v' as the current version. We so remove all versions larger than v', which now can be reused.

Since we have no data structure to find invalid chains out of a large amount of all chains, we partition the chains of each version into small subgroups and use also here the dictionary from Pătraşcu and Thorup [25] to index the block positions B and $d(B)$ that represent the chain. The dictionary from the subgroup with a version v allows us to find a block that lies behind a boundary n_v^* in constant time as long as such a block exists.

In detail, a subgroup always has space for $\Theta(w)$ chains (block positions), the version v, and a data structure from Pătraşcu and Thorup [25] (Figure 5). To organize the subgroups we use a dynamic family S from Corollary 8. For each version v, we create a dynamic set in S. Each set consists of all subgroups of the same version. Instead of having a verification chain with a list L, each chained block in the written area has now a pointer to its subgroup. If the block is indexed in the subgroup, then its chain is verified.

As in the previous section we use the data structure M to store and check the boundaries. Additionally, we store the version v and a table T. The table is used to store, for each version, the number of chains that are currently used. The updates of T are not described below explicitly. Based on the information in T we determine how many versions we can purify. The details of the PURIFY operation are described on the next page.

Note that M and T are of $O(w)$ words and the size of S is linear in the size of the number of chains and thus bounded by $O(|U|)$. Choosing the block size b large enough, but still $b = O(1)$, we can guarantee that all data structures fit into $|U|$ unless $|U| = O(w)$. By running the clean-up of the last section, M and T can be removed if to many writing operations shrink the unwritten area of $\mathcal{D}$ so much that $|U| = O(w)$.

- INITIALIZE(n, initv) Allocate $nw + 1$ bits. Partition the array into blocks of size $b = 6$. If n is not a multiple of b, initialize less than b words and treat them separately. Use the last block(s) to store the threshold $t = 0$, initv, and $v = 1$ in the unused words. Initialize the data structure M and S, store their internally required single words also in the last block(s) and their sets and lists in parallel (Lemma 7).
- READ(i): If $D[0] = 1$, return $D[i]$. Else, check to which area $B = \lfloor i/b \rfloor$ belongs. If B is inside the written area, check if B is verified. If it is, return initv, otherwise $D[i]$.
  If B is inside the unwritten area, check if B is chained with a block $d(B)$. If it is not, return initv, otherwise proceed with checking if $d(B)$ is verified. If it is not, return initv,

otherwise follow the verification pointer and read the version v out of the subgroup. Call M.MINBOUND(V) and return initv if it returned a boundary that is larger than the block position B, otherwise return the right word out of B and $d(B)$.

- WRITE(i, x): If $D[0] = 1$, write at $D[i]$. Otherwise, clean-up three chains as described in the last subsection. Then check to which area $B = \lfloor i/b \rfloor$ belongs. If B is inside the written area, check if B is verified. If it is not, write at $D[i]$ directly. Otherwise, the block may have a data chain with a block $d(B)$. If so, unchain it (like in [17]) by expanding the written area that gives us a new unused block that we use to relocate all values and chains from B. Correct also the chains in the subgroups. If not, just remove B and $d(B)$ from its subgroup. In both cases, B becomes an unused block afterwards. We initialize it and write at $D[i]$ directly. If B is inside the unwritten area, check if B is chained with a block $d(B)$. If it is, check if $d(B)$ is verified and if its chain belonging to a version v is valid (test M.MINBOUND$(v) \geq B$). If all is true, then write x at the right position of B and $d(B)$. If the chain is invalid, delete it from its subgroup and move it into the latest subgroup of the current version. Finally, initialize B and write x as described above.

  But if B is not chained, expand the written area and chain it with the new block. Write both blocks inside the latest subgroup of the current version. Set a verification pointer to the subgroup where the chain is stored. As before, initialize B and write x as described above.

  In all cases, whenever the unwritten area disappears, set $D[0] = 1$.
- INCREASE(n_{old}, n_{new}, initv): If $D[0] = 0$, then copy the words of the last block(s) into the new last block(s). Otherwise, initialize the required data structures as described in INITIALIZE, but set the initial values for the threshold t to $t = \lfloor n_{\text{old}}/b \rfloor$. If $|U| = O(w)$, use the bit vector solution described in the beginning of Section 5.
- PURIFY(n_{new}): Iterate from the current version $v = \lceil w \rceil$ in S down and add up the number of chains stored under a version. Stop at the first version v' that has fewer chains as twice the total number of chains of all versions visited before. Consider versions $v' + 1$ to v and iterate over all subgroups with that version. In each subgroup, check for an invalid chain (B, $d(B)$) by using M, remove it from the subgroup, from the index of the subgroup, and initialize B with initv and repeat this on the subgroup until it has no invalid chains. In the special case where the subgroup has less than w chains, clear the subgroup by moving all chains into the latest subgroup of the version v'.

  Then, change the version number of this subgroup to v', unlink it from its old set and link it into the set of v'. When finished, set the current version number v to v'.
- SHRINK($n_{\text{old}}, n_{\text{new}}$): If the new size of $\mathcal{D}$ either cuts the space used to store the data structures S or M or leaves less than $\Theta(w)$ unused words in the unwritten area, run the full initialization.

  If we do not fully initialize $\mathcal{D}$, we proceed as follows: If we have $\lceil w \rceil$ versions, call PURIFY. Otherwise, create a new set inside S and increment the current version number v by one and check if the last used subgroup has less than w entries. If so, we reuse this subgroup by removing all invalid chains. (In PURIFY we already described the removal.)

Who pays for the purification? The idea is to give a gold coin to each previous set in S, whenever we insert an element. Whenever a set of size x has at least $x/2$ coins, we clean up the set and all sets with a larger version number. Algorithmically we can not check fast enough if a set already has enough coins. Therefore, we wait with the purification until we have $\lceil w \rceil$ versions and check then the condition for every version from the largest version to the smallest until we found the first version with enough coins.

▶ **Theorem 11.** *There is a dynamic array that works in-place and supports the operations* INITIALIZE, READ, WRITE *as well as* INCREASE *in constant time and* SHRINK *in amortized constant time.*

Proof. It remains to show the running times of the operations. Take ψ as the total number of verification chains similar to Subsection 5.2, c_i as the number of chains with the version $i \in \{0, \dots, v-1\}$ and $g = \sum_{i=0}^{v-1} \frac{6i \cdot c_i}{w}$. Moreover, choose τ as the total number of missing elements such that all subgroups are full and $f = (1 - D[0]) \max\{0, 2w - |U|\}$. We use the following potential function $\phi = \psi + v + g + \tau + f$. Note that g corresponds to the usage of the gold coins and f is non-zero only if there are very few blocks in the unwritten area U, but $\mathcal{D}$ is not completely initialized.

- INITIALIZE: We have to set the threshold t and the current version in $O(1)$ time. Thus, $\phi = v = 1$ – note that $f = 0$ by using the bit-vector solution if necessary.
- READ: Checking chains, reading the version number and calling the MINBOUND function of M can be done in constant time and ϕ does not change.
- WRITE: We possibly have to check if a block is chained, verified and valid. We also may create a chain, a verification and insert it into S. All these operations require $O(1)$ time. In total, the number of chains may increase by one $\Delta\psi = O(1)$, $\Delta g = \frac{6v}{w}$ with v being the current version. Thus, $\Delta\phi = O(1)$.
- INCREASE: Copying a few blocks runs in $O(1))$ time and $\Delta\phi = 0$ since $\Delta f = 0$.
- PURIFY: Whenever we run PURIFY we remove the last $v - i$ versions for the largest i with $v_i/2 \leq y := \sum_{i+1}^{v-1} c_i$ is valid. Thus, we first have to search for version i, i.e, we consider $v - i$ versions. Moreover, we then have to iterate over $3y$ chains in $6y/w$ subgroups – a subgroup is always at least half full. In each subgroup we spend $1 + d$ time, where d is the number of deleted chains in each subgroup. This can be done in at most $(v - i) + 6y/w + d^*$ time units where d^* is the total number of deleted chains. In addition, we may have deleted half empty subgroups and moved their chains to the latest subgroup. This can be done in $O(m)$ time if m is the number of moved chains.

  We can pay for the running time since the value of the potential function decreases as follows. By deleting versions larger than i, $\Delta v = -(v - i)$. Since we change the version of all subgroups with a version larger than i to i, i.e., we shrink the version of y chains by at least one, $\Delta g = -6y/w$. In addition, $\Delta(\psi + \tau) \leq -2d^* + (d^* - m) = -d^* - m$ where the $-m$ comes from the fact that half empty subgroups disappear, i.e, are not taken into account in τ. To sum up, the amortized running time is $O(1)$.
- SHRINK: In Subsection 5.2, we already analyzed the case that $\mathcal{D}$ is fully initialized – the only change is that we have further parts in the potential function, but their change is either zero or negative. Otherwise, we have to delete invalid chains in the latest subgroup and update it to the new version, which can be done in $O(1)$ amortized time since $\Delta\psi$ drops linear in the number of deleted chains. Moreover, we have to add a new version to M and possibly delete several older versions. All this can be done in $O(1)$ amortized time. And finally, we may run a PURIFY; again in $O(1)$ amortized time.

We thus have shown all running times as promised in the theorem. ◀

References

1 Tetsuo Asano, Kevin Buchin, Maike Buchin, Matias Korman, Wolfgang Mulzer, Günter Rote, and André Schulz. Reprint of: Memory-constrained algorithms for simple polygons. *Comput. Geom. Theory Appl.*, 47(3, Part B):469–479, 2014. doi:10.1016/j.comgeo.2013.11.004.

2 Tetsuo Asano, Taisuke Izumi, Masashi Kiyomi, Matsuo Konagaya, Hirotaka Ono, Yota Otachi, Pascal Schweitzer, Jun Tarui, and Ryuhei Uehara. Depth-first search using $O(n)$ bits. In *Proc. 25th International Symposium on Algorithms and Computation (ISAAC 2014)*, volume 8889 of *LNCS*, pages 553–564. Springer, 2014. doi:10.1007/978-3-319-13075-0_44.

3 Luis Barba, Matias Korman, Stefan Langerman, Rodrigo I. Silveira, and Kunihiko Sadakane. Space-time trade-offs for stack-based algorithms. In *Proc. 30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *LIPIcs*, pages 281–292. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPIcs.STACS.2013.281.

4 Paul Beame. A general sequential time-space tradeoff for finding unique elements. *SIAM J. Comput.*, 20(2):270–277, 1991. doi:10.1137/0220017.

5 Jon Bentley. *Programming Pearls*. ACM, New York, NY, USA, 1986.

6 Sankardeep Chakraborty, Anish Mukherjee, Venkatesh Raman, and Srinivasa Rao Satti. Frameworks for designing in-place graph algorithms. *CoRR*, abs/1711.09859, 2017. arXiv:1711.09859.

7 Samir Datta, Raghav Kulkarni, and Anish Mukherjee. Space-Efficient Approximation Scheme for Maximum Matching in Sparse Graphs. In *Proc. 41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58 of *LIPIcs*, pages 28:1–28:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.MFCS.2016.28.

8 Yevgeniy Dodis, Mihai Pătraşcu, and Mikkel Thorup. Changing base without losing space. In *Proc. 42nd ACM Symposium on Theory of Computing (STOC 2010)*, pages 593–602. ACM, 2010. doi:10.1145/1806689.1806770.

9 Amr Elmasry, Torben Hagerup, and Frank Kammer. Space-efficient basic graph algorithms. In *Proc. 32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *LIPIcs*, pages 288–301. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.288.

10 Amr Elmasry and Frank Kammer. Space-efficient plane-sweep algorithms. In *Proc. 27th International Symposium on Algorithms and Computation (ISAAC 2016)*, volume 64 of *LIPIcs*, pages 30:1–30:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ISAAC.2016.30.

11 Arash Farzan and J. Ian Munro. Succinct encoding of arbitrary graphs. *Theor. Comput. Sci.*, 513:38–52, 2013. doi:10.1016/j.tcs.2013.09.031.

12 Torben Hagerup and Frank Kammer. Succinct choice dictionaries. *Computing Research Repository (CoRR)*, arXiv:1604.06058 [cs.DS], 2016. arXiv:1604.06058.

13 Torben Hagerup and Frank Kammer. On-the-fly array initialization in less space. In *Proc. 28th International Symposium on Algorithms and Computation (ISAAC 2017)*, volume 92 of *LIPIcs*, pages 44:1–44:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ISAAC.2017.44.

14 Torben Hagerup, Frank Kammer, and Moritz Laudahn. Space-efficient Euler partition and bipartite edge coloring. *Theor. Comput. Sci., to appear*, 2018. doi:10.1016/j.tcs.2018.01.008.

15 Frank Kammer, Dieter Kratsch, and Moritz Laudahn. Space-Efficient Biconnected Components and Recognition of Outerplanar Graphs. In *Proc. 41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58 of *LIPIcs*, pages 56:1–56:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.MFCS.2016.56.

16 Frank Kammer and Andrej Sajenko. Linear-time in-place DFS and BFS in the restore model. *Computing Research Repository (CoRR)*, arXiv:1803.04282 [cs.DS], 2018. arXiv:1803.04282.

17 Takashi Katoh and Keisuke Goto. In-place initializable arrays. *Computing Research Repository (CoRR)*, arXiv:1709.08900 [cs.DS], 2017. arXiv:1709.08900.

18 Veli Mäkinen and Gonzalo Navarro. Dynamic entropy-compressed sequences and full-text indexes. *ACM Trans. Algorithms*, 4(3), 2008. doi:10.1145/1367064.1367072.

19 Kurt Mehlhorn. Data structures and algorithms 1: Sorting and searching. In *EATCS Monographs Theor. Comput. Sci.*, 1984.

20 J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct representations of permutations and functions. *Theor. Comput. Sci.*, 438:74–88, 2012. doi:10.1016/j.tcs.2012.03.005.

21 Gonzalo Navarro. Spaces, trees, and colors: The algorithmic landscape of document retrieval on sequences. *ACM Comput. Surv.*, 46(4):52:1–52:47, 2014. doi:10.1145/2535933.

22 Rasmus Pagh. Low redundancy in static dictionaries with constant query time. *SIAM J. Comput.*, 31(2):353–363, 2001. doi:10.1137/S0097539700369909.

23 Jakob Pagter and Theis Rauhe. Optimal time-space trade-offs for sorting. In *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1998)*, pages 264–268. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743455.

24 Mihai Pătraşcu. Succincter. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2008)*, pages 305–313. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.83.

25 Mihai Pătraşcu and Mikkel Thorup. Dynamic integer sets with optimal rank, select, and predecessor search. In *Proc. 55th IEEE Annual Symposium on Foundations of Computer Science, (FOCS 2014)*, pages 166–175. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.26.

Fast Entropy-Bounded String Dictionary Look-Up with Mismatches

Paweł Gawrychowski
University of Wrocław, Wrocław, 50-137, Poland
gawry@cs.uni.wroc.pl

Gad M. Landau
University of Haifa, Haifa, 3498838, Israel
landau@cs.haifa.ac.il

Tatiana Starikovskaya
DIENS, École normale supérieure, PSL Research University, Paris, 75005, France
tat.starikovskaya@gmail.com

Abstract

We revisit the fundamental problem of dictionary look-up with mismatches. Given a set (dictionary) of d strings of length m and an integer k, we must preprocess it into a data structure to answer the following queries: Given a query string Q of length m, find all strings in the dictionary that are at Hamming distance at most k from Q. Chan and Lewenstein (CPM 2015) showed a data structure for $k = 1$ with optimal query time $\mathcal{O}(m/w + occ)$, where w is the size of a machine word and occ is the size of the output. The data structure occupies $\mathcal{O}(wd \log^{1+\varepsilon} d)$ extra bits of space (beyond the entropy-bounded space required to store the dictionary strings). In this work we give a solution with similar bounds for a much wider range of values k. Namely, we give a data structure that has $\mathcal{O}(m/w + \log^k d + occ)$ query time and uses $\mathcal{O}(wd \log^k d)$ extra bits of space.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases Dictionary look-up, Hamming distance, compact data structures

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.66

Related Version A full version of the paper is available at `https://arxiv.org/abs/1806.09646`.

1 Introduction

The problem of dictionary look-up was introduced by Minsky and Papert in 1968 and is a fundamental task in many areas such as bioinformatics, information retrieval, and web search. Informally, the task is to store a set of strings referred to as dictionary in small space to maintain the following queries efficiently: Given a query string, return all dictionary strings that are close to it under some measure of distance. In this work we focus on Hamming distance and exact solutions to the problem. Formally, the problem is stated as follows.

Dictionary look-up with k mismatches. We are given a dictionary that is a set of d strings of length m and an integer $k > 0$. The task is to preprocess the dictionary into a data structure that maintains the following queries: Given a string P of length m, return all the strings in the dictionary such that the distance between each of them and P is at most k.

As a natural first step, much effort has been concentrated on the case $k = 1$ [3, 22, 4, 7, 8, 31, 11]. We note that the structure of the problem in this case is very special. Namely, if

two strings have Hamming distance at most one, then there is an integer i such that their prefixes of length i are equal and their suffixes of length $m-i-1$ are equal. Many existing solutions rely heavily on this property and cannot be extended to the case of arbitrary k. The first non-trivial solution for $k > 1$ was given in the seminal paper of Cole, Gottlieb, and Lewenstein [13], who introduced a data structure called *k-errata tree*. The k-errata tree requires $w \cdot \mathcal{O}(md + d\log^k d)$ bits of space and has query time $\mathcal{O}(m + \log^k d + occ)$, where w is the size of a machine word and occ is the size of the output. The subsequent work [10, 9, 25] mainly focused on improving the space complexity of the data structure.

Two works are of particular interest to us. For any $q = o(\log md)$ Belazzougui and Venturini [4] showed a data structure for $k = 1$ with query time $\mathcal{O}(m + occ)$ that uses $2mdH_q + o(md) + 2d\log d$ bits of space, where H_q is the q-th empirical entropy of the concatenation of all strings in the dictionary. It was followed by the work of Chan and Lewenstein [11], who improved the query time to $\mathcal{O}(m/w + occ)$, while using approximately the same amount of bits, $2mdH_q + o(md) + \mathcal{O}(wd\log^{1+\varepsilon} d)$. In the model of Chan and Lewenstein the size σ of the alphabet is constant, the query string arrives in a packed form, meaning that each $w/\log\sigma$ letters are stored in one machine word, under the standard assumption $w = \Theta(\log md)$. The interest in this kind of bounds is explained by the fact that the value mdH_q is a lower bound to the output size of any compressor that encodes each letter of the dictionary strings with a code that only depends on the letter itself and on the q immediately preceding letters [26].

1.1 Our contribution and techniques

We investigate further this line of research and give a new data structure with similar bounds for a much wider range of values k. We adopt the model of Chan and Lewenstein and show a data structure for dictionary look-up with k mismatches that has query time $\mathcal{O}(m/w + \log^k d + occ)$ and uses $2mdH_q + o(md) + \mathcal{O}(wd\log^k d)$ bits of space for all $d > 2$ (Theorem 18). If in addition $k \leq \log(m/w)/\log\log d$, the query time becomes $\mathcal{O}(m/w + occ)$, matching the query time of Chan and Lewenstein.

The basis of our data structure is the k-errata tree of Cole, Gottlieb, and Lewenstein [13]. We first introduce a small but important modification to this data structure that will allow us to reduce the time requirements for non-constant k. At a high level, the k-errata tree is a collection of compact tries, where each trie contains suffixes of a subset of strings in the dictionary. The query algorithm runs $\mathcal{O}(\log^k d)$ prefix search queries in the tries. In Section 4.1 we show that the prefix search queries can be implemented in $\mathcal{O}(m/w)$ shared plus $\mathcal{O}(\log d)$ time per query using $\mathcal{O}(md)$ space beyond the space required by the k-errata tree. Next, in Section 4.2 we show how to improve the space complexity to entropy-bounded. Our main contribution at this step is a new reduction from prefix search queries in the tries of the k-errata trees to prefix search queries on a compact trie containing only a subset of all suffixes of the dictionary strings. Finally, in Section 5 we improve the $\mathcal{O}(\log d)$ time that we spend per each query to $\mathcal{O}(1)$ (amortised) time by a clever use of Karp-Rabin fingerprints, which gives the final result, Theorem 18. We emphasize that we derandomize the query algorithm and that our data structure is deterministic, regardless the fact that we use Karp-Rabin fingerprints.

1.2 Related work

Many of the works we cited above consider not only the Hamming distance, but also the edit distance. This is in particular true for $k = 1$, when the edit distance and Hamming distance are equivalent. Another interesting direction is heuristic methods for the Hamming and the edit

distances which have worse theoretical guarantees but perform well in practice [12, 6, 23, 27]. Finally, we note that the solutions discussed in this work are beneficial for low-distance regime, i.e. when $k = o(\log d)$. If $k = \omega(\log d)$, one should turn to approximate approaches, such as locality-sensitive hashing (see [2] and references therein).

Several works have studied the question of developing efficient data structures for string processing when the query arrives in a packed form. In particular, Takuya et al. suggested a data structure called packed compact tries [29] to maintain efficient exact dictionary look-ups, and Bille, Gørtz, and Skjoldjensen used similar technique to develop an efficient text index [5].

2 Preliminaries

We assume a constant-size integer alphabet $\{1, 2, \ldots, \sigma\}$. A *string* is a sequence of letters of the alphabet. For a string $S = s_1 s_2 \ldots s_m$ we denote its length m by $|S|$ and its substring $s_i s_{i+1} \ldots s_j$, where $1 \leq i < j \leq m$, by $S[i, j]$. If $i = 1$, the substring $S[1, j]$ is referred to as *prefix* of S. If $j = m$, $S[i, m]$ is called a *suffix* of S. We say that S is given in a *packed form* if each $w/\log \sigma$ letters of S are stored in one machine word, i.e. S occupies $\mathcal{O}(m/w)$ machine words in total. Given a string S in packed form, we can access (a packed representation) of any $\mathcal{O}(w)$-length substring of S in constant time using the shift operation.

A *trie* is a basic data structure for storing a set of strings. A trie is a tree which has the following three properties:

1. Each edge is labelled by a letter of the alphabet;
2. Each two edges outgoing from the same node are labelled with different letters, and the edges are ordered by the letters;
3. Let the label of a node u be equal to the concatenation of the labels of the edges in the root-to-u path. For each string S in the set there is a node of the trie such that its label is equal to S, and the label of each node is equal to a prefix of some string in the set.

At each node we store the set of ids of the strings that are equal to the node's label. The number of nodes in a trie can be proportional to the total length of the strings. To improve the space requirements, we replace each path of nodes with degree one and with no string ids assigned to them with an edge labelled by the concatenation of the letters on the edges in the path. The result is called a *compact trie*. Each node of the trie is represented in the compact trie as well, some as nodes, and some as positions in the edges. We refer to the set of all nodes and the positions in the edges of the compact trie as positions.

▶ **Fact 1.** *A compact trie containing x strings has $\mathcal{O}(x)$ nodes.*

3 The k-errata tree: Reminder and fix

Our definition of the k-errata tree follows closely that of Cole, Gottlieb, and Lewenstein [13], but as explained below we introduce an important fix to the original definition. We try to be as concise as possible, but we feel obliged to provide all the details both because we modify the original definition and because the details are important for our final result.

Intuition. Let us explain the main idea first. Denote the given dictionary of strings by $\mathcal{D}$. The k-errata tree for $\mathcal{D}$ is built recursively. We start with the compact trie T containing all the strings in $\mathcal{D}$ and decompose it into heavy paths.

▶ **Definition 2** ([21]). The *heavy path* of T is the path that starts at the root of T and at each node v on the path branches to the child with the largest number of leaves in its subtree (*heavy* child), with ties broken arbitrarily. The heavy path decomposition is defined recursively, namely it is defined to be a union of the heavy path of T and the heavy path decompositions of the subtrees of T that hang off the heavy path. The first node in a heavy path is referred to as its *head*.

Recall that our task is to find all strings in $\mathcal{D}$ such that the Hamming distance between them and the query string P is at most k. As a first step, we find the longest path that starts at the root of T and is labelled with a prefix of P. Let this path trace heavy paths $H_1, H_2, \ldots, H_j$, leaving the heavy path H_i at a position u_i of T, $1 \leq i \leq j$. We can partition all the strings in $\mathcal{D}$ into three categories:

1. Strings diverging off a heavy path H_i at some node u, where u is located above u_i;
2. Strings in the subtrees of u_i's children that diverge from the heavy path H_{i+1}, for $1 \leq i < j$;
3. Strings in the subtree rooted at u_j.

Consider the set of strings in $\mathcal{D}$ that diverge from a heavy path H_i at a node u. They necessarily have their first mismatch with P there. The first idea is that we can fix that mismatch in each of the strings (decreasing the Hamming distance between them and P by one), and then run a dictionary look-up with $(k-1)$ mismatches on the resulting set of strings. The second idea is that running an independent dictionary look-up query for each node in each heavy path is expensive, so we introduce a grouping on the nodes that reduces the number of queries to logarithmic.

Data structure. We assign each string in $\mathcal{D}$ a credit of k mismatches and start building the k-errata tree in a recursive manner. First, we build the compact trie T for the dictionary $\mathcal{D}$. For each leaf of T we store the ids of the dictionary strings equal to the leaf's label ordered by the mismatch credits. Second, we decompose T into heavy paths. For each node of T we store a pointer to the heavy path it belongs to, and for each heavy path we store a pointer to its head. We will now make use of weight-balanced trees, defined in analogy with weight-balanced search trees.

▶ **Definition 3.** A weight-balanced tree with leaves of weights $w_1, w_2, \ldots, w_h$ (in left-to-right order) is a ternary tree. We build it recursively top-to-down. Let μ be the smallest index such that $w_1 + \cdots + w_\mu > (w_1 + \cdots + w_h)/2$. Then the left subtree hanging from the root is a weight-balanced tree with leaves of weight $w_1, w_2, \ldots, w_{\mu-1}$, the middle contains one leaf of weight w_μ, and the right subtree is a weight-balanced tree with leaves of weight $w_{\mu+1}, \ldots, w_h$.

We build two sets of $(k-1)$-errata trees for each heavy path H of T. We call the trees in the first set vertical, and in the second set horizontal, according to the way we construct them.

We first explain how we build the vertical $(k-1)$-errata trees. Suppose that H contains nodes $v_1, v_2, \ldots, v_h$, and the weight w_i of a node v_i is the number of strings that diverge from H at v_i. As a preliminary step, we build a weight-balanced tree $WBT(H)$ on the nodes in H. Consider a node of $WBT(H)$ containing $v_i, v_{i+1}, \ldots, v_j$ in its subtree. Let δ be the length of the string S written on the path from the head of H to v_j, and a be the first letter on the edge from v_j to v_{j+1}. We build a new set of strings as follows: For each node v_ℓ, $i \leq \ell \leq j$, we take each string that diverges from the path H at v_ℓ, cut off its prefix of length $\delta + 1$, and decrease the credit of the string by the number of mismatches between the cut-off

prefix and $S \circ a$ (the string S appended with the letter a). If the credit of a string becomes negative, we delete it. Finally, we build the $(k-1)$-errata tree for each of the newly created sets of strings.

We now explain how we build the horizontal $(k-1)$-errata trees. We repeat the following for each node $v_j \in H$. Let δ be the length of the label of v_j. Consider the set of all children of v_j except for the node v_{j+1} (the child of v_j that belongs to H). For each child v in this set, we build a new set of strings as follows: We take each string that ends below v, cut off its prefix of length $\delta + 1$, and decrease the credit of the string by 1. Similar to above, if the credit of a string becomes negative, we delete it. We define the weight of each child as the number of strings in the corresponding set. Next, we build the weight-balanced tree on the set of the children, and for each node of the tree consider a set of strings that is a union of the sets of strings below it. Finally, we build the $(k-1)$-errata tree for each of these sets of strings.

▶ Remark. Our modification to the original definition is that we truncate the strings and store the mismatch credits. Because of that, all the strings we work with are suffixes of the dictionary strings, which allows us to process them efficiently.

Queries. A dictionary look-up with k mismatches for a string P is performed in a recursive way as well. For the purposes of recursion, we introduce an extra parameter, μ, and allow to run dictionary look-ups with mismatches from any position u of a trie of the k-errata tree. We will make use of a procedure called PrefixSearch: Given a string and a position u of a trie, PrefixSearch returns the longest path starting at u that is labelled by a prefix of the query string.

Suppose we must answer a dictionary look-up with k mismatches for a string P that starts at a position u. We initialize $\mu = 0$. If $k = 0$, we run a PrefixSearch to find a path in T labelled by P. If such a path exists, we output all the dictionary strings assigned to the end of this path such that their mismatch credit $\geq \mu$. Assume now $k > 0$. If $|P| = 0$, the look-up terminates and we output all the dictionary strings assigned to the current position such that their mismatch credit $\geq \mu$. Otherwise, we run a PrefixSearch to find the longest path π starting at u that is labelled by a prefix of P. Suppose that π passes through heavy paths $H_1, H_2, \ldots, H_j$, leaving H_i at a position u_i, $1 \leq i \leq j$. Note that for $i < j$, u_i is necessarily a node of T, and for $i = j$ it can be a position on an edge.

Recall that for each node of T we store the heavy path it belongs to, and for each heavy path we store its head. The position u_j is the ending node of π. To find u_{j-1}, consider the heavy path H_j containing u_j, by definition, u_{j-1} is the parent of the head of H_j. We find all the nodes u_i, $1 \leq i \leq j$, analogously. Recall that we partitioned the dictionary strings into three types.

Strings of Type 1. We process each path H_i in turn. We select a set of nodes of the weight-balanced tree $WBT(H_i)$ covering the part of H_i from the beginning and up to (but not including) u_i. To do this, we follow the path from the root of $WBT(H_i)$ to u_i and take the nodes that hang off to the left of the path. Consider one of the selected nodes v and its $(k-1)$-errata tree. All the strings in this tree have equal lengths δ. To finish the recursive step, we run a dictionary look-up with $(k-1)$ mismatches for the suffix of P of length δ in this tree.

Strings of Type 2. We take the weight-balanced tree for u_i and select a set of nodes that covers all its leaves except for the head of H_{i+1}. To select this set, we find the path from the root of the weight-balanced tree for u_i to the head of H_{i+1}, and take the nodes that hang off this path. For each of the selected nodes, we run a dictionary look-up with $(k-1)$

mismatches analogously to above. We also run a dictionary look-up with $(k-1)$ mismatches with $\mu = \mu + 1$ starting from the position in H_i that is one letter below u_i.

Strings of Type 3. If u_j is a position on an edge, we run a dictionary look-up query with $(k-1)$ mismatches from the next position on the edge with $\mu = \mu + 1$. If u_j is a node, we run two dictionary look-up queries with $(k-1)$ mismatches. First query is run in the horizontal $(k-1)$-errata tree corresponding to the set of all children of u_j that are not in H_j. The second query is run from a position in H_j that is one letter below u_j with $\mu = \mu + 1$.

Correctness of the algorithm follows from the following observation: first, we account for all dictionary strings. Second, in the case of $(k-1)$-errata trees, we account for the mismatches between the portion of the strings that we truncate and the query string via the mismatch credits. Finally, when we continue the search in the same tree, there is just one mismatch and we account for it by increasing μ.

Analysis. The bounds on the space and the time complexities are summarised below. The proofs of the lemmas, which we provide in the full version of the paper for completeness, follow closely the proofs given by Cole, Gottlieb, and Lewenstein [13].

▶ **Lemma 4.** *The tries of the k-errata tree contain $\mathcal{O}(d \log^k d)$ strings in total.*

▶ **Lemma 5.** *A dictionary look-up with k mismatches for a query string P requires $\mathcal{O}(\log^k d)$ operations* PrefixSearch*. Apart from the time required for these operations, the algorithm spends $\mathcal{O}(\log^k d + occ)$ time.*

In the next section we give an efficient implementation of PrefixSearch under an assumption that P arrives in a packed form. We will use, in particular, the following simple observation.

▶ **Fact 6.** *Each trie of the k-errata tree is built on a set of equal-length suffixes of the dictionary strings. If we run a* PrefixSearch *for a suffix S of the query string P from a position u of a trie τ of the k-errata tree, then the strings in the subtree of u have length $|S|$.*

4 Prefix search for packed strings

We first remind several well-known data structure results that we use throughout the section. A *priority queue* is a data structure like a regular queue, where each element has an integer ("priority") associated with it. In a priority queue, an element with high priority is served before an element with low priority. A priority queue can be implemented as a heap, that for a set of x elements occupies $\mathcal{O}(wx)$ bits of space and has query time $\mathcal{O}(\log x)$.

A *predecessor data structure* on a set of integers supports the following queries: Given an integer z, return the largest integer in the set that is at most z. For a set of x integer keys, the predecessor data structure can be implemented as a binary search tree in $\mathcal{O}(wx)$ bits of space to support the predecessor queries in time $\mathcal{O}(\log x)$. (We do not use solutions such as [17, 30] to avoid dependency on m, which will be important for our final result.)

A *dictionary data structure* stores a set of integers. A dictionary look-up receives an integer z and outputs "yes" if z belongs to the set.

▶ **Lemma 7** ([28])**.** *Let S be any given set of x integers. There is a dictionary over S that occupies $\mathcal{O}(wx)$ bits of space, and has query time $\mathcal{O}(1)$.*

We will also need *lowest common ancestor queries* on tries. Given two nodes u, v of a trie, their lowest common ancestor is a node of maximal depth that contains both u and v in its subtree.

▶ **Lemma 8** ([18]). *A trie of size x can be preprocessed in $\mathcal{O}(wx)$ bits of space to maintain lowest common ancestor queries in $\mathcal{O}(1)$ time.*

Finally, we need *weighted level ancestor queries* on tries. A weighted level ancestor query receives a node u and an integer ℓ, and must output the deepest ancestor u' of u such that the length of the label of u' is at most ℓ. We will use the weighted level ancestor queries on tries for fast navigation: Suppose that we know a leaf labelled by a string S, then to find a position labelled by a prefix S' of S we can use one weighted level ancestor query instead of performing a PrefixSearch for S'. To avoid dependency on m, we use the following simple folklore solution instead of [20, 1, 14].

▶ **Lemma 9.** *A trie of size x can be preprocessed in $\mathcal{O}(wx)$ bits of space to maintain weighted level ancestor queries in $\mathcal{O}(\log x)$ time.*

Proof. We consider the heavy path decomposition of the trie. For each node we store a pointer to the head of the heavy path containing it, and for each path we build a binary search tree containing the length of the labels of the nodes in it. Suppose we are to answer a weighted level ancestor query for a node u and an integer ℓ. The path from the root of the trie to u (which contains all the ancestors of u) traverses a subset of heavy paths. The size of this subset is $\mathcal{O}(\log x)$, because each time we switch paths the weight of the current node decreases by at least a factor of two. We iterate over this set of paths to find the path that contains the answer u', and then use the binary search tree to find the location of u' in the path. Both steps take $\mathcal{O}(\log x)$ time. ◀

4.1 Linear space

As a warm-up we show a linear-space implementation of PrefixSearch that improves the runtime of dictionary look-up queries to $\mathcal{O}(m/w + \log^{k+1} d + occ)$. Formally, we will show the following result.

▶ **Theorem 10.** *Assume a constant-size alphabet. For a dictionary $\mathcal{D}$ of $d > 2$ strings of length m, there is a data structure for dictionary look-up with k mismatches that occupies $\mathcal{O}(wmd + wd \log^k d)$ bits of space and has query time $\mathcal{O}(m/w + \log^{k+1} d + occ)$, where $w = \Theta(\log md)$ is the size of a machine word.*

Let Suf be the set of all suffixes of the strings in $\mathcal{D}$. We build a compact trie $T(\text{Suf})$ on Suf. (In the literature, $T(\text{Suf})$ is referred to as the suffix tree of $\mathcal{D}$.) As the total length of the strings in $\mathcal{D}$ is md, the size of Suf is $\mathcal{O}(md)$, and therefore it occupies $\mathcal{O}(wmd)$ bits of space. We can reduce PrefixSearch queries on the tries of the k-errata tree to PrefixSearch queries on $T(\text{Suf})$. We distinguish between PrefixSearch queries that start at the root of some trie of the k-errata tree (*rooted* queries), and those that start at some inner node or even a position on an edge of a trie of the k-errata tree (*unrooted* queries). Note that unrooted queries are used in the case $k \geq 1$ only.

▶ **Lemma 11.** *After $\mathcal{O}(wmd + wd \log^k d)$ bits of space preprocessing, we can answer a rooted PrefixSearch query for a string Q and any trie of the k-errata tree in $\mathcal{O}(\log d)$ time given the answer to a rooted PrefixSearch for Q in $T(\text{Suf})$.*

▶ **Lemma 12.** *Assume $k \geq 1$. After $\mathcal{O}(wmd + wd \log^k d)$ bits of space preprocessing, we can reduce an unrooted PrefixSearch query for Q that starts at a position u of a trie τ of the k-errata tree to a rooted PrefixSearch for some suffix Q' of Q in a trie τ' of a $(k-1)$-errata tree in $\mathcal{O}(\log d)$ time given the answer to a rooted PrefixSearch query for Q in $T(\text{Suf})$.*

Lemmas 11 and 12 were proved in [13]. For completeness, we give their proofs in the full version of the paper. Suppose we are to answer a dictionary look-up with k mismatches for a string P. Our algorithm traverses the k-errata tree and generates rooted and unrooted PrefixSearch queries. We maintain a priority queue. Each time we need an answer to a PrefixSearch for a string S in $T(\mathrm{Suf})$, we add S to the priority queue. At each step of the algorithm we extract the longest string from the queue and answer the PrefixSearch query for it. Notice that all strings in the queue are suffixes of P and that the maximal length of strings in the queue cannot increase. We can therefore assume that we must answer PrefixSearch queries for the suffixes of P starting at positions $1 = i_1 \leq i_2 \leq \cdots \leq i_z$, where $z = \mathcal{O}(\log^k d)$.

Bille, Gørtz, and Skjoldjensen [5] showed that we can preprocess $T(\mathrm{Suf})$ in linear space to answer PrefixSearch queries for a single query string of length m in $\mathcal{O}(m/w + \log\log md)$ time. As an immediate corollary we obtain that we can answer z PrefixSearch queries in $z \cdot \mathcal{O}(m/w + \log\log md)$ time, but this is too slow for our purposes. Below we develop their ideas to give a more efficient approach.

▶ **Lemma 13.** *$T(\mathrm{Suf})$ can be preprocessed in $\mathcal{O}(wmd)$ bits of space to answer PrefixSearch for the suffixes of P starting at positions $1 = i_1 \leq i_2 \leq \cdots \leq i_z$ in $\mathcal{O}(m/w + z\log d)$ time.*

Proof. We assume that the strings in the dictionary are stored in the packed form. By construction, each edge of a trie of the k-errata is labelled by a substring of a dictionary string. It means that we can store each label as three integers: the id of the string, and the starting and the ending positions of the substring. Next, we preprocess $T(\mathrm{Suf})$ for weighted level ancestor queries (Lemma 9). A node or a position in the trie is called *boundary* if the length of its label is a multiple of $w/\log\sigma$, where w is the size of a machine word and σ is the size of the alphabet. Boundary nodes cut the tree into micro-trees. We only consider the micro-trees containing more than two nodes. We define the label of a leaf of a micro-tree as a machine word that contains a packed representation of the string written on the path from the root of the micro-tree to the leaf. The labels can be treated as integers; for each micro-tree we create a dictionary (Lemma 7) and a predecessor data structure on the labels of its leaves. We also preprocess each micro-tree for lowest common ancestors. Note that the total size of the micro-trees is $\mathcal{O}(md)$, as each edge of $T(\mathrm{Suf})$ contains at most two nodes of the micro-trees. Therefore, the preprocessing requires $\mathcal{O}(wmd)$ bits of space.

We now explain how to answer the PrefixSearch queries for the suffixes of P starting at the positions $1 = i_1 \leq i_2 \leq \cdots \leq i_z$. For i_1, we start at the root of $T(\mathrm{Suf})$. For i_j, $j > 1$, we use the information obtained at the previous step. Namely, suppose that the PrefixSearch for $P[i_{j-1}, m]$ terminated at a position labelled by $P[i_{j-1}, \ell_{j-1}]$. We take any leaf below this position, let it be labelled by a string $S \in \mathrm{Suf}$. Let $P[i_j, \ell'_{j-1}]$ be the longest prefix of $P[i_j, \ell_{j-1}]$ such that its length is a multiple of $w/\log\sigma$. We then start the PrefixSearch from a position u labelled by $P[i_j, \ell'_{j-1}]$. To find the position u, we first find the leaf labelled by $S[i_j - i_{j-1} + 1, |S|] \in \mathrm{Suf}$, and then use a weighted level ancestor query to jump to u in $\mathcal{O}(\log d)$ time. Notice that u is boundary. If u is not a root of a micro-tree, it has a single outgoing edge of length at least $w/\log\sigma$. We compare the first $w/\log\sigma$ letters of the label of this edge and $P[\ell'_{j-1} + 1, \ell'_{j-1} + w/\log\sigma]$ in $\mathcal{O}(1)$ time by comparing the corresponding machine words. If they are equal, we continue from the next boundary node on the edge in a similar manner. Otherwise, we find the first mismatch between the two strings in $\mathcal{O}(1)$ time as follows: First, compute a bitwise XOR of the two strings, and then locate the most significant bit using the technique of [19].

If u is the root of a micro-tree τ, we search for $P[\ell'_{j-1}+1, \ell'_{j-1}+w/\log\sigma]$ in the dictionary of τ. If it is in the dictionary and corresponds to a leaf v, we continue to v. Otherwise, we find its predecessor pred and successor succ using the predecessor data structure. The

PrefixSearch must terminate either on the path from u to the leaf of the micro-tree labelled by pred, or on the path from u to the leaf of the micro-tree labelled by succ. We compute the longest common prefix of $P[\ell'_{j-1}+1, \ell'_{j-1}+w/\log\sigma]$ with pred and with succ using bitvector operations in $\mathcal{O}(1)$ time as explained above, take the longest of the two, and find the position labelled by it in $\mathcal{O}(\log d)$ time using a weighted level ancestor query.

The running time of each prefix search query is proportional to the number of $(w/\log\sigma)$-length blocks of P that we compare with the labels of the edges of $T(\mathrm{Suf})$. Notice that each two different PrefixSearch queries share at most one block of letters. Therefore, as the size of the alphabet σ is constant, the total running time of z PrefixSearch queries is $\mathcal{O}(m/w + z\log d)$. ◀

Lemmas 4, 5, 11, 12, and 13 give Theorem 10.

4.2 Entropy-bounded space

In this section we improve the space requirements of our implementation of PrefixSearch and show the following theorem.

▶ **Theorem 14.** *Assume a constant-size alphabet. For a dictionary of $d > 2$ strings of lengths m and any $q = o(\log md)$, let H_q be the q-th empirical entropy of the concatenation of all the dictionary strings. There is a data structure for dictionary look-ups with k mismatches that uses $mdH_q + o(md) + \mathcal{O}(wd\log^k d)$ bits of space and has query time $\mathcal{O}(m/w + \log^{k+1} d + occ)$, where $w = \Theta(\log md)$ is the size of a machine word.*

There are two bottlenecks: First, we need to store the dictionary strings, and second, the tree structure of $T(\mathrm{Suf})$ requires $\Omega(md)$ space. To overcome the first bottleneck, we replace the packed representation of the dictionary strings by the Ferragina-Venturini representation:

▶ **Lemma 15** ([16])**.** *Under the assumption of an alphabet of constant size σ, for any $q = o(\log md)$ there exists a data structure that uses $mdH_q + o(md)$ bits of space and supports constant-time access to any $w/\log\sigma = \Theta(\log md)$-length substring of a dictionary string.*

If $d > 2$ is a constant, if suffices to store the Ferragina-Manzini representation of the dictionary strings to obtain the bounds of Theorem 14. Indeed, when a query string P arrives, we can decide if the Hamming distance between P and a dictionary string is at most k in $\mathcal{O}(m/w + k) = \mathcal{O}(m/w + \log^k d)$ time, using comparison by machine words and bitvector operations. As d is constant, we obtain the desired time bound. Below we assume that $d = \Omega(1)$.

We now deal with the second bottleneck. We will consider a smaller trie $T(\mathrm{Suf}')$ on a subset Suf' of Suf, and will show that PrefixSearch queries on tries of the k-errata tree can be reduced to PrefixSearch queries on this trie. Suf' is defined to be the set of all suffixes of the dictionary strings that start at positions $w\log d/\log\sigma$, $2w\log d/\log\sigma$, and so on. We call such suffixes *sampled*. Below we show that we can reduce PrefixSearch queries in the tries of the k-errata tree to PrefixSearch queries in $T(\mathrm{Suf}')$.

▶ **Lemma 16.** *After $\mathcal{O}(md/\log d + wd\log^k d)$ bits of space preprocessing, we can answer a rooted PrefixSearch query for a string $Q = P[\ell, m]$ in $\mathcal{O}(\log d)$ time given the answer to a rooted PrefixSearch query for a string $P[\ell', m]$ in $T(\mathrm{Suf}')$, where $\ell' \geq \ell$ is the smallest multiple of $w\log d/\log\sigma$.*

Proof. At the preprocessing step, we traverse $T(\mathrm{Suf}')$ and remember the leftmost and the rightmost leaves in each of its subtrees. We also remember the neighbours of each leaf in the left-to-right order, and finally we preprocess the trie for lowest common ancestor queries. As

a second step we preprocess each trie of the k-errata tree for lowest common ancestor and weighted level ancestor queries. We also build the following data structure for each trie of the k-errata tree. For each string S in the trie, let $S = pS'$, where S' is the longest sampled suffix of S. We call p a head of S, and define the rank of S to be the rank of S' in Suf'. We build a compact trie T_{heads} containing the heads of all the strings, and preprocess it as in Lemma 13. If T_{heads} contains x strings, we use $\mathcal{O}(wx)$ bits of space for the preprocessing, i.e. $\mathcal{O}(wd \log^k d)$ bits of space in total. We also associate a predecessor data structure with each of its leaves. The predecessor data structure of a leaf labelled by p contains the ranks of all the strings such that their head is equal to p. The predecessor data structures occupy $\mathcal{O}(wd \log^k d)$ bits of space in total as well.

Suppose we are to answer a rooted PrefixSearch query for a string $Q = P[\ell, m]$ and a trie τ of the k-errata tree. Let T_{heads} be the compact trie containing the heads of the strings in τ. By Fact 6, the length of the heads is $(\ell' - \ell)$. We first read $P[\ell, \ell' - 1]$ in blocks of $w/\log\sigma$ letters in $\mathcal{O}(\log d)$ time, and run a PrefixSearch for it in T_{heads} in $\mathcal{O}(\log d)$ time. If the PrefixSearch terminates in a position u of T_{heads} that is not in a leaf, it remains to find the position corresponding to u in τ, which we can do with one weighted level ancestor query.

Assume now that the PrefixSearch terminates in a leaf of T_{heads}. By the condition of the lemma, we know the answer to the rooted PrefixSearch for $P[\ell', m]$ in $T(\mathrm{Suf}')$. We also store the leftmost and the rightmost leaves in each subtree of $T(\mathrm{Suf}')$, and therefore can find the predecessor of $P[\ell', m]$ in Suf' in $\mathcal{O}(1)$ time. We use the predecessor data structure associated with the leaf to find the predecessor pred and successor succ of $P[\ell', m]$ in $\mathcal{O}(\log d)$ time. To find the position where the PrefixSearch for P terminates, we compute the lengths ℓ_p, ℓ_s of the longest common prefix of $P[\ell', m]$ and pred and of $P[\ell', m]$ and succ. We can compute the longest common prefix of $P[\ell', m]$ and pred (which is a sampled suffix of a dictionary string) in $\mathcal{O}(1)$ time via a lowest common ancestor query on $T(\mathrm{Suf}')$. We then compute ℓ_s in a similar way. If $\ell_p = \ell_s$, we return the lowest common ancestor of pred and succ as the answer. If $\ell_p > \ell_s$, then the answer is the ancestor of $P[\ell, \ell' - 1] \circ \mathsf{pred}$ such that the length of its label is $(\ell' - \ell) + \ell_p$, and we can find it by one weighted level ancestor query. The case $\ell_s > \ell_p$ is analogous. ◀

▶ **Lemma 17.** *Assume $k \geq 1$. After $\mathcal{O}(md/\log d + wd \log^k d)$ bits of space preprocessing, we can answer an unrooted PrefixSearch query for a string $Q = P[\ell, m]$ by reducing it to a rooted PrefixSearch query in $\mathcal{O}(\log d)$ time given the answer to a rooted PrefixSearch query for a string $P[\ell', m]$ in $T(\mathrm{Suf}')$, where $\ell' \geq \ell$ is the smallest multiple of $w \log d/\log\sigma$.*

Proof. During the preprocessing step, we preprocess $T(\mathrm{Suf}')$ for lowest common ancestor queries and each trie of the k-errata tree for weighted level ancestor queries. Let u be the position in a trie τ where we start the PrefixSearch for $P[\ell, m]$. The search path for $P[\ell, m]$ traverses a number of heavy paths. The first path is the path containing u. Let S be the label of the part of the path starting from u. We consider two cases. Suppose first that $\ell' = \ell$. In this case, S is a suffix of one of the dictionary strings starting at a position ℓ', i.e. it is sampled. Therefore, we can find the longest common prefix of $P[\ell', m]$ and S using one lowest common ancestor query on $T(\mathrm{Suf}')$. We can then find the node in the path corresponding to this longest common prefix using one weighted level ancestor query. From there, we can find the starting node of the second heavy path traversed by $P[\ell, m]$ in $\mathcal{O}(1)$ time. It remains to answer a rooted PrefixSearch query in the subtree rooted at this node, which is a trie of a $(k-1)$-errata tree by construction. When we know the answer for this PrefixSearch, we can go back to τ using one weighted level ancestor query. In the second case $\ell' > \ell$. We start by comparing $P[\ell + 1, m]$ and S by blocks of $w/\log\sigma$ letters until we reach the start of a sampled suffix, and then proceed as above. ◀

Suppose that we are to answer a dictionary look-up with k mismatches for a string P. Our algorithm traverses the k-errata tree and generates rooted PrefixSearch queries for the suffixes of P in $T(\mathrm{Suf}')$. We maintain a priority queue. Each time we need an answer to a rooted PrefixSearch for a suffix $Q = P[i, m]$ in $T(\mathrm{Suf}')$, we add Q to the priority queue. At each step we extract the longest string from the queue and answer the PrefixSearch query for it. Since the maximal length of suffixes in the queue cannot increase, we can assume that we must answer PrefixSearch queries for the suffixes of P starting at positions $i_1 \le i_2 \le \cdots \le i_z$, where $z = \mathcal{O}(\log^k d)$. Moreover, for each j the position i_j is a multiple of $w \log d / \log \sigma$. We preprocess $T(\mathrm{Suf}')$ as in Lemma 13, which requires $\mathcal{O}(md/\log d) = o(md)$ bits of space. We first run PrefixSearch for $P[i_1, m]$ in $\mathcal{O}(m/w)$ time. Suppose it follows the path labelled by $P[i_1, \ell_1]$. Let $S \in \mathrm{Suf}'$ be an arbitrary string that ends below the end of this path. We then find the leaf corresponding to $S[i_2 - i_1, |S|]$. By construction of $T(\mathrm{Suf}')$ and because $i_2 - i_1$ is a multiple of $w \log d / \log \sigma$, such a leaf must exist. We then use a weighted level ancestor query to find the end of the path labelled by $P[i_2, \ell_1']$, where $P[i_2, \ell_1']$ is the longest suffix of $P[i_2, \ell_1]$ such that its length is a multiple of $w/\log\sigma$, and continue the PrefixSearch for $P[i_2, m]$ from there, and so on. The total running time is $\mathcal{O}(m/w + z \log d) = \mathcal{O}(m/w + \log^{k+1} d)$.

If $d = \Omega(1)$ as we assumed earlier, lemmas 15, 16, 17, and the discussion above give Theorem 14.

5 Removing extra logarithm from the time complexity

In this section we improve the query time to $\mathcal{O}(m/w + \log^k d + occ)$ and show our final result.

▶ **Theorem 18.** *Assume a constant-size alphabet. For a dictionary of $d > 2$ strings of lengths m and for any $q = o(\log md)$, let H_q be the q-th empirical entropy of the concatenation of all strings in the dictionary. There exists a data structure for dictionary look-ups with k mismatches that uses $2mdH_q + o(md) + \mathcal{O}(wd \log^k d)$ bits of space and has query time $\mathcal{O}(m/w + \log^k d + occ)$, where $w = \Theta(\log md)$ is the size of a machine word.*

As explained in Theorem 14, we can assume $d = \Omega(1)$. Recall that the dictionary look-up with k mismatches is run recursively. The first $(k-2)$ levels of recursion require $\mathcal{O}(\log^{k-2} d)$ PrefixSearch queries and can be implemented in $\mathcal{O}(m/w + \log^{k-1} d)$ time. Therefore, it suffices to improve the runtime of the two last levels of the recursion, where we must perform a batch $\mathcal{O}(\log^{k-1} d)$ dictionary look-up queries with one mismatch. To achieve the desired complexity we use the fact that the queries are related, as explained below.

Preprocessing. For a string $S = s_1 s_2 \dots s_m$ we define its reverse $S^R = s_m \dots s_2 s_1$. First, we build a compact trie on the reverses of all the dictionary strings and preprocess it as described in Lemma 13, which takes $\mathcal{O}(wd)$ bits of space. We store the reverses using the Ferragina-Venturini representation (Lemma 15) in $H_q' md + o(md)$ bits of space, where H_q' is the q-th empirical entropy of the reverse of the concatenation of all the strings in the dictionary. By [15, Theorem A.3], $H_q' md + o(md) = H_q md + o(md)$. For the second step, we need Karp-Rabin fingerprints. We modify the standard definition as we work with packed strings.

▶ **Definition 19** (Karp-Rabin fingerprints [24])**.** Consider a string S and its packed representation $w_1 w_2 \dots w_z$, where each w_i is a machine word. (If $|S|$ is not a multiple of $w/\log\sigma$, we append an appropriate number of zeros.) The Karp–Rabin fingerprint of S is defined as $\varphi = \sum_{i=1}^{z} w_i \cdot r^{z-i} \bmod p$, where p is a fixed prime number and r is a randomly chosen integer in $[0, p-1]$.

From the definition it follows that if the strings are equal, their fingerprints are equal. Furthermore, it is well-known that for any $c > 3$ and $p > (\max\{m/w, d\log^k d\})^c$, the probability of two distinct strings of length $zw \leq \max\{m, wd\log^k d\}$ having the same fingerprint (*collision probability*) is less than $1/(\max\{m/w, d\log^k d\})^{c-1}$. Consider a trie τ of the k-errata tree. By definition, the lengths of the leaf labels in τ is at most m. From the bound on the collision probability it follows that we can choose p and r so that the fingerprints of the reverses of these labels are distinct. For each leaf of τ, we compute the Karp-Rabin fingerprint of the reverse of its label and add it to a dictionary (Lemma 7) associated with τ. Also, using the same p and r, we compute Karp-Rabin fingerprints corresponding to inner nodes of the tries of the k-errata tree. Namely, consider one of such nodes, and let S be its label and δ be the length of the strings in the trie. We take the reverse of S, prepend it with $(\delta - |S|) \bmod w/\log\sigma$ zeros, and compute the Karp-Rabin fingerprint of the resulting string.

Queries. We must run $\mathcal{O}(\log^{k-1} d)$ dictionary look-up queries with one mismatch. Consider one of these queries, let it be a query for a string Q (which must be a suffix of P) in a trie τ and recall the algorithm of Section 3. First, we run a PrefixSearch to find the longest path π that is labelled by a prefix of Q. For this step we can use $T(\mathrm{Suf}')$, as the total number of such queries is $\mathcal{O}(\log^{k-1} d)$ and therefore we can spend $\mathcal{O}(\log d)$ time per each of them. Suppose that π traverses the heavy paths $H_1, H_2, \ldots, H_j$ and leaves the heavy path H_i at a position u_i. We can find the positions u_j in $\mathcal{O}(\log d)$ time once we have found the end of π. The rest of the algorithm can be described as follows. First, we must perform dictionary look-ups with 0 mismatches (i.e., PrefixSearch) in $\mathcal{O}(\log d)$ vertical and $\mathcal{O}(\log d)$ horizontal 0-errata trees (that are tries of the k-errata tree by definition). Second, for each $1 \leq i < j$, we must perform a dictionary look-up with 0 mismatches (PrefixSearch) from a position u'_i that follows u_i in the heavy path H_i. Importantly, each u_i is a node. Finally, we must perform a a dictionary look-up with 0 mismatches (PrefixSearch) from a position u'_j that follows u_j in the heavy path H_j.

We note that to perform the PrefixSearch from the position u'_j we can use $T(\mathrm{Suf}')$, as before, because the total number of such PrefixSearch operations is $\mathcal{O}(\log^{k-1} d)$. We now explain how we perform the PrefixSearch operations in vertical and horizontal 0-errata trees, as well as the PrefixSearch operations from nodes u'_i, $1 \leq i < j$. In total, we must perform $\mathcal{O}(\log^k d)$ such operations, and for each of them the query string is a suffix of P. Let $P[i_1, m], P[i_2, m], \ldots, P[i_z, m]$, $z = \mathcal{O}(\log^k d)$ be the suffixes of P for which we are to run a PrefixSearch. We create a bitvector of length $m = o(md)$ where each i_j^{th} bit is set. We then compute the Karp-Rabin fingerprints of the reverses of $P[i_1, m], P[i_2, m], \ldots, P[i_z, m]$ in $\mathcal{O}(m/w + z)$ time using the following fact.

▶ **Fact 20.** *Given the Karp-Rabin fingerprints of X and Y, where the length of X is a multiple of $w/\log\sigma$, we can compute the Karp-Rabin fingerprint of their concatenation, XY in $\mathcal{O}(1)$ time.*

We iterate over all blocks of the bitvector starting from the last one and maintain the Karp-Rabin fingerprint of the reverse of the suffix of P that starts at the current position. When we start a new block, we update the Karp-Rabin fingerprint. If a block contains set bits (which we can decide in constant time), we extract the positions of all set bits in $\mathcal{O}(1)$ time per bit using the technique of [19], and compute the corresponding Karp-Rabin fingerprints. Also, as a preliminary step, we run a PrefixSearch for P^R in the compact trie on the reverses of the dictionary strings in $\mathcal{O}(m/w + \log d)$ time. Let u be the position where this PrefixSearch terminates.

PrefixSearch in vertical and horizontal 0-errata trees. Assume we must answer a PrefixSearch for $P[i_j, m]$ on a tree τ. We search the fingerprint of the reverse of $P[i_j, m]$ in the dictionary associated with τ. The search will return at most one leaf of the tree. We know that its label is equal to $P[i_j, m]$ with high probability, but we need a deterministic answer. We test the leaf as follows. Let S be one of the dictionary strings such that its id is stored at the leaf. We find the leaf v of the compact tree on the reverses of the dictionary strings that corresponds to the reverse S^R of S. Now, we can compute the length of the longest common prefix of P^R and S^R in constant time via a lowest common ancestor query for u and v and check if it is indeed equal or larger than $|P[i_j, m]|$.

PrefixSearch from u'_i, $1 \leq i < j$. This step is equivalent to the following: Find all the strings in the trie that start with a label of u'_i and end with a given suffix of P. We can compute the Karp-Rabin fingerprints of the reverses of the strings that we are looking for as follows. Positions u_i are necessarily nodes and we store the Karp-Rabin fingerprints of the reverses of their labels. Recall that if S_i was the label of u_i, we prepended the reverse S_i^R of S_i with $(\delta - |S|) \bmod w/\log\sigma$ zeros, where δ is the length of the strings in the trie containing u_i. It follows that we can compute the fingerprint φ_i of the reverse of the label of u'_i prepended with $(\delta - |S_i| - 1) \bmod w/\log\sigma$ zeros in $\mathcal{O}(1)$ time. Knowing φ_i and the fingerprint of the reverse of the suffix of P, we can compute the fingerprint of the strings we are searching for in constant time. We note that prepending with zeros is necessary in order to align the borders of the blocks in the reverse of the label of u'_i and the reverse of the label of the suffix of P. We finish the computation as above, that is we find a leaf such that the fingerprint of the reverse of its label is equal to the fingerprint of the strings we are looking for, and test it using the trie on the reverses of the dictionary strings.

References

1. Amihood Amir, Gad M. Landau, Moshe Lewenstein, and Dina Sokol. Dynamic text and static pattern matching. *ACM Trans. Algorithms*, 3(2), 2007.
2. Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proc. of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC'15, pages 793–801, 2015.
3. Djamal Belazzougui. Faster and space-optimal edit distance "1" dictionary. In *Proc. of the Annual Symposium on Combinatorial Pattern Matching*, CPM'09, pages 154–167, 2009.
4. Djamal Belazzougui and Rossano Venturini. Compressed string dictionary look-up with edit distance one. In *Proc. of the Annual Symposium on Combinatorial Pattern Matching*, CPM'12, pages 280–292, 2012.
5. Philip Bille, Inge Li Gørtz, and Frederik Rye Skjoldjensen. Deterministic indexing for packed strings. In *Proc. of the Annual Symposium on Combinatorial Pattern Matching*, CPM'17, pages 6:1–6:11, 2017.
6. Thomas Bocek, Ela Hunt, Burkhard Stiller, and Fabio Hecht. Fast similarity search in large dictionaries. Technical Report ifi-2007.02, Department of Informatics, University of Zurich, 2007.
7. Gerth Stølting Brodal and Leszek Gasieniec. Approximate dictionary queries. In *Proc. of the Annual Symposium on Combinatorial Pattern Matching*, CPM'96, pages 65–74, 1996.
8. Gerth Stølting Brodal and Srinivasan Venkatesh. Improved bounds for dictionary look-up with one error. *Inf. Process. Lett.*, 75:57–59, 2000.
9. Ho-Leung Chan, Tak Wah Lam, Wing-Kin Sung, Siu-Lung Tam, and Swee-Seong Wong. Compressed indexes for approximate string matching. *J. Algorithmica*, 58:263–281, 2006.

10 Ho-Leung Chan, Tak Wah Lam, Wing-Kin Sung, Siu-Lung Tam, and Swee-Seong Wong. A linear size index for approximate pattern matching. In *Proc. of the Annual Symposium on Combinatorial Pattern Matching*, CPM'06, pages 45–59, 2006.

11 Timothy Chan and Moshe Lewenstein. Fast string dictionary lookup with one error. In *Proc. of the Annual Symposium on Combinatorial Pattern Matching*, CPM'15, pages 114–123, 2015.

12 Aleksander Cisłak and Szymon Grabowski. A practical index for approximate dictionary matching with few mismatches. *Computing & Informatics*, 36(5):1088–1106, 2017.

13 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In *Proc. of the 36th Annual ACM Symposium on Theory of Computing*, STOC'04, pages 91–100, 2004.

14 Martin Farach and S. Muthukrishnan. Perfect hashing for strings: Formalization and algorithms. In *Proc. of the Annual Symposium on Combinatorial Pattern Matching*, CPM'96, pages 130–140, 1996.

15 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005.

16 Paolo Ferragina and Rossano Venturini. A simple storage scheme for strings achieving entropy bounds. *Theoretical Computer Science*, 372(1):115–121, 2007.

17 Johannes Fischer and Pawel Gawrychowski. Alphabet-dependent string searching with wexponential search trees. In *Proc. of the Annual Symposium on Combinatorial Pattern Matching*, CPM'15, pages 160–171, 2015.

18 Johannes Fischer and Volker Heun. Theoretical and practical improvements on the RMQ-problem, with applications to LCA and LCE. In *Proc. of the Annual Conference on Combinatorial Pattern Matching*, CPM'06, pages 36–48, 2006.

19 Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, dec 1993.

20 Pawel Gawrychowski, Moshe Lewenstein, and Patrick K. Nicholson. Weighted ancestors in suffix trees. In *Proc. of the Annual European Symposium on Algorithms*, ESA'14, pages 455–466, 2014.

21 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

22 Wing-Kai Hon, Tsung-Han Ku, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Compressed dictionary matching with one error. In *Proc. of the Data Compression Conference*, DCC'11, pages 113–122, 2011.

23 Daniel Karch, Dennis Luxen, and Peter Sanders. Improved fast similarity search in dictionaries. In *Proc. of the International Symposium on String Processing and Information Retrieval*, SPIRE'10, pages 173–178, 2010.

24 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, 1987.

25 Tak Wah Lam, Wing-Kin Sung, and Swee-Seong Wong. Improved approximate string matching using compressed suffix data structures. *J. Algorithmica*, 51:298–314, 2005.

26 Giovanni Manzini. An analysis of the Burrows -Wheeler transform. *J. ACM*, 48(3):407–430, may 2001.

27 Moshe Mor and Aviezri S. Fraenkel. A hash code method for detecting and correcting spelling errors. *Commun. ACM*, 25(12):935–938, 1982.

28 Milan Ružić. Uniform deterministic dictionaries. *ACM Trans. Algorithms*, 4(1):1:1–1:23, mar 2008.

29 Takuya Takagi, Shunsuke Inenaga, Kunihiko Sadakane, and Hiroki Arimura. Packed compact tries: A fast and efficient data structure for online string processing. In *Proc. of*

the 27th International Workshop on Combinatorial Algorithms, volume 9843 of *IWOCA'16*, pages 213–225. Springer, 2016.

30 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space o(n). *Information Processing Letters*, 17(2):81–84, 1983.

31 Andrew Chi-Chih Yao and Foong Frances Yao. Dictionary look-up with one error. *J. Algorithms*, 25:194–202, 1997.

New Results on Directed Edge Dominating Set

Rémy Belmonte
University of Electro-Communications, Chofu, Tokyo, 182-8585, Japan

Tesshu Hanaka
Department of Information and System Engineering, Chuo University, Tokyo, Japan

Ioannis Katsikarelis
Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243
LAMSADE, 75016, Paris, France

Eun Jung Kim[1]
Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243
LAMSADE, 75016, Paris, France

Michael Lampis[2]
Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243
LAMSADE, 75016, Paris, France

Abstract

We study a family of generalizations of Edge Dominating Set on directed graphs called Directed (p, q)-Edge Dominating Set. In this problem an arc (u, v) is said to dominate itself, as well as all arcs which are at distance at most q *from* v, or at distance at most p *to* u.

First, we give significantly improved FPT algorithms for the two most important cases of the problem, $(0, 1)$-dEDS and $(1, 1)$-dEDS (that correspond to versions of Dominating Set on line graphs), as well as polynomial kernels. We also improve the best-known approximation for these cases from logarithmic to constant. In addition, we show that (p, q)-dEDS is FPT parameterized by $p + q + \mathrm{tw}$, but W-hard parameterized just by tw, where tw is the treewidth of the underlying graph of the input.

We then go on to focus on the complexity of the problem on tournaments. Here, we provide a complete classification for every possible fixed value of p, q, which shows that the problem exhibits a surprising behavior, including cases which are in P; cases which are solvable in quasi-polynomial time but not in P; and a single case ($p = q = 1$) which is NP-hard (under randomized reductions) and cannot be solved in sub-exponential time, under standard assumptions.

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

Keywords and phrases Edge Dominating Set, Tournaments, Treewidth

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.67

Funding Partially supported by JSPS and MAEDI under the Japan-France Integrated Action Program (SAKURA), Project GRAPA 38593YJ.

[1] The author was partially supported by the ANR grant "ESIGMA" (ANR-17-CE40-0028).
[2] The author was partially supported by the ANR grant "ESIGMA" (ANR-17-CE40-0028).

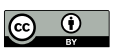


Table 1 Complexity status for various values of p and q: on general digraphs.

Param.	p, q	FPT / W-hard	Kernel	Approximability
k	$p+q \leq 1$	$2^{O(k)}$ [22] $\rightarrow 2^k$ [Thm.3]	$O(k)$ vertices [Thm.8]	3-apprx [Thm.4]
	$p=q=1$	$2^{O(k)}$ [22] $\rightarrow 9^k$ [Thm.2]	$O(k^2)$ vertices [Thm.7]	8-apprx [Thm.5]
	$\max\{p,q\} \geq 2$	W[2]-hard [22]	-	no $o(\ln k)$-approx [22]
tw	any p, q	W[1]-hard [Thm.11]	-	-
tw+p+q	any p, q	FPT [Thm.12]	unknown	-

1 Introduction

Edge Dominating Set (EDS) is a classical graph problem, equivalent to Minimum Dominating Set on line graphs. Despite the problem's prominence, EDS has until recently received very little attention in the context of directed graphs. In this paper we investigate the complexity of a family of natural generalizations of this classical problem to digraphs, building upon recent work [22].

One of the reasons that EDS has not so far been well studied in digraphs is that there are several natural ways in which the undirected version can be generalized. For example, seeing as EDS is exactly Dominating Set in line graphs, one could define Directed EDS as (Directed) Dominating Set in line digraphs [23]. In this formulation, an arc (u, v) dominates all arcs (v, w); however (v, w) does not dominate (u, v). Another natural way to define the problem would be to consider Dominating Set on the underlying graph of the line digraph, so as to maximize the symmetry of the problem, while still taking into account the directions of arcs. In this formulation, (u, v) dominates arcs coming out of v and arcs coming into u, but not other arcs incident on u, v.

A unifying framework for studying such formulations was recently given in [22], which defined (p, q)-dEDS for any two non-negative integers p, q. In this setting, an arc (u, v) dominates every other arc which lies in a directed path of length at most q that begins at v, or lies in a directed path of length at most p that ends at u. In other words, (u, v) dominates arcs in the forward direction up to distance q, and in the backward direction up to distance p. The interest in defining the problem in such a general manner is that it allows us to capture at the same time Directed Dominating Set on line digraphs ($(0, 1)$-dEDS), Dominating Set on the underlying graph of the line digraph ($(1, 1)$-dEDS), as well as versions corresponding to r-Dominating Set in the line digraph. We thus obtain a family of optimization problems on digraphs, with varying degrees of symmetry, all of which crucially depend on the directions of arcs in the input digraph.

Our contribution. In this paper we advance the state of the art on the complexity of Directed (p, q)-Edge Dominating Set on two fronts.[3]

First, we study the complexity and approximability of the problem in general. The problem is NP-hard for all values of p, q (except $p = q = 0$), even for planar bounded-degree DAGs [22], so it makes sense to study its parameterized complexity and approximability. We show that its two most natural cases, $(1, 1)$-dEDS and $(0, 1)$-dEDS, admit FPT algorithms with running times 9^k and 2^k respectively, where k is the size of the optimal solution. These algorithms significantly improve upon the FPT algorithms given in [22], which uses the fact

[3] We note that in the remainder we always assume that $p \leq q$, as in the case where $p > q$ we can reverse the direction of all arcs and solve (q, p)-dEDS.

Table 2 Complexity status for various values of p and q: on tournaments.

Range of p, q	Complexity
$p = q = 1$	NP-hard [Thm. 13], FPT [Thm. 2], polynomial kernel [Thm. 7]
$p = 2$ or $q = 2$	Quasi-P-time [Thm. 25], W[2]-hard [Thm 24]
remaining cases	P-time [Thm. 26 and 27]

that the treewidth (of the underlying graph of the input) is at most $2k$ and runs dynamic programming over a tree-decomposition of width at most $10k$, obtained by the algorithm of [5]. The resulting running-time estimate for the algorithm of [22] is thus around 25^{10k}. Though both of our algorithms rely on standard branching techniques, we make use of several non-trivial ideas to obtain reasonable bases in their running times. We also show that both of these problems admit polynomial kernels. These are the only cases of the problem which may admit such kernels, since the problem is W-hard for all other values of p, q [22].

Furthermore, we give an 8-approximation for $(1, 1)$-dEDS and a 3-approximation for $(0, 1)$-dEDS. We recall that [22] showed an $O(\log n)$-approximation for general values of p, q, and a matching logarithmic lower bound for the case $\max\{p, q\} \geq 2$. Therefore our result completes the picture on the approximability of the problem by showing that the only two currently unclassified cases belong in APX.

Finally, we consider the problem's complexity parameterized by the treewidth of the underlying graph and show that, even though the problem is FPT when all of p, q, tw are parameters, it is in fact W[1]-hard if parameterized only by tw. (See Table 1).

Our second, and perhaps main contribution in this paper is an analysis of the complexity of the problem on tournaments, which are one of the most well-studied classes of digraphs (see Table 2). One of the reasons for focusing on this class is that the complexity of Dominating Set has a peculiar status on tournaments, as it is solvable in quasi-polynomial time, W[2]-hard, but neither in P nor NP-complete (under standard assumptions). Here we provide a *complete classification* of the problem which paints an even more surprising picture. We show that (p, q)-dEDS goes from being in P for $p + q \leq 1$; to being APX-hard and unsolvable in $2^{n^{1-\epsilon}}$ under the (randomized) ETH for $p = q = 1$; to being equivalent to Dominating Set on tournaments, hence NP-intermediate, quasi-polynomial-time solvable, and W[2]-hard, when one of p and q equals 2; and finally to being polynomial-time solvable again if $\max\{p, q\} \geq 3$ and neither p nor q equals 2. We find these results surprising, because few problems demonstrate such erratic complexity behavior when manipulating their parameters and because, even though in many cases the problem does seem to behave like Dominating Set, the fact that $(1, 1)$-dEDS becomes significantly harder shows that the problem has interesting complexity aspects of its own. The most technical part of this classification is the reduction that establishes the hardness of $(1, 1)$-dEDS, which makes use of several *randomized* tournament constructions, which we show satisfy certain desirable properties with high probability; as a result our reduction itself is randomized.

Due to space restrictions, some of our proofs are omitted here.

Related Work. On undirected graphs Edge Dominating Set, also known as Maximum Minimal Matching, is NP-complete even on bipartite, planar, bounded degree graphs as well as other special cases [34, 24]. It can be approximated within a factor of 2 [19] (or better in some special cases [8, 29, 2]), but not a factor better than 7/6 [9] unless P=NP. The problem has been the subject of intense study in the parameterized and exact algorithms community [32], producing a series of improved FPT algorithms [17, 3, 18, 30]; the current best is given in [25]. A kernel with $O(k^2)$ vertices and $O(k^3)$ edges is also known [21].

For (p, q)-dEDS, [22] shows the problem to be NP-complete on planar DAGs, in P on trees, and W[2]-hard and $c \ln k$-inapproximable on DAGs if $\max\{p, q\} > 1$. The same paper gives FPT algorithms for $\max\{p, q\} \leq 1$. Their algorithm performs DP on a tree-decomposition of width w in $O(25^w)$, and uses the fact that $w \leq 2k$, and the algorithm of [5] to obtain a decomposition of width $10k$.

Dominating Set is known not to admit an $o(\log n)$-approximation [12, 27], and to be W[2]-hard and unsolvable in time $n^{o(k)}$ under the ETH [13, 10]. The problem is significantly easier on tournaments, as the optimal is always at most $\log n$, hence there is a trivial $n^{O(\log n)}$ (quasi-polynomial)-time algorithm. It remains, however, W[2]-hard [14]. The problem thus finds itself in an intermediate space between P and NP, as it cannot have a polynomial-time algorithm unless FPT=W[2], and it cannot be NP-complete under the ETH (as it admits a quasi-polynomial time algorithm). The generalization of Dominating Set where vertices dominate their r-neighborhood has also been well-studied in general [7, 11, 15, 26]. This problem is much easier on tournaments for $r \geq 2$, as the size of the solution is always a constant [4].

2 Definitions and Preliminaries

Graphs and domination. We use standard graph-theoretic notation. If $G = (V, E)$ is a graph, $S \subseteq V$ a subset of vertices and $A \subseteq E$ a subset of edges, then $G[S]$ denotes the subgraph of G induced by S, while $G[A]$ denotes the subgraph of G that includes A and all its endpoints. We let $V = A \dot{\cup} B$ denote the disjoint set union of A and B. For a vertex $v \in V$, the set of neighbors of v in G is denoted by $N_G(v)$, or simply $N(v)$, and $N_G(S) := (\bigcup_{v \in S} N(v)) \setminus S$ will be written as $N(S)$. We define $N[v] := N(v) \cup \{v\}$ and $N[S] := N(S) \cup S$. Depending on the context, we use (u, v) for $u, v \in V$ to denote either an undirected edge connecting two vertices u, v, or an *arc* (a directed edge) with *tail* u and *head* v. An *incoming* (resp. *outgoing*) arc for vertex v is an arc whose head (resp. tail) is v.

In a directed graph $G = (V, E)$, the set of *out-neighbors* (resp.*in-neighbors*) of a vertex v is defined as $\{u \in V : (v, u) \in E\}$ (resp. $\{u \in V : (u, v) \in E\}$) and denoted as $N^+_G(v)$ (resp. $N^-_G(v)$). Similarly as for undirected graphs, $N^+(S)$ and $N^-(S)$ respectively stand for the sets $(\bigcup_{v \in S} N^+(v)) \setminus S$ and $(\bigcup_{v \in S} N^-(v)) \setminus S$. For a subdigraph H of G and subsets $S, T \subseteq V$, we let $\delta_H(S, T)$ denote the set of arcs in H whose tails are in S and heads are in T. We use $\delta^-_H(S)$ (resp. $\delta^+_H(S)$) to denote the set $\delta_H(V \setminus S, S)$ (resp. the set $\delta_H(S, V \setminus S)$). If S is a singleton consisting of a vertex v, we write $\delta^+_H(v)$ (resp. $\delta^-_H(v)$) instead of $\delta^+_H(\{v\})$ (resp. $\delta^-_H(\{v\})$). The *in-degree* $d^-_H(v)$ (respectively *out-degree* $d^+_H(v)$) of a vertex v is defined as $|\delta^-_H(v)|$ (resp. $|\delta^+_H(v)|$)), and we write $d_H(v)$ to denote $d^+_H(v) + d^-_H(v)$. We omit H if it is clear from the context. If H is $G[A]$ for some vertex or arc set of G, then we write A in place of $G[A]$. A *source* (resp. *sink*) is a vertex that has no incoming (resp. outgoing) arcs.

For integers $p, q \geq 0$, an arc $e = (u, v)$ is said to (p, q)-*dominate* itself, and all arcs that are on a directed path of length at most p to u or on a directed path of length at most q from v. The central problem in this paper is Directed (p, q)-Edge Dominating Set ((p, q)-dEDS): given a directed graph $G = (V, E)$, a positive integer k and two non-negative integers p, q, we are asked to determine whether an arc subset $K \subseteq E$ of size at most k exists, such that every arc is (p, q)-dominated by K (a (p, q)-*edge dominating set* of G).

Complexity background. We assume that the reader is familiar with the basic definitions of parameterized complexity, such as the classes FPT and W[1], as well as the Exponential Time Hypothesis (ETH, see [10]). For a problem P, we let OPT_P denote the value of its optimal solution. We also make use of standard graph width measures, such as *vertex cover number* vc, *treewidth* tw and *pathwidth* pw [10].

Tournaments. A *tournament* is a directed graph in which every pair of distinct vertices is connected by a single arc. Given a tournament T, we denote by T^{rev} the tournament obtained from T by reversing the direction of every arc. Every tournament has a *king* (sometimes also called a 2-king), i.e. a vertex from which every other vertex can be reached by a path of length at most 2. One such king is the vertex of maximum out-degree (see e.g. [4]). It is folklore that any tournament contains a *Hamiltonian path*, i.e. a directed path that uses every vertex. The Dominating Set problem can be solved by brute force in time $n^{O(\log n)}$ on tournaments, by the following lemma:

▶ **Lemma 1** ([10]). *Every tournament on n vertices has a dominating set of size $\leq \log n + 1$.*

3 Tractability

3.1 FPT algorithms

In this section, we present FPT branching algorithms for $(0,1)$-dEDS and $(1,1)$-dEDS. Both algorithms operate along similar lines, taking into consideration the particular ways available for domination of each arc.

▶ **Theorem 2.** *The $(1,1)$-d*EDS *problem parameterized by solution size k can be solved in time $O^*(9^k)$.*

Proof. We present an algorithm that works in two phases. In the first phase we perform a branching procedure which aims to locate vertices with positive out-degree or in-degree in the solution. The general approach of this procedure is standard (as long as there is an uncovered arc, we consider all ways in which it may be covered), and uses the fact that at most $2k$ vertices have positive in- or out-degree in the solution. However, in order to speed up the algorithm, we use a more sophisticated branching procedure which picks an endpoint of the current arc (u,v) and *completely guesses* its behavior in the solution. This ensures that this vertex will never be branched on again in the future. Once all arcs of the graph are covered, we perform a second phase, which runs in polynomial time, and by using a maximum matching algorithm finds the best solution corresponding to the current branch.

Let us now describe the branching phase of our algorithm. We construct three sets of vertices V^+, V^-, V^{+-}. The meaning of these sets is that when we place a vertex u in V^+, V^-, or V^{+-} we guess that u has (i) positive out-degree and zero in-degree in the optimal solution; (ii) positive in-degree and zero out-degree in the optimal solution; (iii) positive in-degree and positive out-degree in the optimal solution, respectively. Initially all three sets are empty. When the algorithm places a vertex in one of these sets we say that the vertex has been *marked*.

Our algorithm now proceeds as follows: given a graph $G(V,E)$ and three disjoint sets V^+, V^-, V^{+-} we do the following:

1. If $|V^+| + |V^-| + 2|V^{+-}| > 2k$, reject.
2. While there exists an arc (u,v) with both endpoints unmarked do the following and return the best solution:
   a. Call the algorithm with $V^+ := V^+ \cup \{v\}$ and other sets unchanged.
   b. Call the algorithm with $V^{+-} := V^{+-} \cup \{v\}$ and other sets unchanged.
   c. Call the algorithm with $V^- := V^- \cup \{u\}$ and other sets unchanged.
   d. Call the algorithm with $V^{+-} := V^{+-} \cup \{u\}$ and other sets unchanged.
   e. Call the algorithm with $V^+ := V^+ \cup \{u\}$, $V^- := V^- \cup \{v\}$, and V^{+-} unchanged.

It is not hard to see that Step 1 is correct as as $|V^+| + |V^-| + 2|V^{+-}|$ is a lower bound on the sum of the degrees of all vertices in the optimal and therefore cannot surpass $2k$.

Branching Step 2 is also correct: in order to cover (u, v) the optimal solution must either take an arc coming out of v (2a,2b), or an arc coming into u (2c,2d), or, if none of the previous cases apply, it must take the arc itself (2e).

Once we have applied the above procedure exhaustively, all arcs of the graph have at least one marked endpoint. We say that an arc (u, v) with $u \in V^- \cup V^{+-}$, or with $v \in V^+ \cup V^{+-}$ is covered. We now check if the graph contains an uncovered arc (u, v) with exactly one marked endpoint. We then branch by considering all possibilities for its other endpoint. More precisely, if $u \in V^+$ and v is unmarked, we branch into three cases, where v is placed in V^+, or V^-, or V^{+-} (and similarly if v is the marked endpoint). This branching step is also correct, since the degree specification for the currently marked endpoint does not dominate the arc (u, v), hence any feasible solution must take an arc incident on the other endpoint.

Once the above procedure is also applied exhaustively we have a graph where all arcs either have both endpoints marked, or have one endpoint marked but in a way that if we respect the degree specifications the arc is guaranteed to be covered. What remains is to find the best solution that agrees with the specifications of the sets V^+, V^-, V^{+-}.

We first add to our solution S all arcs $\delta(V^+, V^-)$, i.e. all arcs (u, v) such that $u \in V^+$ and $v \in V^-$, since there is no other way to dominate these arcs. We then define a bipartite graph $H = (V^+ \cup V^{+-}, V^- \cup V^{+-}, \delta(V^+ \cup V^{+-}, V^- \cup V^{+-}))$. That is, H contains all vertices in V^+ along with a copy of V^{+-} on one side, all vertices of V^- and a copy of V^{+-} on the other side and all arcs in E with tails in $V^+ \cup V^{+-}$ and heads in $V^- \cup V^{+-}$. We now compute a minimum edge cover of this graph, that is, a minimum set of edges that touches every vertex. This can be done in polynomial time by finding a maximum matching and then adding an arbitrary incident edge for each unmatched vertex. It is not hard to see that a minimum edge cover of this graph corresponds exactly to the smallest $(1, 1)$ edge dominating set that satisfies the specifications of the sets V^+, V^-, V^{+-}.

To see that the running time of our algorithm is $O^*(9^k)$ we observe that there are two branching steps: either we have an arc (u, v) with both endpoints unmarked; or we have an arc with exactly one unmarked endpoint. In both cases we measure the decrease of the quantity $\ell := 2k - (|V^+| + |V^-| + |V^{+-}|)$. The first case produces two instances with $\ell' := \ell - 1$ (2a,2c), and three instances with $\ell' := \ell - 2$. We therefore have the recurrence $T(\ell) \leq 2T(\ell - 1) + 3T(\ell - 2)$ which gives $T(\ell) \leq 3^\ell$. For the second case, we have three branches, all of which decrease ℓ, therefore we also have $T(\ell) \leq 3^\ell$ in this case. Taking into account that, initially $\ell = 2k$ we get a running time of at most $O^*(9^k)$. ◀

▶ **Theorem 3.** *The* $(0, 1)$*-d*EDS *problem parameterized by solution size* k *can be solved in time* $O^*(2^k)$.

3.2 Approximation algorithms

We present here constant-factor approximation algorithms for $(0, 1)$-dEDS, and $(1, 1)$-dEDS. Both algorithms appropriately utilize a maximal matching.

▶ **Theorem 4.** *There are polynomial-time* 3*-approximation algorithms for* $(0, 1)$*-d*EDS.

▶ **Theorem 5.** *There is a polynomial-time* 8*-approximation algorithm for* $(1, 1)$*-d*EDS.

Proof. Let $G = (V, E)$ be an input directed graph. We partition V into (S, R, T) so that S and T are the sets of sources and sinks respectively, and $R = V \setminus S \setminus T$. We construct an $(1, 1)$-edge dominating set K as follows.

1. Add the arc set $\delta(S,T)$ to K.
2. For each vertex of $v \in R \cap N^+(S)$, choose precisely one arc from $\delta^+(v)$ and add it to K.
3. For each vertex of $v \in R \cap N^-(T)$, choose precisely one arc from $\delta^-(v)$ and add it to K.
4. Let $G' = (R, E')$ be the subdigraph of G whose arc set consists of those arcs not $(1,1)$-dominated by K thus far constructed. Let M be a maximal matching in (the underlying graph of) G'. Let M^- and M^+ be respectively the tails and heads of the arcs in M. To K, we add all arcs of M, an arc of $\delta^-_G(v)$ for every $v \in M^-$, and also an arc of $\delta^+_G(v)$ for every $v \in M^+$.

Clearly, the algorithm runs in polynomial time. In particular, for any vertex v considered at Step 2-4, both $\delta^+(v)$ and $\delta^-(v)$ are non-empty and choosing an arc from a designated set is always possible. We show that K is indeed an $(1,1)$-edge dominating set. Suppose that an arc (u,v) is not $(1,1)$-dominated by K. As the first, second and third step of the construction ensures that any arc incident with $S \cup T$ is $(1,1)$-dominated, we know that (u,v) is contained in the subdigraph G' constructed at step 4. For $(u,v) \notin M$ and M being a maximal matching, one of the vertices u, v must be incident with M. Without loss of generality, we assume v is incident with M (and the other cases are symmetric). If $v \in M^-$, then clearly the arc $e \in M$ whose tail coincides with v would $(1,0)$-dominate (u,v), a contradiction. If $v \in M^+$, then the outgoing arc of v added to K at step 4 would $(1,0)$-dominate (u,v), again reaching a contradiction. Therefore, the constructed set K is a solution to $(1,1)$-dEDS.

To prove the claimed approximation ratio, we first note that $\delta(S,T)$ is contained in any (optimal) solution because any arc of $\delta(S,T)$ can be $(1,1)$-dominated only by itself. Note that these arcs do not $(1,1)$-dominate any other arcs of G. Further, we have $|R \cap N^+(S)| \leq OPT_{(1,1)dEDS} - |\delta(S,T)|$ because in order to $(1,1)$-dominate any arc of the form (s,r) with $s \in S$ and $r \in R$, one must take at least one arc from $\{(s,r)\} \cup \delta^+(r)$. Since the collection of sets $\{(s,r) : s \in S\} \cup \delta^+(r)$ are disjoint over all $r \in R \cap N^+(S)$, the inequality holds. Likewise, it holds that $|R \cap N^-(T)| \leq OPT_{(1,1)dEDS} - |\delta(S,T)|$. In order to $(1,1)$-dominate the entire arc set M, one needs to take at least $|M|/2$ arcs. This is because an arc e can $(1,1)$-dominate at most two arcs of M. That is, we have $|M|/2 \leq OPT_{(1,1)dEDS} - |\delta(S,T)|$ Therefore, it is $|K| \leq |\delta(S,T)| + |R \cap N^+(S)| + |R \cap N^-(T)| + 3|M| \leq 8OPT_{(1,1)dEDS}$. ◀

3.3 Polynomial kernels

We give polynomial kernels for $(1,1)$-dEDS and $(0,1)$-dEDS. We first introduce a relation between the vertex cover number and the size of a minimum $(1,1)$-edge dominating set, shown in [22] and then proceed to show a quadratic-vertex/cubic-edge kernel for $(1,1)$-dEDS.

▶ **Lemma 6** ([22]). *Given a directed graph G, let G^* be the undirected underlying graph of G, $vc(G^*)$ be the vertex cover number of G^*, and K be a minimum $(1,1)$-edge dominating set in G. Then $vc(G^*) \leq 2|K|$.*

▶ **Theorem 7.** *There exists an $O(k^2)$-vertex/$O(k^3)$-edge kernel for $(1,1)$-dEDS.*

Proof. Given a directed graph G, we denote the underlying undirected graph of G by G^*. Let K be a minimum $(1,1)$-edge dominating set and $\mathrm{vc}(G^*)$ be the size of a minimum vertex cover in G^*. First, we find a maximal matching M in G^*. If $|M| > 2k$, we conclude this is a no-instance by Lemma 6 and the well-known fact that $|M| \leq \mathrm{vc}(G^*)$ [20]. Otherwise, let S be the set of endpoints of edges in M. Then S is a vertex cover of size at most $4k$ for the underlying undirected graph of G and $V \setminus S$ is an independent set.

We next explain the reduction step. For each $v \in S$, we arbitrarily mark the first $k+1$ tail vertices of incoming arcs of v with "in" (or all, if the in-degree of v is $\leq k$) and also arbitrarily the first $k+1$ head vertices of outgoing arcs of v with "out" (or all, if the out-degree of v is

$\leq k$). After this marking, if there exists a vertex $u \in V \setminus S$ without marks "in", "out", we can delete it. We next show correctness. First, we can observe that if some $v \in S$ has more than $k+1$ incoming arcs, they must be dominated by an outgoing arc of v. Similarly, if $v \in S$ has more than $k+1$ outgoing arcs, they must be dominated by an incoming arc of v. This means that every arc incident on an unmarked vertex u must be dominated because each vertex v in S adjacent to u has at least $(k+1)$ incoming arcs other than (u, v), or $(k+1)$ outgoing arcs other than (v, u), due to the fact that u is unmarked. Moreover, for an incoming (resp. outgoing) arc of u, there exists an outgoing (resp. incoming) arc of $v \in S$ that dominates all arcs dominated by the incoming (resp. outgoing) arc of u except for arcs incident on u. Thus we need not include any arc incident on u in the solution. By the reduction step, we obtain the reduced graph.

From the above, the size of an independent set, being the subset of $V \setminus S$, is bounded by $4k \cdot 2(k+1) = 8k^2 + 8k$, following the reduction step. Thus, the number of vertices in the reduced graph is at most $4k + 8k^2 + 8k = 8k^2 + 12k$. Moreover, there exist at most $4k \cdot (8k^2 + 12k) = 32k^3 + 48k^2$ arcs between the sets of the vertex cover and the independent set. Therefore, the number of arcs in the reduced graph is at most $\binom{4k}{2} + 32k^3 + 48k^2 = 32k^3 + 56k^2 - 2k$. ◀

Using a more strict relation between vc and the size of a minimum $(0,1)$-edge dominating set, we obtain a linear-vertex/quadratic-edge kernel for $(0,1)$-dEDS.

▶ **Theorem 8.** *There exists an $O(k)$-vertex/$O(k^2)$-edge kernel for $(0,1)$-d*EDS*.*

4 W[1]-hardness by treewidth

In this section we characterize the complexity of (p,q)-dEDS parameterized by treewidth. Our main result is that, even though the problem is FPT when parameterized by $p+q+\mathrm{tw}$, it becomes W[1]-hard if parameterized only by tw. The algorithm is based on standard dynamic programming techniques, while for hardness we reduce from the k-Multicolored Clique problem, which is defined as follows: given a graph $G = (V, E)$, with V partitioned into k independent sets $V = V_1 \uplus \cdots \uplus V_k$, $|V_i| = n, \forall i \in [1,k]$, we are asked to find a subset $S \subseteq V$, such that $G[S]$ forms a clique with $|S \cap V_i| = 1, \forall i \in [1,k]$. The problem k-Multicolored Clique is well-known to be W[1]-complete [16].

Construction. Given an instance $[G = (V, E), k]$ of k-Multicolored Clique, with $V = \bigcup_{\forall i \in [1,k]} V_i$ and $V_i = \{v_0^i, \ldots, v_{n-1}^i\}$ we will construct an instance $[G' = (V', E'), \mathrm{tw}(G')]$ of (p,q)-dEDS parameterized by the treewidth of the underlying undirected graph, with $p = q = 2n$, as follows. We first make k *main cycles* on n vertices $V_i' = \{u_0^i, \ldots, u_{n-1}^i\}$, $\forall i \in [1,k]$, each corresponding to a set $V_i \subseteq V$ and we associate each vertex $v_l^i \in V_i$ with the arc (u_l^i, u_{l+1}^i) from cycle V_i' (its *corresponding* arc). Let $\bar{E}$ be the set of *non-edges* between vertices from different sets from G, i.e. the set of all pairs $(v_l^i, v_o^j) \notin E$.

For each $(v_l^i, v_o^j) \in \bar{E}$ with $i < j$, we will create the following *cross-gadget* $\hat{C}_{l,o}^{i,j}$: we first make five new vertices $a_{l,o}^{i,j}$, $b_{l,o}^{i,j}$, $c_{l,o}^{i,j}$, $d_{l,o}^{i,j}$ and $e_{l,o}^{i,j}$ and then add arcs from $a_{l,o}^{i,j}$ and $c_{l,o}^{i,j}$ to $e_{l,o}^{i,j}$ and from $e_{l,o}^{i,j}$ to $b_{l,o}^{i,j}$ and $d_{l,o}^{i,j}$. We let set $Q_{l,o}^{i,j}$ contain all four of these arcs and refer to them as the *cross-arcs*. We also add both arcs between $a_{l,o}^{i,j}$ and $c_{l,o}^{i,j}$, as well as both arcs between $b_{l,o}^{i,j}$ and $d_{l,o}^{i,j}$. These are referred to as the *flip-arcs*. Finally, we add a path of length $4n-2$ from $b_{l,o}^{i,j}$ to $a_{l,o}^{i,j}$ and a path of length $4n-2$ from $d_{l,o}^{i,j}$ to $c_{l,o}^{i,j}$ (on $4n-3$ new vertices each). We call these the *long paths*.

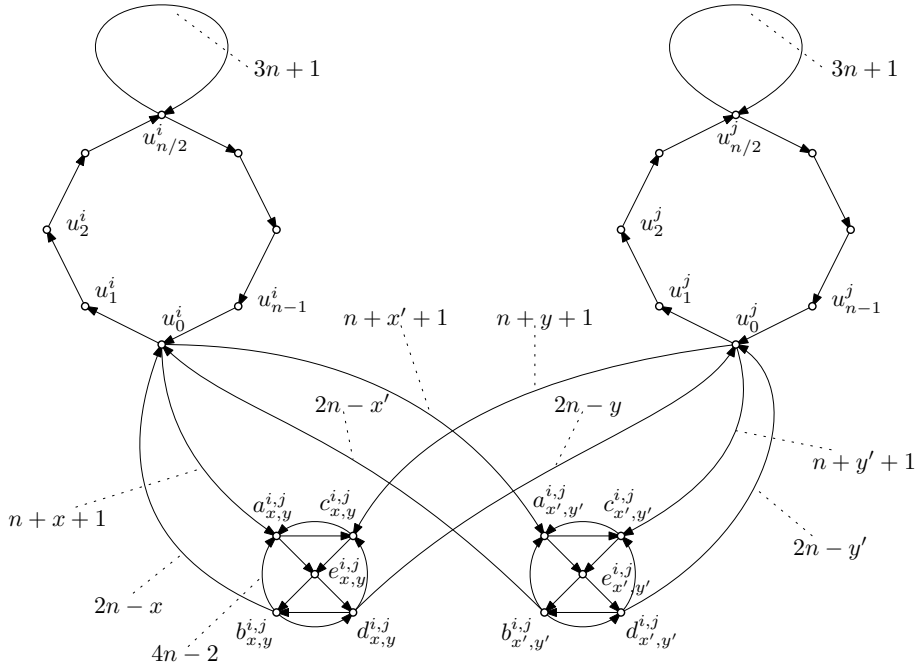


Figure 1 An example of our construction (even n). Dotted lines show the length of each path.

To connect each gadget to the main cycles, we then add a path of length $n+l+1$ (with $n+l$ new vertices) from u^i_0 to $a^{i,j}_{l,o}$ and a path of length $2n-l$ (with $2n-l-1$ new vertices) from $b^{i,j}_{l,o}$ to u^i_0. We also add a path of length $n+o+1$ from u^j_0 to $c^{i,j}_{l,o}$ and a path of length $2n-o$ from $d^{i,j}_{l,o}$ to u^j_0.

Finally, in order to ensure any $(2n, 2n)$-edge dominating set will select at least one arc from each of the k main cycles, we will attach a *guard cycle* to each *middle* vertex of each V'_i: the middle vertex of V'_i is $u^i_{n/2}$ and we attach a cycle of length $3n+1$ to it.[4] This concludes our construction and Figure 1 provides an illustration. Clearly, the construction requires polynomial time.

▶ **Lemma 9.** *If G has a k-multicolored clique of size k, then G' has a $(2n, 2n)$-edge dominating set of size $|\bar{E}| + k$.*

▶ **Lemma 10.** *If G' has a $(2n, 2n)$-edge dominating set of size $|\bar{E}| + k$, then G has a k-multicolored clique of size k.*

▶ **Theorem 11.** *The (p, q)-dEDS problem is W[1]-hard parameterized by the treewidth of the input graph.*

▶ **Theorem 12.** *The (p, q)-dEDS problem can be solved in time $O^*((p+q)^{O(tw)})$ on graphs of treewidth at most tw.*

Proof (Sketch). The proof relies on standard techniques (Dynamic Programming over tree decompositions), so we only sketch the details here. Our algorithm maintains a table for each node of the given tree decomposition, indexed by a set of *state-assignments* to all vertices in

[4] We assume, without loss of generality, that n is even as we can always add a dummy vertex to each subset V_i.

the bag, each entry of which contains the minimum number of selected arcs from the node's terminal subgraph for the state of each vertex to be justified, i.e. for the partial solution described by this set of states to be valid. The state of each vertex in the bag describes its distance to the closest endpoint of a selected arc, i.e. it either has a path of length at most p to the tail of a selected arc, or the head of a selected arc has a path of length at most q to the vertex in question. We also use "promise" states signifying that the partial solution has not yet selected the arc that will be closest to some vertex, by doubling the amount of states we use. It is not hard to see that using such a state representation, we can compute the values of all partial solutions for the problem over the nodes of the tree decomposition in time polynomial on the table's size: the states of introduced vertices must match the distances in the node's subgraph, all partial solutions involving a forgotten vertex must be compared over all its states to retain the minimum, while for join nodes, the state of a vertex must match the "promise" state for the same vertex in the other branch of the join for the partial solutions to be accurately extended. In this way we can check the values of potential global solutions in the table of the root node of the tree decomposition. ◀

5 On Tournaments

A complete complexity classification for the problems (p, q)-dEDS is presented in this section. For $p = q = 1$, the problem is NP-hard under a randomized reduction while being amenable to an FPT algorithm and polynomial kernelization due to the results of Sections 3.1 and 3.3. The hardness reduction is given in Subsection 5.1. When $p = 2$ or $q = 2$, the complexity status of (p, q)-dEDS is equivalent to Dominating Set on tournaments and is discussed in Subsection 5.2. In the remaining cases, when $p + q \leq 1$, or $\max\{p, q\} \geq 3$ while neither of them equals 2, the problems turn out to be in P (Subsection 5.3).

5.1 Hard: when $p = q = 1$

We present a randomized reduction from Independent Set to $(1, 1)$-dEDS. Our reduction preserves the size of the instance up to polylogarithmic factors; as a result it shows that $(1, 1)$-dEDS does not admit a $2^{n^{1-\epsilon}}$ algorithm, under the randomized ETH. Furthermore, our reduction preserves the optimal value, up to a factor $(1 - o(1))$; as a result, it shows that $(1, 1)$-dEDS is APX-hard under randomized reductions.

Before moving on, let us give a high-level overview of our reduction. The first step is to reduce Independent Set to Almost Induced Matching, the problem of finding the maximum set of vertices that induce a graph of maximum degree 1. Our reduction produces an instance of Almost Induced Matching that has several special properties, notably producing a bipartite graph $G = (A, B, E)$. The basic strategy will be then to construct a tournament $T = (V', E')$, where $V' = A \cup B \cup C$, where C is a set of new vertices. All edges of E will be directed from A to B, non-edges of E will be directed from B to A, and all other edges will be set randomly. This intuitively encodes the structure of G in T. The idea is now that a solution S in G (that is, a set of vertices of G that induces a graph with maximum degree 1) will correspond to an edge dominating set in T where all vertices except those of S will have total degree 2, and the vertices of S will have total degree 1. In particular, vertices of $S \cap A$ will have out-degree 1 and in-degree 0, and vertices of $S \cap B$ will have in-degree 1 and out-degree 0.

The random structure of the remaining arcs of the tournament T is useful in two respects: in one direction, given the solution S for G, it is easy to deal with vertices that have degree 1 in $G[S]$: we select the corresponding arc from A to B in T. For vertices of degree 0 however,

we are forced to look for edge-disjoint paths that will allow us to achieve our degree goals. Such paths are guaranteed to exist if C is random and large enough. In the other direction, given a good solution in T, we would like to guarantee that, because the internal structure of A, B, and C is chaotic, the only way to obtain a large number of vertices with low degree is to place those with in-degree 0 in A, and those with out-degree 0 in B.

▶ **Theorem 13.** $(1,1)$-dEDS *on tournaments cannot be solved in polynomial time, unless* $NP \subseteq BPP$. *Furthermore,* $(1,1)$-dEDS *is APX-hard under randomized reductions, and does not admit an algorithm running in time* $2^{n^{1-\epsilon}}$ *for any* ϵ, *unless the randomized ETH is false.*

We first reduce the Independent Set problem on cubic graphs to the following intermediate problem called Almost Induced Matching, commonly known as Maximum Dissociation Number in the literature [33, 31]. A subgraph of G induced on a vertex set $S \subseteq V$ is called an *almost induced matching*, if every vertex $v \in S$ has degree ≤ 1 in $G[S]$.

▶ **Definition 14.** The problem Almost Induced Matching (AIM) takes as input an undirected graph $G = (V, E)$. The goal is to find an almost induced matching having the maximum number of vertices.

▶ **Theorem 15.** *[1, 10]* Independent Set *is APX-hard on cubic graphs. Furthermore,* Independent Set *cannot be solved in time* $2^{o(n)}$ *unless the ETH is false.*

Almost Induced Matching is known to be NP-complete on bipartite graphs of maximum degree 3 and on C_4-free bipartite graphs [6]. It is also NP-hard to approximate on arbitrary graphs within a factor of $n^{1/2-\epsilon}$ for any $\epsilon > 0$ [28]. The next lemma supplements the known hardness results on bipartite graphs and might be of independent interest.

▶ **Lemma 16.** Almost Induced Matching *is APX-hard and cannot be solved in time* $2^{o(n)}$ *under the ETH, even on bipartite graphs of degree at most 4. Furthermore, this hardness still holds if we are promised that* $OPT_{AIM} > 0.6n$ *and that there is an optimal solution* S *that includes at least* $n/20$ *vertices with degree* 0 *in* $G[S]$.

As we use a random construction, the following property of a uniform random tournament is useful. Intuitively, the property established in Lemma 17 states that it is impossible in a large random tournament to have two large sets of vertices X, Y such that all vertices of X have in-degree 0 and out-degree 1 in a $(1,1)$-edge dominating set, while all vertices of Y have in-degree 1 and out-degree 0.

▶ **Lemma 17.** *Let* $T = (V, E)$ *be a random tournament on the vertex set* $\{1, 2, \ldots, n\}$, *in which* (i, j) *is an arc of* T *with probability* $1/2$. *Then the following event happens with high probability: for any two disjoint sets* $X, Y \subseteq V$ *with* $|X| > (\log n)^2$ *and* $|Y| > (\log n)^2$, *there exists a vertex* $x \in X$ *with at least two outgoing arcs to* Y.

▶ **Lemma 18.** *Let* $G = (V = A \dot\cup B \dot\cup C, E)$ *be a random directed graph with* $|A| = |B| = n$ *and* $|C| = 4n$ *such that for any pair* (x, y) *with* $\{x, y\} \cap C \neq \emptyset$ *we have exactly one arc, oriented from* x *to* y, *or from* y *to* x *with probability* $1/2$. *Let* $\ell \geq n/20$ *be a positive integer. Then with high probability, we have: for any two disjoint sets* $X \subseteq A$,$Y \subseteq B$ *with* $|X| = |Y| = \ell$, *there exist* ℓ *vertex-disjoint directed paths from* X *to* Y.

▶ **Theorem 19.** *There is a probabilistic polynomial-time algorithm computing, given an instance* G *of* Almost Induced Matching, *an instance* T *of* $(1,1)$-dEDS *such that with high probability:*

(i) *if* $OPT_{AIM}(G) \geq L_1$, *then* $OPT_{(1,1)dEDS}(T) \leq |V(T)| - L_1/2 + 1$,
(ii) *if* $OPT_{AIM}(G) < L_2 - 5(\log L_2)^2$, *then* $OPT_{(1,1)dEDS}(T) > |V(T)| - L_2/2 + 1$.

Proof of Theorem 13. Let G be an instance of Independent Set on cubic graphs and let G' be the instance of Almost Induced Matching obtained by the construction of Lemma 16. We set ℓ as in the reduction and observe that $OPT_{IS}(G) \geq k$ if and only if $OPT_{AIM}(G') \geq \ell$.

Let G^* be a disjoint union of $10(\log \ell)^2$ copies of G'. Then G^* is a gap instance, whose optimal solution is either at least $10\ell(\log \ell)^2$, or at most $10\ell(\log \ell)^2 - 10(\log \ell)^2 \leq L - 5(\log L)^2$, where $L := 10\ell(\log \ell)^2$. Now Theorem 19 implies that using a probabilistic polynomial-time algorithm for $(1,1)$-dEDS with two-sided bounded errors, one can correctly decide an instance of Independent Set on cubic graphs with bounded errors. We observe that the size of the instance has only increased by a poly-logarithmic factor, hence an algorithm solving the new instance in time $2^{n^{1-\epsilon}}$ would give a randomized sub-exponential time algorithm for 3-SAT.

Finally, for APX-hardness, we observe that we may assume we start our reduction from an Independent Set instance where either $OPT_{IS} \geq k$ or $OPT_{IS} < rk$, for some constant $r < 1$, and for $k = \Theta(n)$. Lemma 16 then gives an instance of Almost Induced Matching where either $OPT_{AIM} \geq L_1$ or $OPT_{AIM} \leq r'L_1 = L_2$, for some (other) constant $r' < 1$. We now use Theorem 19 to create a gap-instance of $(1,1)$-dEDS. ◀

5.2 Equivalent to Dominating Set on tournaments: $p = 2$ or $q = 2$

▶ **Lemma 20.** *On tournaments without a source, we have $OPT_{(0,2)dEDS} \leq OPT_{DS}$.*

Proof. Let $T = (V, E)$ be a tournament with no source and $D \subseteq V$ be a dominating set of T. Then let $K \subseteq E$ be a set containing one arbitrary incoming arc of every vertex in D. We claim K $(0,2)$-dominates all arcs in E: since D is a dominating set, for any vertex $u \notin D$ there must be an arc (v,u) from some $v \in D$. Thus all outgoing arcs (u,w) from such $u \notin D$ are $(0,2)$-dominated by K, as are all arcs (v,u) from $v \in D$. ◀

▶ **Lemma 21.** *Let $T = (V, E)$ be a tournament and let s be a source of T. Then $\delta^+(s)$ is an optimal (p,q)-edge dominating set of T for any $p \leq 1$ and $q \geq 1$.*

Proof. Since s has no incoming arcs, any (p,q)-edge dominating set must select at least one arc from $\{(s,v)\} \cup \delta^+(v)$ for every $v \in V \setminus \{s\}$ in order to (p,q)-dominate (s,v). Because the arc sets $\{(s,v)\} \cup \delta^+(v)$ are mutually disjoint over all $v \in V \setminus \{s\}$, any (p,q)-edge dominating set has size at least $|\delta^+(s)|$. Now, observe that $\delta^+(s)$ $(0,1)$-dominates every arc of T. ◀

▶ **Lemma 22.** *On tournaments on n vertices, for any $p \geq 2$ we have: $OPT_{(p,2)dEDS} \leq OPT_{(2,2)dEDS} \leq 2\log n + 3$.*

Proof. The first inequality trivially holds, so we prove the second inequality. Let $T = (V, E)$ be a tournament on n vertices. If T has no source, then $OPT_{(2,2)dEDS} \leq OPT_{(0,2)dEDS} \leq OPT_{DS} \leq \log n + 1$, where the second and the last inequality follow from Lemma 20 and Lemma 1, respectively. If T^{rev} contains no source, observe that a $(0,2)$-edge dominating set of T^{rev} is a $(2,0)$-edge dominating set of T and the statement holds.

Therefore, we may assume that T has a source s and a sink t. Let $S_1 \subseteq V \setminus \{s\}$ be a dominating set of $T - s$ of size at most $\log n + 1$. Clearly, every arc (u,v) of $T - s$ lies on a directed path of length at most two from some vertex of S_1. Let $D_1 \subseteq E$ be a minimal arc set such that $D_1 \cap \delta^-(v) \neq \emptyset$ for every $v \in S_1$. Since every $v \in S_1$ has positive in-degree, such a set D_1 exists and we have $|D_1| \leq |S_1|$. Observe that D_1 (0,2)-dominates every arc of $T - s$. Applying a symmetric argument to $T^{rev} - t$, we know that there exists an arc set D_2 of size at most $\log n + 1$ which $(2,0)$-dominates every arc of $T - t$. Now $D_1 \cup D_2$ (2,2)-dominates every arc incident with $V \setminus \{s,t\}$. Therefore, $D_1 \cup D_2 \cup \{(s,t)\}$ is a $(2,2)$-dEDS. ◀

▶ **Lemma 23.** *There is an FPT reduction from* Dominating Set *on tournaments parameterized by solution size to* (p,q)-EDS *parameterized by solution size, when* $p = 2$ *or* $q = 2$.

Proof. Without loss of generality we assume that $q = 2$. Let $T = (V, E)$ be an input tournament to Dominating Set, and let k be the solution size. It can be assumed that T has no source. We construct a tournament T' on vertex set $V \cup \{t\}$, in which t is a sink. Given a dominating set D of T, we select an arbitrary arc set K of T' so that $\delta^-_K(v) = 1$ for each $v \in D$. It is easy to see that K $(0,2)$-dominates every arc of T': any arc (u,v) with $u \in D$ is clearly dominated by K. For any arc (u,v) with $u \notin D$, there is $w \in D$ such that $(w,u) \in E$ and thus K $(0,2)$-dominates (u,v).

Conversely, suppose that K is a $(p,2)$-edge dominating set of size at most k and let K^+ be the set of heads of K found in V. Let K^- be the set of vertices $u \in V$ such that $(u,t) \in K$. We have $|K^+ \cup K^-| \leq k$, because each arc of K either contributes an element in K^+ or in K^-. We claim that $K^+ \cup K^-$ is a dominating set of T. Suppose the contrary, therefore there exists $u \in V \setminus (K^+ \cup K^-)$ that is not dominated by $K^+ \cup K^-$. However, the arc (u,t) is dominated by K. We have $(u,t) \notin K$, as $u \notin K^-$. Therefore, since t is a sink, (u,t) is $(0,2)$-dominated by an arc $(v,w) \in K$. This means that either $w = u$, or the arc (w,u) exists. However, $w \in K^+$, which means that u is dominated. ◀

▶ **Theorem 24.** *On tournaments, the problems* $(p,2)$-dEDS *are W[2]-hard for each fixed* p.

Proof. For all problems, we use the reduction from Set Cover to Dominating set on Tournaments given in Theorem 13.14 of [10] and our results follow from the W[2]-hardness of that problem (see also Theorem 13.28 therein) and Lemma 23. ◀

▶ **Theorem 25.** *On tournaments, the problems* $(0,2)$-dEDS, $(1,2)$-dEDS *and* $(2,2)$-dEDS *can be solved in time* $n^{O(\log n)}$.

Proof. For $(0,2)$-dEDS and $(1,2)$-dEDS, the case when a given tournament contains a source can be solved in polynomial time by Lemma 21. If the input tournament contains no source, then by Lemma 20 we have $OPT_{(1,2)dEDS} \leq OPT_{(0,2)dEDS} \leq OPT_{DS}$, which is bounded by $\log n + 1$ by Lemma 1. Lemma 22 states that $OPT_{(p,2)dEDS} \leq 2\log n + 3$. Exhaustive search over vertex subsets of size $O(\log n)$ performs in the claimed runtime. ◀

5.3 P-time solvable: $p + q \leq 1$ or, $2 \notin \{p,q\}$ and $\max\{p,q\} \geq 3$

▶ **Theorem 26.** $(0,1)$-dEDS *can be solved in polynomial time on tournaments.*

Proof. We will show that $OPT_{(0,1)dEDS} = n - 1$ and give a polynomial-time algorithm for finding such an optimal solution. First, given a tournament $T = (V, E)$, to see why $OPT_{(0,1)dEDS} \geq n - 1$ consider any optimal solution $K \subseteq E$: if there exists a pair of vertices $u, v \in V$ with $d^-_K(u) = d^-_K(v) = 0$, i.e. a pair of vertices, neither of which has an arc of K as an incoming arc, then the arc between them (without loss of generality let its direction be (v,u)) is not dominated: as $d^-_K(u) = 0$, the arc itself does not belong in K and as $d^-_K(v) = 0$, there is no arc preceding it that is in K. This leaves (v,u) undominated. Therefore, there cannot be two vertices with no incoming arcs in any optimal solution, implying any solution must include at least $n - 1$ arcs.

To see $OPT_{(0,1)dEDS} \leq n-1$, consider a partition of T into strongly connected components $C_1, \ldots, C_l$, where we can assume these are given according to their topological ordering, i.e. for $1 \leq i < j \leq l$, all arcs between C_i and C_j are directed towards C_j. Let S be the set

of arcs traversed in breadth-first-search (BFS) from some vertex $s \in C_1$ until all vertices of C_1 are spanned. Also let S' be the set of arcs $(s,u), \forall u \in C_i, \forall i \in [2,l]$, i.e. all outgoing arcs from s to every vertex of $C_2, \ldots, C_l$. Note that set S' must contain an arc from s to *every* vertex that is not in C_1: T being a tournament means every pair of vertices has an arc between them and C_1 being the first component in the topological ordering means all arcs between its vertices and those of subsequent components are oriented away from C_1. Then $K \coloneqq S \cup S'$ is a directed $(0,1)$-edge dominating set of size $n-1$ in T: observe that $d^-_K(u) = 1, \forall u \neq s \in T$, i.e. every vertex in T has positive in-degree within K except s. Thus all outgoing arcs from all such vertices u are $(0,1)$-dominated by K, while all outgoing arcs from s are in K, due to the BFS selection for S and the definition of S'.

Since such an optimal solution K can be computed in polynomial time (partition into strongly connected components, BFS), the claim follows. ◀

▶ **Theorem 27.** *For any p, q with $\max\{p,q\} \geq 3$, $p \neq 2$ and $q \neq 2$, (p,q)-dEDS can be solved in polynomial time on tournaments.*

Proof. Suppose without loss of generality that $q \geq 3$, as otherwise we can solve (q,p)-dEDS on T^{rev}, the tournament obtained by reversing the orientation of every arc. In any tournament T, there always exists a *king* vertex, that is, a vertex with a path of length at most 2 to any other vertex in the graph. One such vertex is the vertex of maximum out-degree v. If v is not a source, it suffices to select one of its incoming arcs: since there is a path of length at most 2 from v to any other vertex u in the graph, any outgoing arc from any such u will be $(0,3)$-dominated by this selection. This is clearly optimal.

Suppose now that s is a source. We consider two cases: if $p \leq 1$, then Lemma 21 implies that $\delta^+(s)$ is optimal. Finally, suppose s is a source and $p \geq 3$. If T does not have a sink, then a king of T^{rev} has an incoming arc, which $(0,3)$-dominates T^{rev} as observed above, and thus T has a $(0,3)$-edge dominating set of size 1.

Therefore, we may assume that T has both a source s and a sink t. Let s' and t' be vertices of $V \setminus \{s,t\}$ with maximum out- and in-degree, respectively. Now $\{(s,t),(s,s'),(t',t)\}$ is a $(3,3)$-edge dominating set. This is because s' is a king of $T-s$ and thus every arc (u,v) with $u \neq s$ is $(0,3)$-dominated by (s,s'). Similarly, every arc (u,v) with $v \neq t$ is $(3,0)$-dominated by (t',t). The only arc not $(3,3)$-dominated by these two arcs is (s,t), which is dominated by itself. Examining all vertex subsets of size up to 3, we can compute an optimal $(3,3)$-edge dominating set in polynomial time. ◀

References

1 Paola Alimonti and Viggo Kann. Some apx-completeness results for cubic graphs. *Theor. Comput. Sci.*, 237(1-2):123–134, 2000.

2 Brenda S. Baker. Approximation Algorithms for NP-Complete Problems on Planar Graphs. *J. ACM*, 41(1):153–180, 1994.

3 Daniel Binkele-Raible and Henning Fernau. Enumerate and measure: Improving parameter budget management. In *IPEC*, volume 6478 of *Lecture Notes in Computer Science*, pages 38–49. Springer, 2010.

4 Arindam Biswas, Varunkumar Jayapaul, Venkatesh Raman, and Srinivasa Rao Satti. The Complexity of Finding (Approximate Sized) Distance-d Dominating Set in Tournaments. In *Frontiers in Algorithmics*, pages 22–33, 2017.

5 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A c^k n 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016.

6 Rodica Boliac, Kathie Cameron, and Vadim V. Lozin. On computing the dissociation number and the induced matching number of bipartite graphs. *Ars Comb.*, 72, 2004.

7 Glencora Borradaile and Hung Le. Optimal dynamic program for r-domination problems over tree decompositions. In *IPEC*, volume 63 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

8 Jean Cardinal, Stefan Langerman, and Eythan Levy. Improved approximation bounds for edge dominating set in dense graphs. *Theor. Comput. Sci.*, 410(8-10):949–957, 2009.

9 Miroslav Chlebík and Janka Chlebíková. Approximation hardness of edge dominating set problems. *Journal of Combinatorial Optimization*, 11(3):279–290, 2006.

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

11 Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Trans. Algorithms*, 1(1):33–47, 2005.

12 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *STOC*, pages 624–633. ACM, 2014.

13 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness i: Basic results. *SIAM J. Comput.*, 24(4):873–921, 1995.

14 Rodney G. Downey and Michael R. Fellows. Parameterized Computational Feasibility. In *Feasible Mathematics II*, pages 219–244, 1995.

15 David Eisenstat, Philip N. Klein, and Claire Mathieu. Approximating k-center in planar graphs. In *SODA*, pages 617–627. SIAM, 2014.

16 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009.

17 Henning Fernau. edge dominating set: Efficient Enumeration-Based Exact Algorithms. In *Parameterized and Exact Computation*, pages 142–153. Springer Berlin Heidelberg, 2006.

18 Fedor V. Fomin, Serge Gaspers, Saket Saurabh, and Alexey A. Stepanov. On two techniques of combining branching and treewidth. *Algorithmica*, 54(2):181–207, 2009.

19 Toshihiro Fujito and Hiroshi Nagamochi. A 2-approximation algorithm for the minimum weight edge dominating set problem. *Discrete Applied Mathematics*, 118(3):199–207, 2002.

20 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979.

21 Torben Hagerup. Kernels for edge dominating set: Simpler or smaller. In *MFCS*, volume 7464 of *Lecture Notes in Computer Science*, pages 491–502. Springer, 2012.

22 Tesshu Hanaka, Naomi Nishimura, and Hirotaka Ono. On directed covering and domination problems. In *ISAAC*, volume 92 of *LIPIcs*, pages 45:1–45:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

23 Frank Harary and Robert Z Norman. Some properties of line digraphs. *Rendiconti del Circolo Matematico di Palermo*, 9(2):161–168, 1960.

24 Joseph D. Horton and Kyriakos Kilakos. Minimum edge dominating sets. *SIAM J. Discret. Math.*, 6(3):375–387, 1993.

25 Ken Iwaide and Hiroshi Nagamochi. An improved algorithm for parameterized edge dominating set problem. *J. Graph Algorithms Appl.*, 20(1):23–58, 2016.

26 Stephan Kreutzer and Siamak Tazari. Directed nowhere dense classes of graphs. In *SODA '12*, pages 1552–1562, 2012.

27 Dana Moshkovitz. The projection games conjecture and the np-hardness of ln n-approximating set-cover. *Theory of Computing*, 11:221–235, 2015.

28 Yury L. Orlovich, Alexandre Dolgui, Gerd Finke, Valery S. Gordon, and Frank Werner. The complexity of dissociation set problems in graphs. *Discrete Applied Mathematics*, 159(13):1352–1366, 2011.

29 Richard Schmied and Claus Viehmann. Approximating edge dominating set in dense graphs. *Theor. Comput. Sci.*, 414(1):92–99, 2012.

30 Mingyu Xiao, Ton Kloks, and Sheung-Hung Poon. New parameterized algorithms for the edge dominating set problem. *Theor. Comput. Sci.*, 511:147–158, 2013.

31 Mingyu Xiao and Shaowei Kou. Exact algorithms for the maximum dissociation set and minimum 3-path vertex cover problems. *Theor. Comput. Sci.*, 657:86–97, 2017.

32 Mingyu Xiao and Hiroshi Nagamochi. A refined exact algorithm for edge dominating set. *Theor. Comput. Sci.*, 560:207–216, 2014.

33 Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM J. Comput.*, 10(2):310–327, 1981.

34 Mihalis Yannakakis and Fanica Gavril. Edge dominating sets in graphs. *SIAM Journ. on Applied Mathematics*, 38(3):364–372, 1980.

Interval-Like Graphs and Digraphs

Pavol Hell[1]
School of Computing Science, Simon Fraser University
Burnaby, B.C., Canada V5A 1S6
pavol@sfu.ca

Jing Huang
Mathematics and Statistics, University of Victoria
Victoria, B.C., Canada V8W 2Y2
huangj@uvic.ca

Ross M. McConnell
Computer Science Department, Colorado State University
Fort Collins, CO 80523-1873
rmm@cs.colostate.edu

Arash Rafiey
Mathematics and Computer Science, Indiana State University
Terre Haute, IN 47809
arash.rafiey@indstate.edu

Abstract

We unify several seemingly different graph and digraph classes under one umbrella. These classes are all, broadly speaking, different generalizations of interval graphs, and include, in addition to interval graphs, adjusted interval digraphs, threshold graphs, complements of threshold tolerance graphs (known as 'co-TT' graphs), bipartite interval containment graphs, bipartite co-circular arc graphs, and two-directional orthogonal ray graphs. (The last three classes coincide, but have been investigated in different contexts.) This common view is made possible by introducing reflexive relationships (loops) into the analysis. We also show that all the above classes are united by a common ordering characterization, the existence of a min ordering. We propose a common generalization of all these graph and digraph classes, namely *signed-interval digraphs*, and show that they are precisely the digraphs that are characterized by the existence of a min ordering. We also offer an alternative geometric characterization of these digraphs. For most of the above graph and digraph classes, we show that they are exactly those signed-interval digraphs that satisfy a suitable natural restriction on the digraph, like having a loop on every vertex, or having a symmetric edge-set, or being bipartite. For instance, co-TT graphs are precisely those signed-interval digraphs that have each edge symmetric. We also offer some discussion of future work on recognition algorithms and characterizations.

2012 ACM Subject Classification Mathematics of computing → Combinatoric problems, Mathematics of computing → Graph theory

Keywords and phrases graph theory, interval graphs, interval bigraphs, min ordering, co-TT graph

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.68

[1] The author was supported by an NSERC (Canada) Discovery Grant

43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018).
Editors: Igor Potapov, Paul Spirakis, and James Worrell; Article No. 68; pp. 68:1–68:13

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

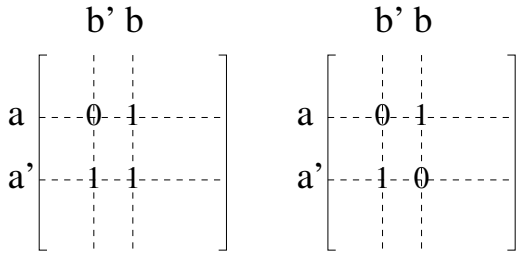


Figure 1 A min ordering of a digraph is an ordering of the vertices such that neither of the depicted submatrices occurs in the corresponding adjacency matrix.

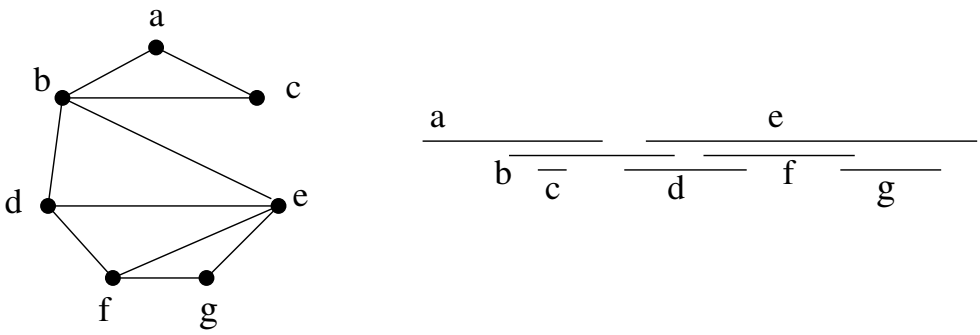


Figure 2 An interval graph and corresponding interval model. There is an implicit loop at each vertex.

1 Introduction

A digraph H is *reflexive* if each $vv \in E(H), v \in V(H)$ (H has all loops); *irreflexive* if no $vv \in E(H)$ (H has no loops); and *symmetric* if $ab \in E(H)$ implies $ba \in E(H)$. In this paper, we shall treat both graphs and digraphs; for simplicity we view graphs as symmetric digraphs. (Thus, graphs can have loops, and irreflexive graphs are loopless.) Loops play an important role in this paper, and this is not common in the literature on graph classes that we consider. They allow us to view several seemingly unrelated graph classes through a common lens.

A *min ordering* of a digraph H is a linear ordering $<$ of the vertices of H, so that $ab \in E(H), a'b' \in E(H)$ and $a < a', b' < b$ implies that $ab' \in E(H)$. In other words, a min ordering is an ordering of the vertices such that when the rows and columns of the adjacency matrix are ordered in this way, neither the matrix whose rows are 01 and 11 nor the matrix whose rows are 01 and 10 appears as a submatrix. (See Figure 1.) Note that the presence or absence of loops (1's on the diagonal of the adjacency matrix) can affect whether the graph has a min ordering.

Our goal in this paper is to promote a class of digraphs (or 0,1-matrices) that is a broad generalization of interval graphs and that retains some of the desirable structural properties of interval graphs. A graph H is an *interval graph* if it is the intersection graph of a family of intervals on the real line, i.e., if there exists a family of intervals $\{[x_v, y_v] | v \in V(H)\}$ such that $uv \in E(H)$ if and only if $[x_u, y_u] \cap [x_v, y_v] \neq \emptyset$. The family of intervals is an *interval model* of H. (See Figure 2.) We note that the definition implies that an interval graph is reflexive. A related concept for bipartite graphs is as follows. A bipartite graph H with parts A, B is an *interval bigraph* if there are intervals $\{[x_a, y_a], a \in A\}$, and $\{[x_b, y_b], b \in B\}$, such that for $a \in A$ and $b \in B$, $ab \in E(H)$ if and only if $[x_a, y_a] \cap [x_b, y_b] \neq \emptyset$.

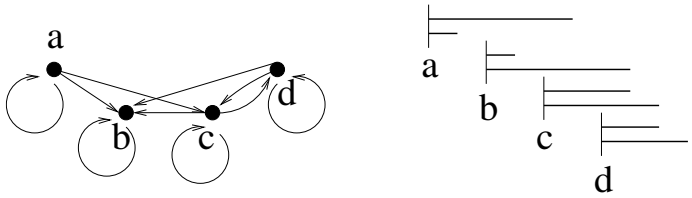


Figure 3 An adjusted interval digraph and a corresponding adjusted interval model. The source interval for each vertex is the upper one.

Interval graphs are important in graph theory and in applications, and are distinguished by several elegant characterizations and efficient recognition algorithms [3, 8, 11, 13, 16, 24, 31]. For this reason, there have been attempts to extend the concept to digraphs [29], with mixed success. (Many of the desirable structural properties are absent.) More recently a more restricted class of digraphs has been found to offer a nicer generalization of interval graphs; these are the adjusted interval digraphs [10]. A digraph H is an *adjusted interval digraph* if there are two families of real intervals, the *source intervals* $\{[x_v, y_v]|v \in V(H)\}$ and the *sink intervals* and $\{[x_v, z_v]|v \in V(H)\}$ such that $uv \in E(H)$ if and only if the source interval for u intersects the sink interval for v. (See Figure 3.) This differs from the class in [29] in that the left endpoint, x_v, must be shared by the two intervals $[x_v, y_v]$ and $[x_v, z_v]$ assigned to v; they are "adjusted." The interval graphs are the special case where $[x_v, y_v] = [x_v, z_v]$ for each $v \in V(H)$. An *adjusted interval model* of H is a set of source and sink intervals that represent H in this way.

Adjacency on a set of intervals can also be defined by interval containment. A graph is a *containment graph of intervals* [31] if there is a family of intervals $\{[x_v, y_v]|v \in V(H)\}$ on the real line such that $uv \in E(H)$ if and only if one of $[x_u, y_u]$ and $[x_v, y_v]$ contains the other. A graph is a containment graph of intervals if and only if it and its complement are both *transitively orientable*, thus if and only if it is a *permutation graph* [31].

For this paper, a more relevant class is a bipartite version of this concept. A bipartite graph H with parts A, B is an *interval containment bigraph* [31] if there are sets of intervals $\{I_a|a \in A\}$, and $\{J_b|b \in B\}$, such that $ab \in E(H)$ if and only if $J_b \subseteq I_a$. These graphs have been studied, from the point of view of another geometric representation, as two-directional orthogonal ray graphs [30]. A bipartite graph H with parts A and B is called a *two-directional orthogonal ray graph* if there exists a set $\{U_a, a \in A\}$ of upwards vertical rays, and a set $\{R_b, b \in B\}$ of horizontal rays to the right such that $ab \in E(H)$ if and only if $U_a \cap R_b \neq \emptyset$. It is known that a bipartite graph is an interval containment graph if and only if it is a two-directional orthogonal ray graph [22], and if and only if its complement is a circular arc graph [9].

It is sometimes convenient to view bipartite graphs as digraphs, with all edges oriented from part A to part B; thus we speak of a *bipartite interval containment digraph*, a *bipartite interval digraph*, or a *two-directional orthogonal ray digraph*. In general, a *bipartite digraph* is a bipartite graph with parts A and B and all arcs being oriented from A to B.

There is an interesting intermediate concept that uses both intersection and containment of intervals to define adjacency. An interval model of an interval graph G can be viewed as two mappings $\{v \to x_v|v \in V(H)\}$ and $\{v \to y_v|v \in V(H)\}$ such that $x_v \leq y_v$ for each $v \in V(H)$, and such that $uv \in E(H)$ if and only if $y_v \leq x_u$ and $y_u \leq x_v$. The constraint $x_v \leq y_v$ comes from the need for $[x_v, y_v]$ to be an interval. The proposition that two intervals intersect is the same as $x_v \leq y_u$ and $x_u \leq y_v$, since this means that neither interval lies entirely to the right of the other.

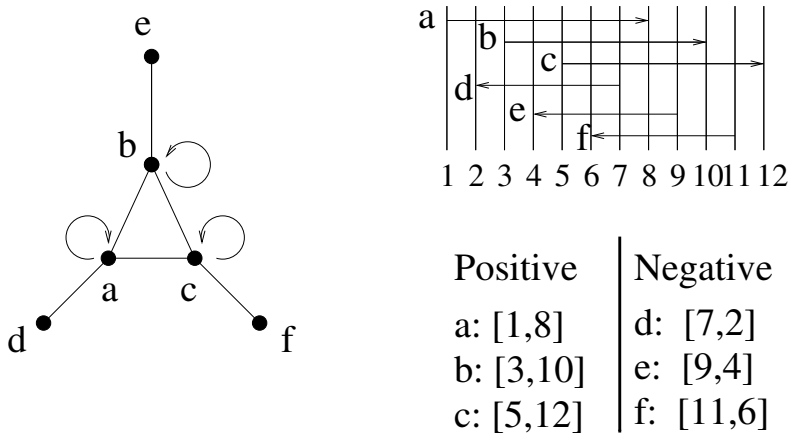


Figure 4 A co-TT graph and a corresponding co-TT model; ab is an edge since $1 \leq 10$ and $3 \leq 8$, ad is an edge since $1 \leq 2$ and $7 \leq 8$. However, bd is not an edge: although $7 \leq 10$, 3 is not less than or equal to 2. The example of this figure is one of the well-known minimal graphs that are not interval graphs, illustrating that the interval graphs are a proper subclass of the co-TT graphs.

A generalization of interval models is obtained by dropping the constraint $x_v \leq y_v$ in this formulation. To develop the motivation for this, we start with the complements of threshold tolerance graphs. A graph H is a *threshold tolerance graph* [27] if its vertices v can be assigned weights w_v and tolerances t_v so that ab is an edge of H if and only if $w_a + w_b > t_a$ or $w_a + w_b > t_b$. (When all t_v are equal, this defines a better known class of *threshold graphs* [5].) *Co-threshold tolerance ('co-TT') graphs* are complements of threshold tolerance graphs. Equivalently, a graph H is a co-TT graph, if there exist real numbers $x_v, y_v, v \in V(H)$, such that $ab \in E(H)$ if and only if $x_a \leq y_b$ and $x_b \leq y_a$ [14]. This differs from the definition of interval graphs in that it is no longer required that $x_v \leq y_v$, illustrating the motivation for dropping the constraint in this case. (See Figure 4.) That these are precisely the co-TT graphs is easily seen by letting $x_v = w_v$ and $y_v = t_v - w_v$. The two mappings $v \to x_v$ and $v \to y_v$, are called the *co-TT model* of H.

One view of a co-TT model is that there are now intervals whose 'beginning' x_v may come after their 'end' y_v. In other words, we may have 'intervals' $[x_v, y_v]$ with $y_v < x_v$. We may view a co-TT model as consisting of intervals $[x_v, y_v], v \in V(H)$, some of which go in the positive direction (have $x_v \leq y_v$) and others go in the negative direction (have $x_v > y_v$). We speak of *positive* or *negative* intervals, and *positive* or *negative* vertices that correspond to them. (In the literature [14, 12, 17, 27], the direction is denoted by *colors* of the intervals: positive intervals, and vertices, are colored *blue*, and negative intervals, and vertices, are colored *red*.) The above definition of adjacency has interesting consequences. Two positive vertices are adjacent if and only if they intersect; in particular, each positive vertex has a loop. Two negative vertices are never adjacent; in particular negative vertices have no loops. Finally, a positive vertex u corresponding to a positive interval $[a, b]$ and a negative vertex v corresponding to a negative interval $[c, d]$ are adjacent if and only if $[d, c]$ is contained in $[a, b]$ (i.e., $a \leq d \leq c \leq b$). We also use the following signed shorthand, which will be useful later: a positive vertex or interval will be called a $+$*-vertex* or $+$*-interval* respectively, and a negative vertex or interval will be called a $-$*-vertex* or $-$*-interval* respectively. It follows from the above discussion that in a co-TT graph, the $+$-vertices induce a reflexive interval graph, the $-$-vertices form an independent set, and the edges between the $+$-vertices and the $-$-vertices form a bipartite interval containment graph.

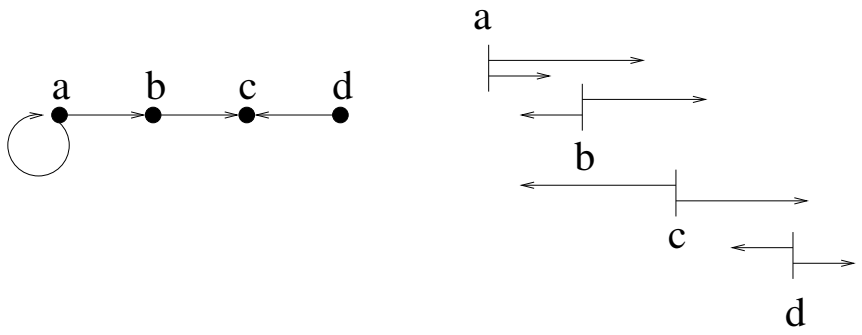


Figure 5 A signed interval digraph and a corresponding signed interval model. The source interval for each vertex is the upper one. There is a loop at a because its positive source interval intersects its positive sink interval. There is an edge from a to b because a's positive source interval contains b's negative sink interval, an edge from b to c because b's positive source interval intersects c's positive sink interval, and an edge from d to c because d's negative source interval is contained in c's positive sink interval.

Note that co-TT graphs are a generalization of interval graphs; the interval graphs are those co-TT graphs where all vertices are positive. In other words, they are the reflexive co-TT graphs.

2 Signed Interval Digraphs

We have now seen extensions of interval graphs in two different directions. First, by taking two (adjusted) intervals instead of just one interval, we were able to extend the definition from reflexive graphs to reflexive digraphs. Second, by admitting intervals $[a, b]$ that go in the negative direction (have $b < a$), we were able to extend the definition from reflexive graphs to graphs that have some vertices with loops and others without. Both these generalizations have proved very fruitful [10, 8, 10, 21, 14, 12, 17, 27].

We now define a new class of digraphs that unifies these extensions. A digraph H is a *signed-interval digraph* if there exist three mappings from $V(H)$ to the real line, $v \to x_v, v \to y_v$, and $v \to z_v$, such that $uv \in E(H)$ if and only if $x_u \leq z_v$ and $x_v \leq y_u$. We call the three mappings $v \to x_v, v \to y_v$, and $x \to z_v$ a *signed-interval model* of H. Alternatively, a signed interval model is obtained in by assigning, for each $v \in V(H)$ a *source interval* $[x_v, y_v]$ and a *sink interval* $[x_v, z_v]$, such that $uv \in E(H)$ if and only if $x_u \leq z_v$ and $x_v \leq y_u$. (See figure 5.) Since it is possible that $x_v > y_v$ and/or $x_v > z_v$, each of $[x_v, y_v]$ and $[x_v, z_v]$ can be negative or positive. Since the source interval and sink interval for v share the endpoint x_v, we retain the property that the intervals are adjusted.

Signed-interval digraphs with all intervals positive, are reflexive, and are adjusted interval digraphs. Signed-interval digraphs with $y_v = z_v$, for all $v \in V(H)$, are symmetric, and are co-TT graphs. Signed-interval digraphs that satisfy both conditions, i.e., with all $x_v \leq y_v = z_v, v \in V(H)$, are interval graphs. Furthermore, we show below that there are no reflexive signed-interval digraphs other than adjusted interval digraphs, no symmetric signed-interval digraphs other than co-TT graphs, and no reflexive and symmetric signed-interval digraphs other than interval graphs.

The structure of signed-interval digraphs can be described in a language similar to what was used for co-TT graphs. Let H be a signed-interval digraph and consider a signed-interval model of H given by the ordered pairs (I_v, J_v) of intervals where $I_v = [x_v, y_v]$ and $J_v = [x_v, z_v]$. For $\alpha, \beta \in \{+, -\}$, we say a vertex v is of type (α, β) if I_v is an α-interval

and J_v is a β-interval. The subdigraph of H induced by $(+,+)$-vertices is an adjusted interval digraph. The $(-,-)$-vertices of H form an independent set. The arcs between the $(+,-)$- and $(-,-)$-vertices form a bipartite interval containment digraph. The arcs between the $(-,+)$- and $(-,-)$-vertices also form a bipartite interval containment digraph. Similar properties hold for the other parts and their connections.

We emphasize that our definition of co-TT graphs differs from the standard definition [12, 14, 27]. In the standard definition, the condition $ab \in E(H) \iff x_a \leq y_b$ and $x_b \leq y_a$ is applied only for $a \neq b$, and so the graphs have no loops. Thus a graph under the standard interpretation is co-TT if and only if with a suitable addition of loops it is co-TT under our definition above. This difference is not important as it was shown in [13] that if a graph H is co-TT (in the standard sense), then it has a co-TT model with negative intervals for all simplicial vertices without true twins and all other intervals positive. Thus there is an easy translation between the co-TT graphs as defined here and the standard irreflexive co-TT graphs: namely, loops are to be placed on all vertices other than simplicial vertices without true twins.

3 Min Orderings

Interval graphs, adjusted interval digraphs, co-TT graphs, and two-directional orthogonal ray digraphs all have min orderings when care is taken to specify which vertices have loops and which do not. [8, 10, 18, 30].

Min orderings are a useful tool for graph homomorphism problems. A *homomorphism* of a digraph G to a digraph H is a mapping $f : V(G) \to V(H)$ such that $f(u)f(v) \in E(H)$ whenever $uv \in E(G)$. If a digraph H has a min ordering, there is a simple polynomial-time algorithm to decide if a given input graph G admits a homomorphism to a fixed digraph H [15, 20]. In fact, the algorithm is well known in the AI community as the arc-consistency algorithm [20]; it is easy to see that it also solves *list homomorphism* problems, where we seek a homomorphism of input G to fixed H taking each vertex of G to one of a 'list' of allowed images. In fact, many (but not all) homomorphism and list homomorphism problems that can be solved in polynomial time can be solved using arc-consistency with respect to a min ordering.

Graph and digraph homomorphism problems are special cases of constraint satisfaction problems. A general tool for solving polynomial time solvable constraint satisfaction problems are the so-called polymorphisms [4]. Without going into the technical details, we mention that min-orderings are equivalent to conservative semilattice polymorphisms [10].

We prove below that a digraph has a min ordering if and only if it is a signed-interval digraph. We also give another geometric characterization of signed-interval digraphs, as bi-arc digraphs. We show that a *reflexive* signed-interval digraphs are precisely adjusted interval digraphs, that *symmetric* signed-interval digraphs are precisely co-TT graphs, that *reflexive and symmetric* signed-interval digraphs are precisely interval graphs, and that *bipartite* signed-interval digraphs are precisely two-directional ray graphs.

The main result of this section is the following.

▶ **Theorem 1.** *A digraph admits a min ordering if and only if it is a signed-interval digraph.*

Before embarking on the proof we offer an alternate definition of a min ordering. Consider any linear ordering $<$ of $V(H)$. To this ordering, we prepend an intial element α, which is a place holder and not a vertex. Thus, $\alpha < x$ for each vertex x. We denote by $O(a)$ the last

vertex b (in the order $<$), such that b is an out-neighbor of a (i.e., such that $ab \in E(H)$), or α if a has no out-neighbor. Similarly, for each vertex b, we denote by $I(b)$ the last vertex a such that a is an in-neighbor of b (i.e., such that $ab \in E(H)$), or α if a has no in-neighbor.

▶ **Proposition 2.** *A linear ordering $<$ of $V(H)$ is a min ordering of a digraph H if and only if the following property holds:*

$$ab \in E(H) \text{ if and only if } a \leq I(b) \text{ and } b \leq O(a).$$

Proof. Suppose first that $<$ is a min ordering of H with α prepended. If $ab \in E(H)$, then by the definition of $O(a), I(b)$ we have $a \leq I(b)$ and $b \leq O(a)$. On the other hand, let $a \leq I(b)$ and $b \leq O(a)$. Note that if $a = I(b)$ or $b = O(a)$ we have $ab \in E(H)$ also by definition. Therefore it remains to consider vertices a, b such that $a < c = I(b)$ and $b < d = O(a)$. Then $ad, cb \in E(H)$ and the min ordering property implies that $ab \in E(H)$. This proves the property.

Conversely, assume that $<$ is a linear ordering of $V(H)$ with α prepended and that the property holds for $<$. We claim it is a min ordering of H. Otherwise some $ab \in E(H), a'b' \in E(H), a < a', b' < b$ would have $ab' \notin E(H)$. This is a contradiction, since we have $a < a' \leq I(b')$ and $b' < b \leq O(a)$. ◀

We proceed to prove the theorem.

Proof. Suppose $<$ is a min ordering of a digraph H with α prepended. We represent each vertex $v \in V(H)$ by the mappings $v \to v, v \to O(v), v \to I(v)$. In other words, v is represented by the two intervals $[v, O(v)]$ and $[v, I(v)]$. It follows from Proposition 2 that $ab \in E(H)$ if and only if $a \leq I(b)$ and $b \leq O(a)$. Thus H is a signed-interval digraph.

Conversely, suppose we have the three mappings $v \to x_v, v \to y_v, v \to z_v$ from $V(H)$ to the real line, such that $ab \in E(H)$ if and only if $x_a \leq z_b$ and $x_b \leq y_a$. Without loss of generality we may assume the points $\{x_v | v \in V(H)\}$ are all distinct. Then we claim that the left to right ordering of the points x_v yields a min ordering $<$ of H. (Specifically, we define $a < b$ if and only if x_a precedes x_b.) Consider now $ab \in E(H), a'b' \in E(H)$, with $a < a', b' < b$. This means that $x_a < x_{a'} \leq z_{b'}$ and $x_{b'} < x_b \leq y_a$, whence we must have $ab' \in E(H)$. ◀

4 An alternate geometric representation

Digraphs that admit a min ordering have another geometric representation. Let C be a circle with two distinguished points (the *poles*) N and S, and let H be a digraph. Let $I_v, v \in V(H)$ and $J_v, v \in V(H)$ be two families of arcs on C such that each I_v contains N but not S, and each J_v contains S but not N. We say that the families I_v and J_v are *consistent* if they have the same clockwise order of their clockwise ends, i.e., the clockwise end of I_a precedes in the clockwise order the clockwise end of I_b if and only if the clockwise end of J_a precedes in the clockwise order the clockwise end of J_b. Suppose two families I_v, J_v are consistent; we define an ordering $<$ on $V(H)$ where $a < b$ if and only if the clockwise end of I_a precedes in the clockwise order the clockwise end of I_b; we call $<$ the ordering *generated by* the consistent families I_v, J_v.

A *bi-arc model* of a digraph H is a consistent pair of families of circular arcs, $I_v, J_v, v \in V(H)$, such that $ab \in E(H)$ if and only if I_a and J_b are disjoint. A digraph H is called a *bi-arc digraph* if it has a bi-arc model.

▶ **Theorem 3.** *A digraph H admits a min ordering if and only if it is a bi-arc digraph.*

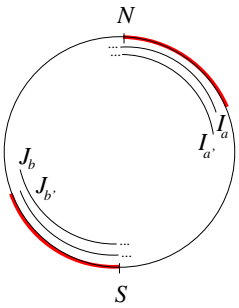


Figure 6 Illustration for the proof of Theorem 3.

Proof. Suppose I_v, J_v form a bi-arc model of H. We claim that the ordering $<$ generated by I_v, J_v is a min ordering of H. Indeed, suppose $a < a'$ and $b' < b$ have $ab, a'b' \in E(H)$. Then $I_{a'}$ spans the area of the circle between N and the clockwise end of I_a, and J_b spans the area of the circle between S and the clockwise end of $J_{b'}$. (See Figure 1.) This implies that I_a and $J_{b'}$ are disjoint: indeed, the counterclockwise end of I_a is blocked from reaching $J_{b'}$ by J_b (since $ab \in E(H)$), and the counterclockwise end of $J_{b'}$ is blocked from reaching I_a by $I_{a'}$ (since $a'b' \in E(H)$). (The clockwise ends are fixed by the ordering $<$.)

Conversely, suppose $<$ is a min ordering of H. We construct families of arcs I_v and J_v, with $v \in V(H)$, as follows. The intervals I_v will contain N but not S, the intervals J_v will contain S but not N. The clockwise ends of I_v are arranged in clockwise order according to $<$, as are the clockwise ends of J_v. The counterclockwise ends will now be organized so that $I_v, J_v, v \in V(H)$, becomes a bi-arc model of H. For each vertex $v \in V(H)$, we define $O(v)$ and $I(v)$ as in the proof of Theorem 1. Then we assign the counterclockwise endpoint of I_v to be N if v has no out-neighbors, or else extend I_v counterclockwise as far as possible without intersecting $J_{O(v)}$, and assign the the counterclockwise endpoint of each J_v to be S if v has no in-neighbors, or else extend J_v counterclockwise as far as possible without intersecting $I_{I(v)}$. We claim this is a bi-arc model of H. Clearly, if $b > O(a)$, then I_a intersects J_b by the construction, and similarly for $a > I(b)$ we have J_b intersecting I_a. This leaves disjoint all pairs I_a, J_b such that $a \leq I(b)$ and $b \leq O(a)$; since $aO(a), I(b)b \in E(H)$, the definition of min ordering implies that $ab \in E(H)$, as required. ◀

▶ **Corollary 4.** *The following statements are equivalent for a digraph H.*

- *H has a min ordering*
- *H is a signed-interval digraph*
- *H is a bi-arc digraph.*

5 0,1-Matrices and bipartite graphs

Irreflexive graphs with at least one edge do not admit a min ordering, since the vertices that are not reflexive form an independent set. However, in the special case of bipartite graphs, a version of min ordering has been studied, and has yielded interesting examples. We will describe that version below, but we first want to explain how to view that definition as a special case of min ordering as defined here.

A useful perspective on min orderings is obtained by considering 0,1-matrices. Square 0,1-matrices naturally correspond to adjacency matrices of digraphs. Let a *simultaneous permutation* of rows and columns of a matrix be one where the permutation of the rows is the

same as the permutation of the columns. An *independent permutation* of rows and columns allows the permutation of the rows to be different from the permutation of the columns.

Let L be the two by two matrix with rows 01 and 11, and let K be the two by two matrix with rows 01 and 10. (These have been given other names in the literature, up to a simultaneous permutation of rows and columns they are the gamma matrix, and the identity matrix.) A matrix M is called K, L*-free* if it does not contain K or L as a submatrix. If M is the adjacency matrix of a digraph H, and if the rows and columns of H are in the order $<$, then M is K, L-free if and only if $<$ is a min ordering. We call a M a *min-orderable matrix* if its rows and columns can be simultaneously permuted to produce a K, L-free matrix. A digraph has a min ordering if and only if its adjacency matrix is min-orderable.

Therefore, we can say much about matrices that are min-orderable.

▶ **Theorem 5.** *A square 0,1-matrix is min-orderable if and only if it is the adjacency matrix of a signed-interval digraph.*

Another natural interpretation of a 0,1-matrix is that it represents adjacencies in a bipartite graph, with rows corresponding to one part and columns to the other part. The *bi-adjacency matrix* of a bipartite graph H with marts A, B has its i, j-th entry is 1 if and only if the i-th vertex in A is adjacent to the j-th vertex in B. Note that for this interpretation it is not required that the matrix be square. For matrices that are not necessarily square, we can still ask for independent permutations of rows and columns, to produce a K, L-free matrix. This suggests a definition of min ordering for bipartite graphs as follows. A *min ordering of a bipartite graph* H with parts A and B is a linear ordering $<_A$ of A and a linear ordering $<_B$ of B so that for any $a, a' \in A, b, b' \in B$ such that $ab \in E(H), a'b' \in E(H)$ and $a < a', b' < b$ we have $ab' \in E(H)$. This is the definition that has been used in the literature; it is clear how it avoids the problems of the general definition.

There is a simple transformation that connects the two interpretations of 0,1-matrices. For a matrix M with k rows and ℓ columns, we define the $(k + \ell)$ by $(k + \ell)$ square matrix M^+ to contain the matrix M in the first k rows and the last ℓ columns, with 0 everywhere else. Then a simultaneous row/column permutation of M^+ corresponds to independent row and column permutations of M. Note that the square matrix M^+ is an adjacency matrix of the digraph obtained from H by directing all edges from H from the first part to the second part. Thus to view the special definition of a min ordering for bipartite graphs as a particular case of the general definition, it suffices to view bipartite graphs as digraphs with all edges oriented from the first part to the second part. We shall say that H is a *bipartite digraph* if it is obtained from a bipartite graph in this way.

A robust class of bipartite graphs is relevant for our discussion. A bipartite graph H with parts A and B is called a *two-directional orthogonal ray graph* if there exists a set $U_a, a \in A$, of upwards vertical rays, and a set $R_b, b \in B$, of horizontal rays to the right such that $ab \in E(H)$ if and only if $U_a \cap R_b \neq \emptyset$. Note that we may, if needed, view a two-directional orthogonal ray graph as a bipartite digraph, with all edges oriented from (say) vertical rays to horizontal rays.

The following theorem is obtained by a combination of results from [9, 22, 30].

▶ **Theorem 6.** *The following statements are equivalent for a bipartite graph H.*

- *H is a two-directional orthogonal ray graph*
- *the complement of H is a circular arc graph*
- *H is an interval containment graph.*

Matrices that can be permuted to avoid small submatrices have been of much interest [1, 23, 25]. This of course corresponds to characterizations of digraphs by forbidden ordered subgraphs [7, 19]. Our focus was on K, L-free matrices. Let the matrix Γ be obtained from

L by simultaneously exchanging the rows and columns; i.e., Γ has rows 11, 10. Let I be the two by two identity matrix. It is easy to see that considering I, Γ-free matrices is equivalent to considering K, L-free matrices, as the permutation that simultaneously reverses rows and columns of matrix M transforms a I, Γ-free matrix to a K, L-free matrix and vice versa. Matrices that are Γ-free have been intensively studied [1, 25], cf. [31]. A bipartite graph H is *chordal bipartite* if it contains no induced cycle other than C_4. A reflexive graph is *strongly chordal* if it contains no induced cycle or induced trampoline. (A *trampoline* is a complete graph on $v_0, v_1, v_2, \ldots, v_{k-1}, k > 2$ with vertices $u_i, i = 0, 1, \ldots, k-1$, each only adjacent to v_i, v_{i+1}, subscripts computed modulo k.) The adjacency matrix of a reflexive graph H can be made Γ-free by simultaneous row / column permutations if and only if H is strongly chordal; the bi-adjacency matrix of a bipartite graph H can be made Γ-free by independent permutations of rows and columns if and only if H is chordal bipartite [1]. These results amount to forbidden structure characterizations of matrices that are permutable (by simultaneous or independent row and column permutations) to a Γ-free format. Algorithms to recognize such matrices efficiently have been given in [25, 28]. For L-free matrices, or equivalently, for I-free matrices a forbidden structure characterization is given in [17]. An $O(n^2)$ recognition algorithm is claimed in [2], cf. [31].

6 Special cases

We now explore what min orderings look like in the special cases we have discussed, namely reflexive graphs, reflexive digraphs, undirected graphs, and bipartite graphs. The results are all corollaries of Theorem 1 and Proposition 2.

▶ **Corollary 7.** *A reflexive graph H is a signed-interval digraph if and only if it is an interval graph.*

▶ **Corollary 8.** *A reflexive digraph H is a signed-interval digraph if and only if it is an adjusted interval digraph.*

Next we focus on symmetric digraphs, i.e., graphs.

▶ **Corollary 9.** *A graph H is a signed-interval digraph, i.e., has a min ordering, if and only if it is a co-TT graph.*

Proof. Consider a co-TT model of H, given by the mappings $v \to x_v, v \to y_v$, setting the third mapping $v \to z_v$ with each $z_v = y_v$, yields a signed-interval digraph model of H. Conversely, assume H is a graph, i.e., a symmetric digraph, that is a signed-interval digraph. Let $<$ be a min ordering of H; we again have $O(v) = I(v)$ for all vertices v. We claim that the mappings $v \to x_v = v, v \to y_v = O(v)$ define a co-TT model. Indeed, from Proposition 2 we have $ab \in E(H)$ if and only if $a \leq O(b) = y_b$ and $b \leq O(a) = y_a$, as required. ◀

Finally, for bipartite graphs we have the following result, stated for convenience in the language of bipartite digraphs. Note that this has only one consequence, that is, when we consider an edge $ab \in E(H)$ we always assume $a \in A$ and $b \in B$.

▶ **Corollary 10.** *A bipartite digraph H is a signed-interval digraph, i.e., has a min ordering, if and only if it is a two-directional orthogonal ray graph.*

Note that two-directional orthogonal ray graphs themselves have two other equivalent characterizations in Theorem 6. The characterization of two-directional orthogonal ray graph by the existence of a min ordering is also observed in [18, 30].

Proof. Suppose H has a signed-interval model given by the three mappings $v \to x_v, v \to y_v, v \to z_v$ such that $ab \in E(H)$ if and only if $x_a \leq z_b$ and $x_b \leq y_a$. We construct a two-directional ray model for H as follows. For each $a \in A$, we take an upwards vertical ray starting in the point P_a with x-coordinate equal to y_a and with y-coordinate equal to x_a. For each $b \in B$, we take a horizontal ray to the right, starting in the point Q_b with x-coordinate x_b and y-coordinate z_b. Now P_a intersects Q_b if and only if $x_b \leq y_a$ and $x_a \leq z_b$, i.e., if and only if $ab \in E(H)$ as required.

Now suppose that H has a two-directional model, i.e., upwards vertical rays $U_a, a \in A$, and horizontal rays to the right $R_b, b \in B$, such that $ab \in E(H)$ if and only if $U_a \cap R_b \neq \emptyset$. We will prove that H has a min ordering, whence it is a signed-interval digraph by Theorem 1. We will define the orders $<$ on A and on B as follows. Assume the starting point of the vertical ray U_a has the (x, y)-coordinates (u_a, v_a), and the starting point of the horizontal ray R_b has the (x, y)-coordinates (r_b, s_b), for $a \in A$, and $b \in B$. It is easy to see that we may assume, without loss of generality, that all $u_a, a \in A$, and $r_b, b \in B$ are distinct, and similarly for $v_a, a \in A$ and $s_b, b \in B$. We define $a < a'$ in A if and only if $v_a < v'_a$, and define $b < b'$ in B if and only if $r_b < r_{b'}$. We show that this is a min ordering of the bipartite digraph H. Otherwise, some $ab \in E(H), a'b' \in E(H), a < a', b' < b$ have $ab' \notin E(H)$. There are two possibilities for $ab' \notin E(H)$; either $u_a < r_{b'}$ or $u_a > r_{b'}, v_a > s_{b'}$. In the former case, $U_a \cap R_b = \emptyset$, in the latter case $U_{a'} \cap R_{b'} = \emptyset$, contradicting the assumptions. ◀

7 Algorithms and characterizations

Interval graphs are known to have elegant characterization theorems [11, 24], cf. [13, 31] and efficient recognition algorithms [3, 6, 16]. Thus one might hope to be able to obtain similar results for their generalizations and digraph analogues. This is true for all the generalizations described in this paper, at least to some degree. In this section we summarize what is known.

The prototypical characterization of interval graphs is the theorem of Lekkerkerker and Boland [24]. In our language, it states that *a reflexive graph H is an interval graph if and only if it contains no asteroidal triple and no induced C_4 or C_5*. An *asteroidal triple* consists of three non-adjacent vertices such that any two are joined by a path not containing any neighbors of the third vertex. An equivalent characterization by the absence of a slightly less concise obstruction is given in [10]. *A reflexive graph H is an interval graph if and only if it contains no invertible pair.* An *invertible pair* is a pair of vertices u, v such that there exist two walks of equal length, P from u to v, and Q from v to u, where the i-th vertex of P is non-adjacent to the $(i+1)$-st vertex of Q (for each i), and also two walks of equal length R, S from v to u and u to v respectively, where the i-th vertex of R is non-adjacent to the $(i+1)$-st vertex of S (for each i). It is not difficult to see that an asteroidal triple is a special case of an invertible pair. A number of variants of the definition of an invertible pair have arisen [10, 12, 17, 18], and they have proved useful to give characterization theorems for various classes. It is proved in [10] that *a reflexive digraph is an adjusted interval digraph if and only if it contains no directed invertible pair.* A *directed version* of an invertible pair is defined in [10] in a manner similar to the above definition of an invertible pair. With yet another *labeled version* of an invertible pair, we have the following obstruction characterization of co-TT graphs: *a graph is a co-TT graph if and only if it contains no labeled invertible pair*, which follows from the characterization in [12] in terms of an interval ordering from [26]. For bipartite graphs, an analogous *bipartite version* of an invertible pair yields the following result. *A bipartite graph is a two-directional orthogonal ray graph if and only if it contains no bipartite invertible pair*, [18]. In fact, in [9] a stronger version is

shown: there is a bipartite analogue of an asteroidal triple, called an *edge-asteroid*, and *a bipartite graph is a two-directional orthogonal ray graph if and only if it contains no induced* 6*-cycle and no edge-asteroid*. Finally, in [21], there is an obstruction characterization for signed-interval digraphs, which is a little more technical than just an invertible pair, [21].

There is a long history of efficient algorithms for the recognition of interval graphs, many of them linear time, starting from [3] and culminating in [6]. A polynomial time algorithm for the recognition of adjusted interval digraphs is given in [10]. It is not known how to obtain a linear time, or even near-linear time algorithm. An $O(n^2)$ algorithm for the recognition of two-directional orthogonal ray graphs follows from Theorem 6 and [26]. A more efficient algorithm in this case is also not known. On the other hand, an $O(n^2)$ algorithm for the recognition of co-TT graphs has been given in [12]. In [21], a polynomial-time algorithm for the recognition of a signed-interval digraph is proposed. (A new version of [21] will be posted on arXiv soon.)

References

1 R. P. Anstee and M. Farber. Characterizations of totally balanced matrices. *J. Algorithms*, 5:215–230, 1984.

2 V. L. Beresnev and A. I. Davydov. On matrices with connectedness properties. *Upravlyaemye Sistemy*, 19:3–13, 1979.

3 K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Computer and System Sci.*, 13:335–379, 1976.

4 A. Bulatov, P. Jeavons, and A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Computing*, 34:720–742, 2005.

5 V. Chvátal and P. L. Hammer. Set-packing and threshold graphs. Technical report, Univ. Waterloo Res. Report, 1973.

6 D. G. Corneil, S. Olariu, and L. Stewart. The LBFS structure and recognition of interval graphs. *SIAM J. Discrete Math.*, 23:1905–1953, 2009.

7 P. Damaschke. Forbidden ordered subgraphs. In R. Bodendiek and R. Henn, editors, *Topics in Combinatorics and Graph Theory*, pages 219–229. Physika-Verlag HD, 1990.

8 T. Feder and P. Hell. List homomorphisms to reflexive graphs. *J. Combinatorial Theory B*, 72:236–250, 1998.

9 T. Feder, P. Hell, and J. Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19:487–505, 1999.

10 T. Feder, P. Hell, J. Huang, and A. Rafiey. Interval graphs, adjusted interval digraphs, and reflexive list homomorphisms. *Discrete Applied Mathematics*, 160:697–707, 2012.

11 D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific J. Math.*, 15:835–855, 1965.

12 P. Golovach, P. Heggerness, R. M. McConnell, V. F. dos Santos, J. P. Spinrad, and J. L. Szwarcfiter. On recognition of threshold tolerance graphs and their complements. *Discrete Applied Mathematics*, 216:171–180, 2017.

13 M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

14 M. C. Golumbic, N. L. Weingarten, and V. Limouzy. Co-TT graphs and a characterization of split co-TT graphs. *Discrete Applied Mathematics*, 165:168–174, 2014.

15 W. Gutjahr, E. Welzl, , and G. J. Woeginger. Polynomial graph-colourings. *Discrete Applied Mathematics*, 35:29–45, 1992.

16 M. Habib, R. McConnell, C. Paul, , and L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234:59–84, 2000.

17 P. Hell, J. Huang, R. M. McConnell, and J. Lin. Comparability graphs coverable by two cliques, and cocomparability bigraphs. *Manuscript*, 2018.

18 P. Hell, M. Mastrolilli, M. M. Nevisi, and A. Rafiey. Approximation of minimum cost homomorphisms. In *Algorithms - ESA 2012*, volume 7501 of *Lecture Notes in Computer Science*, pages 587–598. Springer, 2012.

19 P. Hell, B. Mohar, and A. Rafiey. Orderings without forbidden patterns. In A. S. Schulz and D. Wagner, editors, *Algorithms - ESA 2014*, volume 8737 of *Lecture Notes in Computer Science*. Springer, 2014.

20 P. Hell and J. Nešetřil. *Graph Homomorphisms*. Wiley, 2004.

21 P. Hell and A. Rafiey. Bi-arc digraphs and conservative polymorphisms, 2016. `arXiv: arXiv:1608.03368`.

22 J. Huang. Representation characterizations of chordal bipartite graphs. *J. Combinatorial Theory B*, 96:673–683, 2006.

23 B. Klintz, R. Rudolf, and G. J. Woeginger. Permuting matrices to avoid forbidden submatrices. *Discrete Applied Mathematics*, 60:223–248, 1995.

24 C. G. Lekkerkerker and J. C. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Math.*, 51:45–64, 1962.

25 A. Lubiw. Doubly lexical orderings of matrices. *SIAM J. Comput.*, 16:854–879, 1987.

26 R. M. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica*, 37:93–147, 2003.

27 C. L. Monma, B. Reed, and W. T. Trotter. Threshold tolerance graphs. *J. Graph Theory*, 12:343–362, 1988.

28 R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16:973–989, 1987.

29 M. Sen, S. Das, A. B. Roy, and D. B. West. Interval digraphs: an analogue of interval graphs. *J. Graph Theory*, 13:581–592, 1989.

30 A. M. Shresta, S. Tayu, and S. Ueno. On two-directional orthogonal ray graphs. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 1807–1810, 2010.

31 J. Spinrad. *Efficient Graph Representations*. Fields Institute Monographs, AMS, 2003.

Double Threshold Digraphs

Peter Hamburger
Department of Mathematics, Indiana-Purdue University
Fort Wayne, IN 46805, USA
hamburge@ipfw.edu

Ross M. McConnell
Department of Computer Science, Colorado State University
Fort Collins, CO 80523, USA
rmm@cs.colostate.edu

Attila Pór
Department of Mathematics, Western Kentucky University
Bowling Green, KY 42101
attila.por@wku.edu

Jeremy P. Spinrad
Department of Computer Science, Vanderbilt University
Nashville, TN 37235, USA
Jeremy.P.Spinrad@vanderbilt.edu

Zhisheng Xu
Department of Computer Science, Colorado State University
Fort Collins, CO 80523, USA
xuzs9298@cs.colostate.edu

Abstract

A *semiorder* is a model of preference relations where each element x is associated with a utility value $\alpha(x)$, and there is a threshold t such that y is preferred to x iff $\alpha(y) - \alpha(x) > t$. These are motivated by the notion that there is some uncertainty in the utility values we assign an object or that a subject may be unable to distinguish a preference between objects whose values are close. However, they fail to model the well-known phenomenon that preferences are not always transitive. Also, if we are uncertain of the utility values, it is not logical that preference is determined absolutely by a comparison of them with an exact threshold. We propose a new model in which there are two thresholds, t_1 and t_2; if the difference $\alpha(y) - \alpha(x)$ is less than t_1, then y is not preferred to x; if the difference is greater than t_2 then y is preferred to x; if it is between t_1 and t_2, then y may or may not be preferred to x. We call such a relation a (t_1, t_2) double-threshold semiorder, and the corresponding directed graph $G = (V, E)$ a (t_1, t_2) double-threshold digraph. Every directed acyclic graph is a double-threshold digraph; increasing bounds on t_2/t_1 give a nested hierarchy of subclasses of the directed acyclic graphs. In this paper we characterize the subclasses in terms of forbidden subgraphs, and give algorithms for finding an assignment of utility values that explains the relation in terms of a given (t_1, t_2) or else produces a forbidden subgraph, and finding the minimum value λ of t_2/t_1 that is satisfiable for a given directed acyclic graph. We show that λ gives a useful measure of the complexity of a directed acyclic graph with respect to several optimization problems that are NP-hard on arbitrary directed acyclic graphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases posets, preference relations, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.69

Related Version A full version of the paper is available at [6], `https://arxiv.org/abs/1702.06614`.

1 Introduction

A poset P can be identified with a transitive digraph on its elements. The poset $P = P(V, <)$ is a *semiorder* [10] if for some utility function $\alpha : V \to \mathbb{R}$ we have $u <_P v$ if and only if $\alpha(v) - \alpha(u) > 1$. Semiorders were introduced as a possible mathematical model for preference in the social sciences. A first possible model for preference is the *weak orders*, in which each element is assigned a utility value, such that u is preferred to v iff the value of u is greater than the value of v. This was viewed as too restrictive; many preference relationships cannot be modeled by a weak order. Semi-orders were designed to model imprecision in the valuation function; we may be indifferent between elements not only if they have exactly the same values, but also if the difference between the values is smaller than some threshold. There is a great deal of literature on the subject of semiorders and preference; see the books [2, 11].

Our original motivation for defining double-threshold digraphs comes from an attempt to deal with an issue in mathematical psychology. Intuitively, it is natural to think that preference is transitive; if one prefers a to b and b to c, then one "should" prefer a to c. However, a variety of evidence exists showing that preferences are not always transitive. This has led to a great deal of discussion; for a summary of this issue, see [3]. Viewpoints range from the idea that the intuitive notion that preference is transitive are simply wrong and must be thrown away entirely to questioning whether what was being measured in the non-transitive findings was really a preference relation. Between these two views, there has been work on finding mathematical models that explain non-transitive preference; Fishburn [3] gives some possible models.

One approach to mathematical modeling is to try to give a reasonable model of extremely non-transitive preference; the famous cyclic voter's paradoxes can be viewed as a model of preference which can allow not just non-transitivity, but also cycles.

Unlike these approaches, we generalize semi-orders to allow non-transitivity, but we require that the given set of preferences continue to be acyclic. In other words, we consider any preference relation represented by a directed acyclic graph (a *dag*). As in the case of semiorders, we assume that reported preferences are influenced by an underlying hidden utility function, which may be approximate, imperfectly known by a subject, or otherwise fail to capture all factors influencing a report of a preference.

One of our objectives is to obtain a measure of the departure of a given arbitrary acyclic set of pairwise preferences from a model where preferences are driven exclusively by an underlying hidden utility function, as well as derive an assignment of utility values that has the most explanatory power, in a sense that we define within a new model that we propose.

We propose a generalization of a semiorder, a *double-threshold semiorder*. We loosen the definition of a semiorder to a broader class of relations that are acyclic but not necessarily transitive, by allowing two thresholds t_1 and t_2 such that $t_1 \leq t_2$, and finding a valuation $\alpha(x)$ for each element x. For two elements x and y, (x, y) is not reported as a preference if $\alpha(y) - \alpha(x) < t_1$, (x, y) can freely be reported as a preference or not if $t_1 \leq \alpha(y) - \alpha(x) \leq t_2$, and (x, y) is reported as a preference if $\alpha(y) - \alpha(x) > t_2$. Let a *satisfying utility function* or a *satisfying assignment of α values* for (t_1, t_2) be a utility function α that meets these constraints. This accommodates within the model the well-known phenomenon in the literature on perception that there can be a range of differences between the minimum difference that is sometimes perceived and the minimum difference that is perceived reliably.

When the relation of the double-threshold semiorder is modeled by a dag, it is called a *double-threshold digraph*. If a dag can be represented with thresholds (t_1, t_2), then it can be represented with any pair (t'_1, t'_2) of thresholds such that $t'_2/t'_1 = t_2/t_1$, since a solution α

for (t_1, t_2) can be turned into a solution for (t'_1, t'_2) by rescaling all α values by the factor $t'_1/t_1 = t'_2/t_2$. Therefore, for any pair (t_1, t_2) of thresholds, the question of whether a particular dag can be represented with them depends on the ratio $r = t_2/t_1$; larger ratios allow representations of more dags.

Henceforth, given a digraph G, let $n(G)$ denote the number of vertices and $m(G)$ the number of edges. When G is understood, we may denote these as n and m. For a dag G, let $\lambda(G)$ denote the minimum ratio of t_2/t_1 such that G has a satisfying utility function for (t_1, t_2). When G models a weak order, $t_1 = 1$ and $t_2 = \epsilon$ for any $\epsilon > 0$ has a satisfying utility function. For this trivial special case, which is easily recognized in linear time, we define $\lambda(G)$ to be 0, the lower bound on the satisfiable ratios t_2/t_1, and call such a dag a *degenerate* dag. All other dags are *nondegenerate*.

When G or the preference relation it models is understood, we denote $\lambda(G)$ simply by λ. For a dag that models a nondegenerate semiorder, $\lambda = 1$; higher values of λ provide a measure of the degree to which a given set of preferences depart from a semiorder. An acyclic preference relation is a (t_1, t_2)-*semiorder* if it has a satisfying utility function for (t_1, t_2), that is, if $t_2/t_1 \geq \lambda$. When such a preference relation is modeled as a digraph, we say the digraph is a (t_1, t_2) *double-threshold digraph*. We show that for any nondegenerate dag G, $\lambda(G)$ can be expressed as a ratio j/i where i and j are integers such that $1 \leq i \leq j < i + j \leq n$ (Theorem 4), allowing t_1, t_2, and the utility function to have small integer values. Also, for any dag, $t_1 = 1$ and $t_2 = n - 1$ is always satisfiable, so $\lambda \leq n - 1$. An example of when the bound is tight is when G is a directed path.

Thus, the classes of dags with λ bounded by different values give a nested hierarchy of dags, starting with weak orders and semiorders. For each class in the hierarchy, we give a characterization of the class in terms of a set of forbidden subgraphs for the class.

When G has no satisfying utility function for t_1, t_2, we show how to return a forbidden subgraph as a certificate of this in $O(nm/r)$ time, where $r = t_2/t_1$, and an $O(nm/\lambda)$ time bound for finding λ (Theorem 18). The algorithm combines elements of the Bellman-Ford single-source shortest paths algorithm [1], Karp's minimum mean cycle algorithm [8], and dynamic programming techniques based on a topological sort of a dag. For $t_2/t_1 = \lambda$, a satisfying assignment, together with a forbidden subgraph for a smaller ratio, give a certificate that $\lambda = t_2/t_1$, and these take $O(nm/\lambda)$ time to produce.

If λ is less than 2, G must be transitive. The converse is not true: it is easy to show that the class of posets does not have bounded λ. Consider a chain $(v_1, v_2, \ldots, v_{n-1})$ in a poset and a vertex v_n that is incomparable to the others; $t_2 \geq t_1(n-2)/2$. Even though they are transitive, some posets are not good models of a preference relation that is based on an underlying utility function.

Although we show that bounding λ can make some NP-complete problems tractable, bounded-ratio double-threshold digraphs are in one sense enormously larger than semiorders. Semiorders correspond to digraphs that can be represented with ratio 1. These classes of digraphs both have implicit representations [12], implying that there are $2^{O(n \log n)}$ such digraphs on a set of n labeled vertices. By contrast, every height 1 digraph can be represented with ratio 1: for each vertex x, assign $\alpha(x) = 0$ if is it a source or $\alpha(x) = 1$ if it is a sink and make the thresholds $t_1 = t_2 = 1$. The number of such digraphs on n labeled vertices, hence the number with ratio λ for any λ greater than or equal to 1, is $2^{\Theta(n^2)}$.

The *underlying undirected graph* of a dag is the symmetric closure, that is, the undirected graph obtained by ignoring the orientations of the edges. In this paper, we say that a dag is *connected* if its underlying undirected graph is connected. Similarly, by a *clique*, *coloring*,

independent set, or *clique cover* of a dag, we mean a clique, coloring, independent set or clique cover of the underlying undirected graph. Hardness results about these problems on undirected graphs also apply to dags, since every undirected graph G is the underlying undirected graph of the dag obtained by assigning an acyclic orientation to G's edges.

Finding a maximum independent set or clique in a dag takes polynomial time if the dag is transitive (a poset), hence if it is a semiorder, but for arbitrary dags, there is no polynomial-time approximation algorithm for finding a independent set or clique whose size is within a factor of $n^{1-\epsilon}$ of the largest unless P = NP [7]. However, for a connected dag G, we give an $O(\lambda m^{\lfloor \lambda+1 \rfloor/2})$ algorithm for finding a maximum clique (Corollary 10), and an approximation algorithm that finds a clique whose size is within a desired factor of i of that of a maximum clique in $O(nm/\lambda + m^{\lfloor \lambda/i+1 \rfloor/2})$ time (Corollary 11).

We show that finding a maximum independent set is still NP-hard when $\lambda \geq 2$, but we give a polynomial-time approximation algorithm that produces an independent set whose size is within a factor of $\lfloor \lambda + 1 \rfloor$ of the optimum (Theorem 12). We give approximation bounds of $\lfloor \lambda + 1 \rfloor$ for minimum coloring and minimum clique cover (Theorems 13 and 14), which also have no polynomial algorithms for finding an $n^{1-\epsilon}$ approximation for arbitrary dags unless P = NP.

Thus, restricting attention to dags such that λ is bounded by a constant makes some otherwise NP-hard problems easy and gives rise to polynomial-time approximation algorithms that cannot exist in general unless P = NP. In each case, the time bound or the approximation bound is an increasing function of λ. This supports the view of λ as a measure of complexity of a dag. By contrast, for most similar attempts to measure complexity of a graph or digraph, the measurement is NP-hard to compute; examples include dimension of a poset, interval number, boxicity, and many others; see [12].

A concept similar to λ was given previously by Gimbel and Trenk in [5]. They developed a generalization of weak orders to partial orders that corresponds to the special case of a $(1, k)$ transitive dag. Not assuming transitivity requires us to use different algorithmic methods, but our bounds improve their bounds for their special case from $O(n^4k)$ and $O(n^6)$ to $O(mn/k)$. Most of their structural results are disjoint from ours because they are relevant to partial orders and their underlying undirected graphs, the *comparability graphs.*

2 Satisfying utility functions and forbidden subgraphs

We give the following formal definition:

▶ **Definition 1.** A dag is a (t_1, t_2) double-threshold digraph if there exists an assignment of a real value $\alpha(v)$ to each vertex v such that whenever (u, v) is an edge, $\alpha(v) - \alpha(u) \geq t_1$ and whenever (u, v) is not an edge, $\alpha(v) - \alpha(u) \leq t_2$.

Whether the constraints can be satisfied can be formulated as the problem of finding a feasible solution to a linear program:

- $\alpha(v) - \alpha(u) \geq t_1$ for each (u, v) such that (u, v) is an edge;
- $\alpha(v) - \alpha(u) \leq t_2$ for each (u, v) such that neither (u, v) nor (v, u) is an edge;
- $\alpha(v) \leq 0$ for all $v \in V(G)$.

The last constraint is added as a convenience; for any satisfying assignment, an arbitrary constant can be subtracted from all of the α values to obtain a new satisfying assignment, so the constraint cannot affect the existence of a feasible solution.

This is a special case of a linear program, a *system of difference constraints*, where each constraint is an upper bound on the difference of two variables. This reduces to the problem of finding the weight of a least-weight path ending at each vertex in a digraph derived from

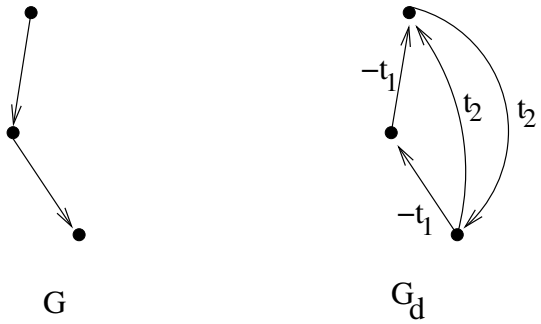


Figure 1 Reduction of finding a satisfying utility function to the single-source least-weight paths problem. Edges of weight t_1 in G_d are acyclic.

the constraints, as described in [1], where there is a satisfying assignment if and only if the digraph of the reduction has no negative-weight cycle. Applying the reduction to the problem of determining whether there is a satisfying utility function on G yields a digraph G_d, where $V(G_d) = V(G)$ (see Figure 1). G_d has an edge (y, x) of weight $-t_1$ for each edge (x, y) of G, and edges (u, v) and (v, u) of weight t_2 for each pair $\{u, v\}$ such that neither of (u, v) and (v, u) is an edge of G. A negative cycle in G_d proves that the system is not satisfiable; otherwise, for each $x \in V$, assigning $\alpha(x)$ to be the minimum weight of any path ending at x gives a satisfying assignment for (t_1, t_2).

The single-source least-weight paths problem where some weights are negative can be solved in $O(nm)$ time, but G_d has $\Theta(n^2)$ edges, so a direct application of this approach takes $\Theta(n^3)$ time to find a satisfying assignment or produce a negative-weight cycle in G_d. We derive tighter bounds below.

In terms of G, a negative cycle of G_d translates to a forbidden subgraph characterization of (t_1, t_2) double-threshold digraphs:

▶ **Definition 2.** Let (u, v) be a *hop* in G if neither (u, v) nor (v, u) is an edge of G. Let a *forcing cycle* be a simple cycle $(v_1, v_2, ..., v_k)$ such that such that for each consecutive pair (v_i, v_{i+1}) (indices mod k), the pair is either a directed edge of G or a hop. Let the *ratio* of the forcing cycle be the ratio of the number of edges to the number of hops.

▶ **Theorem 3.** *For a nondegenerate dag G, the minimum satisfiable ratio λ is equal to the maximum ratio of a forcing cycle in G.*

One consequence of the theorem is that when G is a nondegenerate dag, a satisfying assignment of α values for thresholds (t_1, t_2), together with a forcing cycle with ratio equal to t_2/t_1 gives a certificate that $\lambda(G) = t_2/t_1$, as illustrated in Figure 2.

▶ **Theorem 4.** *For every nondegenerate dag G, $\lambda(G)$ can be expressed as a ratio i/j of integers such that $1 \leq i \leq j < i + j \leq n$.*

Proof. This follows from the fact that $\lambda \geq 1$ and is the ratio of the number j of edges to the number i of hops on a forcing cycle. ◀

Aside from showing that optimum values of t_1 and t_2 can be expressed as small integers, the theorem gives an immediate $O(n^3 \log n)$ bound for finding λ. This is because it implies that the number of possible values j/i that λ can take on is $O(n^2)$, and that these can be generated and sorted in $O(n^2 \log n)$ time. A binary search on this list, spending $O(n^3)$ time at each probe j/i to determine whether G is an (i, j) double-threshold digraph, as described above, can then be used to find λ. Once λ is known, a satisfying assignment of utility values

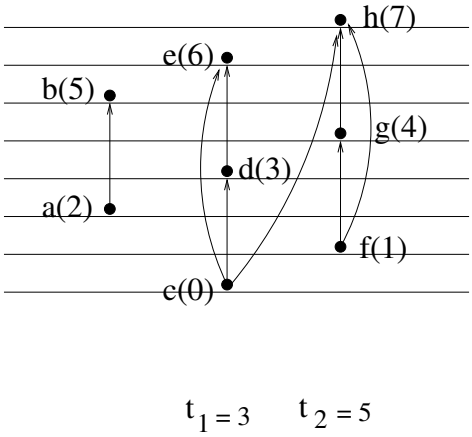


Figure 2 A dag such that $\lambda = 5/3$. The number next to each vertex is the value of the utility function, conforming to $t_1 = 3$ and $t_2 = 5$. The cycle (a, b, c, d, e, f, g, h) is a cycle of directed edges and hops in which the ratio of edges to hops is 5/3. Since $\lambda < 2$, the dag is transitive.

for $t_2/t_1 = \lambda$, together with a forcing cycle with forcing ratio equal to λ gives a certificate that the claimed value of λ is correct. We improve these bounds to $O(nm/\lambda)$ in section 5.

3 *k*-clique extendable orderings

In the book [12], Spinrad introduced the class of *k-clique extendable orderings* of the vertices of graphs, which we explain below. Finding whether a graph has a 2-clique extendable ordering takes polynomial time, but no polynomial time bounds are known for $k \geq 3$. However, we show in the next section that a topological sort of a nondegenerate dag G is a k-clique extendable ordering for $k = \lfloor \lambda(G) \rfloor + 1$, and develop several applications of this result to optimization problems. In this section, we give the details and analysis of the time bound of an algorithm suggested in [12] for finding a maximum clique, given a k-clique extendable ordering.

Two sets *overlap* if they intersect and neither is a subset of the other. Let $\sigma = (v_1, v_2, \ldots, v_n)$ be an ordering of the vertices of a graph, $G = (V, E)$. For $U \subseteq W \subseteq V$ let us say that W *ends with* U if the elements of U are the last elements of W in σ, that is, if no element of $W \setminus U$ occurs after an element of U. W *begins with* U if W ends with U in $(v_n, v_{n-1}, \ldots, v_1)$.

▶ **Definition 5.** An ordering $\sigma = (v_1, v_2, \ldots, v_n)$ of vertices of a graph $G = (V, E)$ is k-clique extendable ordering of G if, whenever X and Y are two overlapping cliques of size k, $|X \cap Y| = k - 1$, and $X \cup Y$ begins with $X \setminus Y = \{a\}$ and ends with $Y \setminus X = \{b\}$ in σ, then a and b are adjacent and $X \cup Y$ is a clique.

This is a generalization of transitivity, since a dag is transitive if and only if its topological sorts are two-clique extendable orderings, hence a graph is a comparability graph if and only if it has a two-clique extendable orderings. In [12], it is shown that three-clique extendable orderings arise naturally in connection with visibility graphs, and that it takes polynomial time to find a maximum clique in a graph, given a three-clique extendable ordering. A polynomial-time generalization for k-clique extendable orderings is implied; we give details and a time bound next.

▶ **Lemma 6.** *If $\sigma = (v_1, v_2, \ldots, v_n)$ is a k-clique extendable ordering of a graph G and X and Y are overlapping cliques of any size greater than or equal to k, such that $|X \cap Y| \geq k - 1$ and $X \cup Y$ begins with $X \setminus Y$ and ends with $Y \setminus X$ in σ, then $X \cup Y$ is a clique.*

Proof. It suffices to show that every element of $X \setminus Y$ is adjacent to every element of $Y \setminus X$. Let x be an arbitrary element of $X \setminus Y$, y be an arbitrary element of $Y \setminus X$, and Z be any $k-1$ elements of $X \cap Y$. Then $\{x\} \cup Z$ and $Z \cup \{y\}$ are two k-cliques and, by the definition of a k-clique extendable ordering, their union is a clique, and x and y are adjacent. ◀

▶ **Corollary 7.** *If $\sigma = (v_1, v_2, \ldots, v_n)$ is a k-clique extendable ordering of a graph G, X is a k-clique ending with $\{v\}$ and Z is a largest clique of G ending with the $(k-1)$-clique $X \setminus \{v\}$, then $Z \cup \{v\}$ is a largest clique of G ending with X.*

Proof. For any clique Y ending with X, $Y \setminus \{v\}$ is a clique ending with $X \setminus \{v\}$. $Z \cup \{v\} = Z \cup X$, which is a clique by Lemma 6. ◀

Corollary 7 is the basis of the recurrence for a dynamic programming algorithm for finding a maximum clique of G, given a k-clique extendable ordering. We enumerate all k-cliques and then label each k-clique K with the maximum size h_K of a clique that ends with K. If $(u_1, u_3, \ldots, u_k)$ is the left-to-right ordering of a k-clique in the ordering, then its label is one plus the maximum of the labels of cliques of the form $(x, u_1, u_2, \ldots, u_{k-1})$. The size of the maximum clique of G is the maximum of the labels. Details and the proof of the following resulting time bound appear in the ArXiv version [6].

▶ **Theorem 8.** *Given a k-clique extendable ordering of a graph G, a maximum clique can be found in $O(km^{k/2})$ time.*

It is easy to see that when the vertices of G have positive weights, the problem of finding a maximum weighted clique can be solved in the same time bound, using a trivial variant of Corollary 7.

4 Optimization problems on dags with bounded λ values

We now show that restricting attention to dags such that λ is bounded by a constant makes some otherwise NP-hard problems easy or gives rise to polynomial-time approximation algorithms that cannot exist for the class of all dags unless P = NP. The NP-hard problems we consider can be trivially solved in linear time on degenerate dags, so we focus on nondegenerate dags.

▶ **Theorem 9.** *Let G be a nondegenerate dag and $k = \lfloor \lambda(G) \rfloor + 1$. A topological sort of G is a k-clique extendable ordering.*

Proof. Let $(v_1, v_2, \ldots, v_n)$ be a topological sort, and let α be a satisfying utility function for (t_1, t_2) such that $t_2/t_1 = \lambda$. Let $(w_1, w_2, \ldots, w_k)$ and $(w_2, w_3, \ldots, w_k, w_{k+1})$ be the left-to-right orderings of two k-cliques K' and K. Then $(w_1, w_2, \ldots, w_{k+1})$ is a directed path in G, hence $\alpha(w_{k+1}) - \alpha(w_1) \geq kt_1 > t_2$, (w_1, w_{k+1}) is an edge and $K \cup K'$ is a clique. ◀

▶ **Corollary 10.** *It takes $O(\lambda m^{\lfloor \lambda+1 \rfloor/2})$ time to find a maximum clique in a connected nondegenerate dag G.*

Proof. To avoid an additive $O(nm/\lambda)$ term, run the dynamic programming algorithm on a topological sort under the assumption that it is a 2-clique extendable ordering in $O(m)$ time by Theorem 8, and return the result if it is a clique. Otherwise, do the same under the assumption that it is a 3-clique extendable ordering, in $O(m^{3/2})$ time. If a max clique has not yet been returned, then $\lambda \geq 3$ by Theorem 9, so compute λ in $O(nm/\lambda) = O(m^2)$ time, which is now subsumed by the bound we want to show. A topological sort is a $\lfloor \lambda \rfloor + 1$ extendable ordering by Theorem 9, so it takes $O(\lambda m^{\lfloor \lambda+1 \rfloor/2})$ time to find a maximum clique by Theorem 8. ◀

Even if λ is bounded by a moderately large constant, this bound could be prohibitive in practice, but it also gives an approximation algorithm that allows a tradeoff between time and approximation factor:

▶ **Corollary 11.** *Given a connected nondegenerate dag G and integer i such that $1 \leq i \leq \lambda$, a clique whose size is within a factor of i of the size of a maximum clique can be found in $O((\lambda/i)m^{(\lfloor \lambda/i \rfloor+1)/2})$ time.*

Proof. Let G' be the result of removing the edges $\{(u,v)|(u,v) \in E(G) \text{ and } \alpha(v)-\alpha(u) < i\}$. A satisfying function α for G and thresholds $(1, \lambda(G))$ is also a satisfying function for G' and thresholds $(i, \lambda(G))$, so $\lambda(G') \leq \lambda(G)/i$. Applying Theorems 8 and 9, we get a maximum clique of G' in $O((\lambda/i)m^{(\lfloor \lambda/i \rfloor+1)/2})$ time. A maximum clique of G induces a directed path $(v_0, v_1, ..., v_k)$ in G, and $\{v_0, v_i, v_{2i}, \ldots, v_{\lfloor k/i \rfloor}\}$ is a clique of G', so the size of a maximum clique in G' is within a factor of i of the size of a maximum clique in G. ◀

If $\lambda(G) < 2$, a maximum independent set in G can be obtained in polynomial time, since G is transitive [4]. However, even when $\lambda(G) = 2$, the problem of determining whether G has an independent set of size k is NP-complete. This is seen as follows. It is NP-complete to decide whether a 3-colorable graph has an independent set of a given size k, even when the 3-coloring is given [9]. Given such a graph G', k, and three-coloring, let C_1, C_2, and C_3 be the three color classes. Every edge e has endpoints in two of the classes; orient e from the endpoint in the class with the smaller subscript to the endpoint in the class with the larger subscript. Doing this for all edges results in a dag G such that $\lambda(G) = 2$, since, for each vertex x, if $x \in C_i$, assigning $\alpha(x) = i$ gives a satisfying assignment of utility values for $t_2 = 2$ and $t_1 = 1$. There is an independent set of size k in G if and only if there is one in G'.

▶ **Theorem 12.** *For G in the class of dags where $\lfloor \lambda(G) \rfloor + 1 \leq k$, there is a polynomial k-approximation algorithm for the problem of finding a maximum independent set in G.*

Proof. Find a satisfying assignment of utility values for (t_1, t_2) such that $t_2/t_1 = \lambda(G)$, then find an interval of the form $[x, x + t_1)$ such that the size of the set Y whose α values are in the interval is maximized. Y is an independent set, since no pair of them has α values that differ by t_1. Return these vertices as an independent set.

For the approximation bound, let X be a maximum independent set. The α values of X lie in an interval of the form $[y, y + t_2]$, which is a subset of the union $[y, y + kt_1)$, of k intervals of the form $[x, x + t_1)$, hence $|X| \leq k|Y|$. ◀

Proofs of the following make similar use of the availability of satisfying α values and are given in the ArXiv version [6].

▶ **Theorem 13.** *For G in the class of dags where $\lfloor \lambda(G) \rfloor + 1 \leq k$, there is a polynomial k-approximation algorithm for the problem of finding a minimum coloring of G.*

▶ **Theorem 14.** *For G in the class of dags where $\lfloor \lambda(G) \rfloor + 1 = k$, there is a polynomial k-approximation algorithm for the problem of finding a minimum clique cover of G.*

5 $O(nm/\lambda)$ bounds for finding satisfying utility functions, λ, and certificates

In this section, we first show how to find a satisfying assignment of utility values for given thresholds (t_1, t_2), in $O(nm/r)$ time, where $r = t_2/t_1$. We then show how to find λ in $O(nm/\lambda)$ time. By solving the second problem to find λ, then selecting (t_1, t_2) such that

$t_2/t_1 = \lambda$ and solving the first, we get the certificates for λ, that is, a satisfying assignment and a cycle such that the ratio of edges to hops is λ, which comes from a zero-weight cycle in G_d.

For both of these problems, we use the following. When G is an arbitrary digraph where each vertex x has a weight $w(x)$ and each edge (y, z) has a weight $w(y, z)$, it takes $O(m)$ time to find $w'(v) = \min(\{\infty\} \cup \{w(u) + w(u, v) | (u, v)$ is an edge of $G\}$ for each vertex v of G. Let us call this the *general relaxation procedure*. In the special case where G is a dag, it takes $O(m)$ time to find $w'(v) = \min_u(\{w(u) + w_{uv})\})$, where w_{uv} is the minimum weight of any path from u to v and $w_{vv} = 0$. This can be used to solve the single-source shortest paths problem on a connected dag in $O(m)$ time [1]. Let us call this the *dag variant* of the relaxation procedure.

In a digraph with edge weights, let the *length* of a walk be the number of occurrences of edges on the walk and its *weight* be the sum of weights of occurrences of edges. If an edge occurs k times on the walk, it contributes k to the length, and if its weight is w, it contributes kw to the weight and kw to the number of (occurrences of) edges of weight w on the walk.

5.1 Finding a satisfying utility function or a forbidden subgraph for (t_1, t_2)

The Bellman-Ford algorithm is a dynamic programming algorithm that works as follows on a connected digraph G where a vertex s has been added that has an edge of weight zero to all other vertices. Let $D(i, v)$ be the minimum weight of any walk from s to v that has at most $i+1$ edges. $D(i, v)$ is just the minimum weight of any walk of length at most i in G ending at v; henceforth we omit s from the discussion. $D(0, v) = 0$ for all $v \in V$. During the "i^{th} pass" the algorithm computes $D(i, v)$ as $min(\{(D(i-1, u) + w(u, v) | (u, v) \in E\})$. This is just an instance of the general relaxation procedure where $w(v) = D(i-1, v)$ and the loop (v, v) is considered to be an edge of weight 0 for each $v \in V$. If there is no negative cycle, there is always a path ending at v that is a minimum-weight walk ending at v, so $D(n-1, v)$ gives the minimum weight of any path ending at v. If there is a negative cycle, this is detected when $D(n, v) < D(n-1, v)$ for some v, indicating a walk of length n of smaller weight of any path, which must have a negative cycle on it. By annotating the dynamic programming entries with suitable pointers, it is possible to find such a cycle within the same bound. The n passes to compute $D(n, v)$ for all v each take $O(m)$ time, for a total of $O(nm)$ time.

To exploit the structure of G_d to improve the running time, we let $B(i, v)$ denote the minimum weight of any path that has at most i edges of weight t_2, rather than at most i edges in total. We use the elements $B(i, v)$, rather than the elements of $D(i, v)$, as the elements of the dynamic programming table. Let us call this *reindexing the dynamic programming table*. We obtain $B(0, v)$ by assigning $w(v) = 0$ and running the dag variant of the relaxation procedure on the edges of weight $-t_1$, since they are acyclic. For pass i such that $i > 0$, any improvements obtained by allowing an i^{th} edge of weight t_2 are computed with the general relaxation procedure, where loops are considered to be edges of weight 0, and, after this, any additional improvements obtained by appending additional edges of weight $-t_1$ are computed by the dag variant of the relaxation procedure.

Because every vertex has a walk of length and weight 0 ending at it, $B(i, v) \leq 0$ for $i \geq 0$. Therefore, for $i > 0$, if $B(i, v) < B(i-1, v)$, the ratio of edges of weight $-t_1$ to edges of weight t_2 is greater than $r = t_2/t_1$. Any such walk must have more than ir edges of weight $-t_1$, hence length greater than $i(r+1)$. Therefore, if there is no negative cycle in G_d, for $i = \lfloor (n-1)/(r+1) \rfloor + 1$, $B(i, v) = D(n-1, v)$, and a negative cycle occurs if $B(i+1, v) < B(i, v)$ for this i and some v. A negative cycle can be found by the standard

technique of annotating the results of the relaxation operations with pointers to earlier results. The advantage of reindexing the table is that the algorithm now takes $O(n/r)$ passes instead of n of them.

To get the $O(nm/r)$ bound, it remains to show how to perform each pass in $O(m)$ time. The bottleneck is evaluating $w'(v) = \min\{\{w(v)\} \cup \{w(u) + t_2 | (u, v)$ is an edge of weight $t_2\}$ for the general relaxation step. Since all of the edges have the same weight, we rewrite this as $w'(v) = min\{w(v), w(x) + t_2\}$ where x minimizes $w(u) = B(i-1, u)$ for all u such that $w(u, v) = t_2$. To evaluate this, we just have to find x. At the beginning of the pass, we radix sort the vertices in ascending order of $B(i-1, *)$, giving list L. To compute $B'(i, v)$, we mark the vertices that have an edge to v, then traverse L until we find x as the first unmarked vertex we encounter, then unmark the vertices that have edges to v. This takes time proportional to the in-degree of v, hence $O(m)$ time for all vertices in the pass.

5.2 Finding λ

To find $\lambda(G)$, we use the fact that that if $t_2/t_1 = \lambda$, the corresponding weighting of G_d will give it a zero-weight cycle in G_d, which gives a forcing cycle of ratio λ in G as a certificate.

For arbitrary (t_1, t_2), let the *mean weight* of a directed cycle or path of length at least one in G_d be the weight of the cycle divided by the number of edges. The minimum mean weight of a cycle is the *minimum cycle mean*. Subtracting a constant c from the weight of all edges in G_d subtracts c from the mean weight of every cycle and path of length at least one. For arbitrary t_1 and t_2, weighting G_d in accordance with $(t_1 + c, t_2 - c)$ in place of (t_1, t_2) has the same effect of subtracting c from the weights of all edges. Thus, for arbitrary (t_1, t_2), if c is the minimum cycle mean of the corresponding weighting of G_d, then $\lambda = (t_2 - c)/(t_1 + c)$. Finding λ reduces to finding the minimum cycle mean in the weighting of G_d obtained from an arbitrarily assigned (t_1, t_2).

In a digraph G with edge weights, let $F(i, v)$ be the minimum weight of any walk of length *exactly* i ending at v. In [8], Karp showed the following:

▶ **Theorem 15.** *The minimum cycle mean of a digraph with edge weights is* $\min_{v \in V} \max_{0 \le i < n}[(F(n, v) - F(i, v))/(n - i)]$.

Karp actually shows this when an arbitrary vertex s is selected and $F(i, v)$ is defined to be the minimum weight of all walks of length i from s to v, but if it is true for walks beginning at an arbitrary vertex s, then it is true when s is allowed to vary over all vertices of V. Omitting s from consideration in this way in his proof gives a direct proof of this variant of his theorem. He reduces the problem to the special case where G is strongly connected by working on each strongly-connected component separately, but the only purpose of this in his proof is to ensure that there is a path from s to all other vertices, and this is unnecessary when s is allowed to vary over all vertices.

$F(i, v)$ can be computed by a variant of Bellman-Ford, by using the recurrence $F(i, v) = \min(\{\infty\} \cup \{F(i-1, u) + w(u, v) | (u, v) \in E\})$ in place of $D(i, v) = \min(\{D(i-1, v)\} \cup \{D(i-1, u) + w(u, v) | (u, v) \in E\})$. The only difference from the algorithm of Section 5.1 is that loops of the form (v, v) are not considered to be edges. Computing $F(n, v)$ for all $v \in V$ takes n passes, each of which applies the general relaxation operation, for a total of $O(nm)$ time.

An obstacle to an $O(nm/\lambda)$ bound that we did not have in Section 5.1 is that in Theorem 15, computing $[(F(n, v) - F(i, v))/(n - i)]$ for $0 \le i < n$ requires $\Theta(n^2)$ computations, which is not $O(nm/\lambda)$.

We again reindex the dynamic programming table (Section 5.1), letting $H(i, v)$ denote the minimum-weight walk ending at v in G_d that has *exactly* i edges of weight t_2. We compute the values in passes, computing $H(i, v)$ for each $v \in V$ during pass i. As in Section 5.1, each pass takes takes $O(m)$ time; the only change is that in the general relaxation step, loops are not considered to be edges. We claim that $O(n/\lambda)$ passes suffice, but a new difficulty is knowing when to stop, since, unlike r of the Section 5.1, λ is not known in advance.

A walk with i edges of weight t_2 and weight $H(i, v)$ has i edges of weight t_2, so it must have $(it_2 - F(i, v))/t_1$ edges of weight $-t_1$. Its length, $l(i, v)$, can be computed as $i + it_2 - F(i, v)$ in $O(1)$ time.

Let a term $H(i, v)$ be *term of interest* if $l(i, v) = n$, that is, if it corresponds to a *walk of interest* of length n. We use the following reindexed variant of Karp's theorem, which says that it suffices to compute an inner maximum over a smaller set, and only for terms of interest. The proof is the one Karp gives, reindexed, and omitting reference to a start vertex s by allowing the start vertex to vary over all vertices. For completeness, we give the modified proof in the ArXiv version [6].

▶ **Theorem 16.** *In G_d, the minimum mean weight of a cycle is equal to* $\min_{\{(i,v)|l(i,v)=n\}} \max_{0\le j<i}(H(i, v) - H(j, v))/(n - l(j, v))$

The solution is given as Algorithm 1. During the i^{th} pass, the algorithm computes $H(i, v)$ for all $v \in V$. Before proceeding to the next pass, it updates a partial computation of the expression of Theorem 16, computing $\max_{0\le j<i}(H(i, v) - H(j, v))/(l(i, v) - l(j, v))$ for each the terms of interest $H(i, v)$ that has been computed during the pass, and keeping track of the minimum of these computations so far. Let a term of interest $H(i, v)$ be *critical* if the minimum cycle mean is equal to $\max_{0\le j<i}(H(i, v) - H(j, v))/(l(i, v) - l(j, v))$. The strategy of the algorithm is to return the minimum it has found so far once it detects that a critical term has been evaluated.

Let a *critical walk* be a walk of length n giving rise to a critical term.

▶ **Lemma 17.** *In G_d, the mean weight of a critical walk is less than or equal to the minimum cycle mean.*

The proof is given in the ArXiv version [6].

▶ **Theorem 18.** *Given a nondegenerate dag G, it takes $O(nm/\lambda)$ time to find $\lambda(G)$.*

Proof. The basis of this is Algorithm 1. For a term of interest, $H(i, v)$, the mean weight of the corresponding walk is $(it_2 - (n - i)t_1)/n$, which is an increasing function of i. Thus, once this exceeds the minimum value, *min*, found so far a critical term has been found and is already reflected in the value of *min*. Thus, Algorithm 1 returns the minimum cycle mean.

The minimum cycle mean is the ratio of edges of weight $-t_1$ to edges of weight t_2 on a cycle of minimum mean. This must also be true for a critical walk, by Lemma 17. This ratio for the walks of interest in pass i is $(n - i)/i$, so the algorithm halts before the first pass i' such that $(n - i')/i' > \lambda$, and $i' = O(n/\lambda)$. Thus, Algorithm 1 halts after $O(n/\lambda)$ passes.

Using the approach of Section 5.1, the operations in a pass take $O(m)$ time except for evaluating $\max_{0\le j<i}(H(i, v) - H(j, v))/(n - l(j, v))$ for terms $H(i, v)$ of interest. For any vertex v, $H(i, v)$ is a term of interest for at most one value of i. Therefore, the cost of evaluating $\max_{0\le j<i}(H(i, v) - H(j, v))/(n - l(j, v))$ for terms of interests is bounded by the total number of dynamic programming table entries $H(j, w)$ for $0 \le j \le i$ and $w \in V$ computed by the algorithm, which is the number n of them computed in each pass times $O(n/\lambda)$ passes. This is $O(n^2/\lambda)$. ◀

Algorithm 1: Find the minimum cycle mean of G_d.

Data: G_d, (t_1, t_2)
Result: The minimum cycle mean λ of G_d

```
1  min := ∞ ;
2  H(0, v) := 0 for all v ∈ V ;
3  for i := 1 to ∞ do
4  |  if min < (it_2 − (n − i)t_1)/n then
5  |  |  return min
6  |  Compute H(i, v) for all v ∈ V from H(i − 1, v) for all v ∈ V ;
7  |  for each term H(i, v) such that l(i, v) = n do
8  |  |  k := max_{0≤j<i}(H(i, v) − H(j, v))/(n − l(j, v)) ;
9  |  |  if k < min then
10 |  |  |  min = k
```

References

1 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT, 2009.

2 Peter C. Fishburn. *Interval Orders and Interval Graphs: Study of Partially Ordered Sets*. Wiley, 1985.

3 Peter C. Fishburn. Nontransitive preferences in decision theory. *Journal of Risk and Uncertainty*, 4:113–134, 1991.

4 Fanica Gavril. Maximum weight independent sets and cliques in intersection graphs of filaments. *Information Processing Letters*, 11:181–188, 2000.

5 John G. Gimbel and Ann N. Trenk. On the weakness of an ordered set. *SIAM J. Discrete Math*, 11:655–663, 1998.

6 Peter Hamburger, Ross M. McConnell, Attila Pór, and Jeremy P. Spinrad. Double threshold digraphs, 2018. `arXiv:arXiv:1702.06614`.

7 Johan Hastad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Math.*, 182:105–142, 1999.

8 Richard Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

9 Jan Kratochvil and Jaroslav Nešetřil. Independent set and clique problems in intersection-defined classes of graphs. *Comment.Math.Univ.Carolinae*, 31:85–93, 1990.

10 Duncan R. Luce. Semiorders and a theory of utility discrimination. *Econometrica*, 24:178–191, 1956.

11 Marc Pirlot and Ph Vincke. *Semiorders: Properties, Representations, Applications*, volume 36 of *Theory and Decision Library. Series B: Mathematical and Statistical Methods*. Kluwer, 1997.

12 Jeremy P. Spinrad. *Efficient Graph Representations*, volume 19 of *Fields Institute Monographs*. American Mathematical Society, 1991.

Tree Tribes and Lower Bounds for Switching Lemmas

Jenish C. Mehta
California Institute of Technology, Pasadena, CA 91125, USA
jenishc@gmail.com

Abstract

Let f be a Boolean function on n variables, $\boldsymbol{\rho}$ a random p-restriction that independently keeps each variable unset (or free) with probability p and otherwise uniformly sets it to 0 or 1, and $\mathrm{DT}_{\mathrm{depth}}(f)$ denote the depth of the smallest depth decision tree for f. Let $\mathrm{R}_d(f|\boldsymbol{\rho})$ be the *resilience* of f to $\boldsymbol{\rho}$ for depth d, defined as

$$\mathrm{R}_d(f|\boldsymbol{\rho}) = \Pr_{\rho \leftarrow \boldsymbol{\rho}}[\mathrm{DT}_{\mathrm{depth}}(f|\rho) \geq d].$$

If $d \gg pn$, all functions have resilience close to 0 since less than d variables would remain unset with high probability. For $d \ll pn$, most functions f on n variables have resilience close to 1, and some functions, like AND and OR, have resilience close to 0. Håstad's Switching Lemma states that for t-DNFs, the resilience $\mathrm{R}_d(f|\boldsymbol{\rho})$ is upper bounded by $(5pt)^d$, and from known upper bounds on the size of constant depth circuits computing the parity function, it follows that there exist t-DNFs whose resilience is close to the bound obtained by Håstad. However, the exact bounds for such maximally resilient DNFs or their structure is unclear, and moreover, the argument is non-constructive.

In this work, we give an explicit construction of functions called Tree Tribes parameterized by an integer t and denoted Ξ_t (on n variables), such that

$$\mathrm{R}_d(\Xi_t|\boldsymbol{\rho}) \leq (4p2^t)^d,$$

and more importantly, the resilience is also *lower* bounded by the same quantity up to constants,

$$\mathrm{R}_d(\Xi_t|\boldsymbol{\rho}) \geq (c_0 p 2^t)^d,$$

for $0 \leq p \leq c_p 2^{-t}$ and $0 \leq d \leq c_d \frac{\log n}{2^t t \log t}$ (where c_0, c_p, c_d are universal constants). As a result, for sufficiently large n and small d, this gives a hierarchy of functions with strictly increasing resilience, and covers the entire region between the two extremes where functions have resilience (close to) 0 or 1.

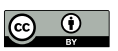
2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography, Theory of computation → Oracles and decision trees

Keywords and phrases Tree Tribes, Resilience, Switching lemmas, lower bounds, decision tree

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.70

Related Version A full version of the paper is available at [5], `https://arxiv.org/abs/1703.00043`.

Funding Supported by NSF CAREER Grant CCF-1553477 and NSF Grant 1618795



43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018).
Editors: Igor Potapov, Paul Spirakis, and James Worrell; Article No. 70; pp. 70:1–70:11

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

1 Introduction

One useful and powerful idea to separate a Boolean function g from some set $\mathcal{F}$ of Boolean functions over $\{0,1\}^n$ is to use *restrictions*. By showing that restricted to some subset of $\{0,1\}^n$, the functions in $\mathcal{F}$ become *simple*, but the function g does not become simple, it can be concluded that $g \notin \mathcal{F}$. The extent to which functions in $\mathcal{F}$ become simple is captured by Restriction Lemmas, which informally try to answer the following question: Given a family $\mathcal{F}$ of Boolean functions characterized by some parameter t, and a family $\mathcal{S}$ of distributions over subsets of $\{0,1\}^n$ characterized by some parameter p, how complex does a function $f \in \mathcal{F}$ remain after it is restricted to a subset S chosen according to some distribution $\boldsymbol{\sigma} \in \mathcal{S}$? Defining the measure of complexity of a function and the sets $\mathcal{F}$ and $\mathcal{S}$ gives a Restriction Lemma of a particular type.

To make more concrete statements, let f be a Boolean function on n variables, $\boldsymbol{\sigma}$ a distribution over subsets of $\{0,1\}^n$, and let $f|\sigma$ denote the partial function obtained by restricting f to a subset σ chosen according to $\boldsymbol{\sigma}$. In general, we will use the boldface symbol $\boldsymbol{\sigma}$ to denote both the distribution over subsets and a subset chosen according to $\boldsymbol{\sigma}$, and the interpretation will be clear from context. Let $M : \{0,1\}^{2^n} \to [0,1]$ be some function such that $M(f)$ measures the complexity of a (total) Boolean function f. For a partial function g, $M(g)$ is defined naturally to be the minimum value of $M(f)$ for any total function f that is an extension of g. Given these definitions, we define the *resilience* of a Boolean function f to $\boldsymbol{\sigma}$ for the measure M and threshold $\gamma \in [0,1]$ as follows:

$$\mathrm{R}^M_\gamma(f|\boldsymbol{\sigma}) = \Pr_{\sigma \leftarrow \boldsymbol{\sigma}}[M(f|\sigma) \geq \gamma] = \Pr[M(f|\boldsymbol{\sigma}) \geq \gamma].$$

It is possible to study the variation in the resilience of functions restricted to various different distributions $\boldsymbol{\sigma}$ under measures of complexity M, and in this work, we study it for a particular choice of M and $\boldsymbol{\sigma}$. We will let $M(f)$ be the depth of the smallest depth decision tree for f, and since $M(f) \in [n] = \{0,1,...,n\}$, denoting the threshold specifically as d instead of γ, we will also modify the range of the threshold d and let $d \in [n]$. The restriction $\boldsymbol{\rho}$ is defined as follows: independently for every variable x_i, leave x_i unset with probability p and otherwise uniformly set it to 0 or 1. Note that $\rho \leftarrow \boldsymbol{\rho}$ chooses a p-biased subset of $\{0,1\}^n$. Under the measure M and $\boldsymbol{\rho}$ defined thus, the resilience can be written as:

$$\mathrm{R}_d(f|\boldsymbol{\rho}) = \Pr_{\rho \leftarrow \boldsymbol{\rho}}[\mathrm{DT}_{\mathrm{depth}}(f|\rho) \geq d] = \Pr[M(f|\boldsymbol{\rho}) \geq \gamma]$$

where the measure M is understood to be $\mathrm{DT}_{\mathrm{depth}}$ when omitted from the superscript.

Note that since every variable is independently left unset with probability p, there are $\approx pn$ variables that are left unset with high probability, and so if $d \gg pn$, then for any function f, $\mathrm{R}_d(f|\boldsymbol{\rho}) \approx 0$. Thus, to understand the resilience of different functions, the interesting range is $d \ll pn$, and in general, it will be worthwhile for us to intuitively understand d to be tiny, or about $O(\log \log n)$, since even the statements we make for $d = 1$ will be interesting in themselves. For one extreme, let f be the AND (or OR) function on n bits, and note that $\mathrm{DT}_{\mathrm{depth}}(f) = n$. However, since setting any bit or f to 0 (or 1) reduces it to a constant function, $\mathrm{DT}_{\mathrm{depth}}(f|\boldsymbol{\rho}) > 0$ with probability $(\frac{1+p}{2})^n \approx e^{-n} \approx 0$ (since $p \ll \frac{1}{2}$), and thus $\mathrm{R}_d(f|\boldsymbol{\rho}) \approx 0$ for all $d \geq 1$. On the other extreme, if f is the parity function, it is still the case that $\mathrm{DT}_{\mathrm{depth}}(f) = n$, but under the action of $\boldsymbol{\rho}$, there will be $\geq \frac{1}{2}pn$ variables that are unset with high probability, and $f|\boldsymbol{\rho}$ will be the parity function on the subset of variables, and if $d < \frac{1}{2}pn$, then $\mathrm{R}_d(f|\boldsymbol{\rho}) \approx 1$. In fact, since most functions require a circuit of

large size to decide them, it turns out that for most functions f on n variables, $\mathrm{R}_d(f|\boldsymbol{\rho}) \approx 1$. As a consequence of this, the property of not being reduced to a constant function when restricted to $\boldsymbol{\rho}$ satisfies the largeness condition of natural proofs [8].

This motivates the question about the type of Boolean functions f that have resilience between the two extremes of 0 and 1, and our main interest in this paper would be to understand the structure of such functions in the intermediate region. Upper bounds on $\mathrm{R}_d(f|\boldsymbol{\rho})$ are traditionally called Restriction or Switching Lemmas, which originated with the works of [2] and [1], and were proved in their strongest form by Håstad [3], who showed that if f is a DNF where each term has width t, then $\mathrm{R}_d(f|\boldsymbol{\rho})$ is upper bounded by $(5pt)^d$. Further, this bound can be used to prove tight lower bounds on the size of constant depth circuits deciding the parity function, and since there exist matching upper bounds, it implies that there exist t-DNFs whose resilience is close to $(5pt)^d$ (else it would be possible to get a better lower bound for the size of constant depth circuits deciding parity leading to a contradiction). However, the exact bounds achieved in terms of p, t and d are not known, and even the structure of such maximally resilient t-DNFs is unclear. Moreover, the argument is non-constructive.

Our result

Our main result is an *explicit* construction of functions for which we can prove tight lower and upper bounds on the resilience. The functions that we construct are called Tree Tribes, since they are similar to the Tribes function but on a tree-like structure, and are denoted by Ξ_t where t is an integer parameter. Our main theorem is the following.

▶ **Theorem 1.** *For every integer $t \geq 1$, for every n, there is an explicit function Ξ_t on n variables, such that for all p, t, and d,*

$$R_d(\Xi_t|\boldsymbol{\rho}) \leq (4p2^t)^d,$$

and for $0 \leq p \leq c_p 2^{-t}$ and $0 \leq d \leq c_d \left(\frac{\log n}{2^t t \log t}\right)$,

$$R_d(\Xi_t|\boldsymbol{\rho}) \geq (c_0 p 2^t)^d,$$

where c_p, c_d and c_0 are universal constants.

We would like to remark that the lower bound holds for d at most $\sim 2^{-t} \log n$, which is meaningful for $t \in O(\log \log n)$. However, the non-trivial part of the lower bound is that d is upper bounded by a function not only of t (which in general would be small), but of *both* t and n. Henceforth, when we consider the resilience of different functions, n will be assumed to be very large, and t and d will assumed to be small (about $O(\log \log n)$) compared to n.

An immediate corollary of Theorem 1 is the following:

▶ **Corollary 2.** *Let d and n be fixed. Let $i_1, \ldots, i_r$ be a sequence of integers such that $c_{t0} < i_1 < \ldots < i_r < c_{t1}(\log \log n - \log d)$ and $0 \leq p \leq c_p 2^{-i_r}$. Then the sequence of functions $\{\Xi_{i_j}\}_{j=1}^r$ (each on n variables) has strictly increasing resilience, i.e.*

$$R_d(\Xi_{i_j}|\boldsymbol{\rho}) < R_d(\Xi_{i_{j+1}}|\boldsymbol{\rho}),$$

where c_{t0}, c_{t1}, c_p are universal constants.

And similarly, we can get a resilience heirarchy in the following sense:

▶ **Corollary 3.** *Let d, n and r be fixed. Let t be an integer variable such that $c_{t0} \leq t \leq c_{t1}(\log\log n - \log d)/r$ and p a variable that is some function of t, i.e. $p = p(t)$ and $0 \leq p \leq c_p 2^{-rt}$. Then the sequence of functions $\{\Xi_{it}\}_{i=1}^{r}$ (each on n variables) has strictly increasing resilience, i.e.*

$$R_d(\Xi_{it}|\boldsymbol{\rho}) \in o(R_d(\Xi_{(i+1)t}|\boldsymbol{\rho}),$$

where c_{t0}, c_{t1}, c_p are universal constants.

The proof of the resilience upper bound in Theorem 1 uses a method of *conditioning on variables*, and thus avoids the complex conditioning in [3, 4], and also the combinatorial reasoning in [7, 6], and as a consequence, is *simpler* than both methods. The proof of the resilience lower bound is recursive, and proceeds by analyzing the coefficients of polynomials that arise in the analysis of Tree Tribes. Note that in Theorem 1, since we want to lower bound the probability of the event $\mathrm{DT}_{\mathrm{depth}}(\Xi_t|\boldsymbol{\rho}) \geq d$, we require that the decision tree with the least depth (and thus *every* decision tree) for $\Xi_t|\boldsymbol{\rho}$ must have depth greater than d with sufficient probability. To achieve this, our proof proceeds as follows: If T is the decision tree for Ξ_t, we lower bound the probability of finding *"paths with a split"* in $T|\boldsymbol{\rho}$. A *"path with a split"* is a subtree of $T|\boldsymbol{\rho}$, which consists of a path of distinct variables $y_1, \ldots, y_d$ (where y_1 is closest to the root in $T|\boldsymbol{\rho}$), such that y_d is connected to two leaves with *different* values (more specifically, y_d has a path to a leaf labelled 0 and a leaf labelled 1). Any decision tree for such a subtree of $T|\boldsymbol{\rho}$ must have depth at least d (at least if the variables $y_1, \ldots, y_d$ do not appear elsewhere in the tree), since the OR function is embedded in $T|\boldsymbol{\rho}$ *and* all the variables of T would be distinct in our construction. However, we need a sufficient number of vertices in the tree before we get sufficient probability mass for the event of finding such a path with a split, and this is the reason we get an upper bound on the depth d for which Theorem 1 holds.

We state the Preliminaries in Section 2, the construction of Tree Tribes in Section 3, prove the resilience upper bound in Theorem 1 in Section 4, and the resilience lower bound in Section 5. Due to space constraints, the proofs are relegated to the full version [5].

2 Preliminaries

The extended preliminaries, including the standard definitions of decision trees and random p-restrictions are given in the full version [5], and we state only the non-standard definitions here.

▶ **Definition 4** (Resilience)**.** Let f be a Boolean function on n variables, and ρ a random p-restriction. For $d \in [n]$, we let $\mathrm{R}_d(f|\boldsymbol{\rho})$ be the resilience of f to $\boldsymbol{\rho}$ at depth d, defined as

$$\mathrm{R}_d(f|\boldsymbol{\rho}) = \Pr[\mathrm{DT}_{\mathrm{depth}}(f|\rho) \geq d].$$

If T is some decision tree for some function f, we alternately write $\mathrm{R}_d(T|\boldsymbol{\rho})$ to mean $\mathrm{R}_d(f|\boldsymbol{\rho})$.

▶ **Definition 5** (Clipped decision trees, [6])**.** A decision tree T is t-clipped, if any vertex of T is at a distance of at most t from some leaf.

▶ **Definition 6** (Operators on polynomials)**.** For a univariate polynomial Q in the variable p, denoted as $Q = \sum_{i \geq 0} c_i p^i$ with a finite number of non-zero coefficients, define the operator $[p^i]$ as $[p^i]Q = c_i$ and the operator $[\uparrow p^i]$ as

$$[\uparrow p^i]Q = \sum_{j \geq i} \left([p^j]Q\right) p^{j-i}.$$

▶ **Definition 7** (Absolute maximizer)**.** Given a closed set $\mathcal{D} \subseteq [0,1]$ and some polynomial Q in $\mathbb{R}[p]$, we define the absolute maximizer of $[\uparrow p^i]Q$ by functions G_i as $G_i(Q) = \max_{p\in\mathcal{D}} \left|[\uparrow p^i]Q\right|$.

3 Tree Tribes

We now formally define Tree Tribes and their variants. A specific variant, t-clipped xor Tree Tribe, will be the function that will help us achieve the bounds in Theorem 1.

▶ **Definition 8** (Tree Tribe)**.** A Boolean function f is called a Tree Tribe, denoted by Ξ, if there is a decision tree T deciding f, such that all the variables at all the vertices of T are distinct, and from every vertex of T, there is a path to a leaf labeled 0 and a path to a leaf labeled 1.

An example for a Tree Tribe is the OR function. Given the above definition, it is possible to derive a variety of different Tree Tribes by imposing additional structure, and we define the specific structure that we'll need.

▶ **Definition 9** (Complete clipped decision trees)**.** Denote a complete t-clipped decision tree T on r levels by $W_t(r)$, and define it recursively as follows: $W_t(0)$ is a leaf. $W_t(r)$ consists of vertices $\{v_1, ..., v_t\}$, a leaf denoted by v_{t+1}, and edges $e_{i,0}$ and $e_{i,1}$ for $i \in \{1, ..., t\}$. The vertices v_1 to v_t will be said to belong to layer or level 1. Each edge $e_{i,0}$ is labelled 0 and it will be called a 0-edge, and it connects v_i and v_{i+1}. Each edge $e_{i,1}$ is labelled 1 and it will be called a 1-edge, and it connects v_i to the root of a copy of $W_t(r-1)$ on distinct variables.

▶ **Definition 10** (Clipped xor Tree Tribe)**.** A Boolean function f is called a t-clipped xor Tree Tribe, denoted by $\Xi_t(r)$ for an integer t, if $T(f)$ is a Tree Tribe, and can be expressed as a complete t-clipped decision tree, in which the leaves are labeled by the parity of the edges on the path from the root to the leaf.

Note that we define the level of some variable x in $\Xi_t(r)$ by the recursive step at which it was added to $\Xi_t(r)$. The variables at level 1 are x_1 to x_t, each of which is connected to a copy of $\neg\Xi_t(r-1)$ on distinct variables. The first t variables in each of the t copies of$\neg\Xi_t(r-1)$, a total of t^2 variables, are at level 2, and each of them is connected to a copy of $\Xi_t(r-2)$, and so on. An examples for $\Xi_2(3)$ is shown in the figure below.

Since our aim was to understand the structure of functions that are resilient towards random restrictions, we show that the functions $\Xi_t(r)$ exhibit many nice properties (given in the full version [5]). For instance, the Fourier coefficient of a subset $S \subseteq \{0,1\}^n$ depends *only* on the parameters (level in the tree and distance from root) of the variable in S that represents the vertex farthest from the root in S. These and other properties can be found in [5] along with some remarks on why they turn out to be resilient towards random restrictions.

We now proceed to show that these functions achieve the bounds stated in Theorem 1.

4 Resilience upper bound

We start by by showing the resilience upper bound in Theorem 1. Our bound will hold in general for any function f that has a t-clipped decision tree. It was shown in [6] that for any Boolean function f that has a t clipped decision tree, $\mathrm{R}_d(f|\boldsymbol{\rho}) \leq O(pt2^t)^d$, and we improve the bound to $\mathrm{R}_d(f|\boldsymbol{\rho}) \leq (4p2^t)^d$, an improvement that is necessary for us since it matches the resilience lower bound. As stated earlier, our proof uses a method of *conditioning on variables* that is different and simpler from the complex conditioning in [3, 4], and the combinatorial

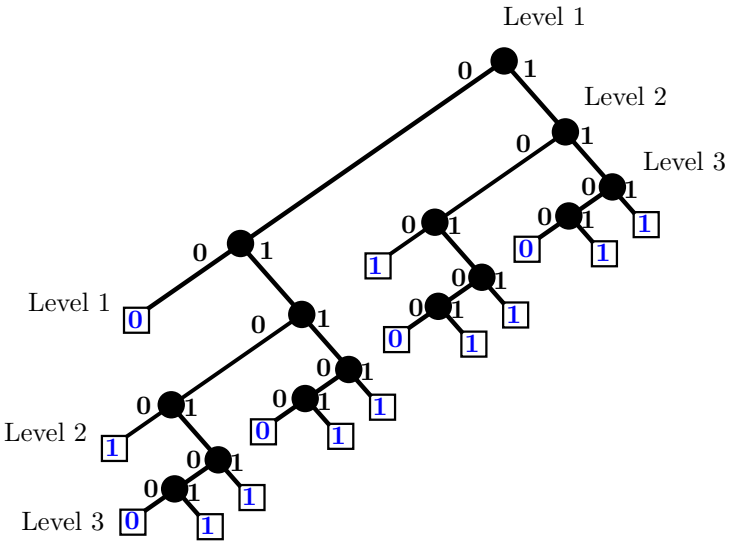


Figure 1 A 2-clipped xor tree tribe on 3 levels, or $\Xi_2(3)$. The variables at each of the vertices are distinct, and the leaves are labelled by the parity of the edges along the root to leaf path. Note that there are 2 vertices at level 1, 4 vertices at level 2, and 8 vertices at level 3.

reasoning in [7, 6], and is also amenable to extension to more general restrictions. The proofs of the lemmas in this section are given in the full version [5].

Let f be a Boolean function that has a t-clipped decision tree T.

▶ **Definition 11.** A decision tree T is (t_0, t)-clipped for $t_0 \leq t$, if the root has distance at most t_0 from some leaf, and every other vertex has distance at most t to some leaf.

▶ **Definition 12.** For positive integers t_0, t, n, d, define the probabilities $\gamma_{d,n}(t_0, t)$ as follows:

$$\gamma_{d,n}(t_0, t) = \max_{\substack{(t_0,t)\text{-clipped trees } T \text{ that} \\ \text{decide any function on } n \text{ variables}}} \mathrm{R}_d(T|\boldsymbol{\rho})$$

It is simple to see that γ is monotone in n.

▶ **Lemma 13.** *If $n' \leq n$, then $\gamma_{d,n'}(t_0, t) \leq \gamma_{d,n}(t_0, t)$.*

4.1 Recurrence for γ

We proceed by writing a recurrence for $\gamma_{d,n}(t_0, t)$.

Let T be the (t_0, t)-clipped decision tree for which $\gamma_{d,n}(t_0, t)$ has maximum value. Let x_1 be the root, and let T_0 and T_1 be subtrees out of the 0 and 1 edges of x_1. Under the action of a random p-restriction $\rho \leftarrow \boldsymbol{\rho}$, if x_1 is assigned 0 by ρ, we get a $(t_0 - 1, t)$ clipped decision tree T_0 on n_0 variables where $n_0 < n$. By the definition of decision trees, since x_1 appears as the root of T, it cannot appear again as a variable in T_0, and thus T_0 is indeed a $(t_0 - 1, t)$-clipped tree. If x_1 is assigned 1 by ρ, similarly, we get a (t, t) clipped decision tree T_1 on n_1 variables where $n_1 < n$.

Let the event $E_{T,d}$ be defined as follows:

$$E_{T,d} \equiv \mathrm{DT}_{\mathrm{depth}}(T|\boldsymbol{\rho}) \geq d.$$

▶ **Lemma 14.** *In case x_1 is assigned $*$ by a random p-restriction $\rho \leftarrow \boldsymbol{\rho}$,*

$$E_{T,d} = E_{T_0,d-1} \bigcup E_{T_1,d-1}.$$

Let $\boldsymbol{\rho} = \boldsymbol{\rho_{x_1}\rho'}$ where $\boldsymbol{\rho'}$ is a random restriction on variables different from x_1, and $q = (1-p)/2$. Thus, we can write the following,

$$\begin{aligned}
\Pr_{\rho\leftarrow\boldsymbol{\rho}}[E_{T,d}] &= \Pr_{\rho\leftarrow\boldsymbol{\rho}}[E_{T,d}|\rho(x_1)=0]\Pr_{\rho\leftarrow\boldsymbol{\rho}}[\rho(x_1)=0] + \Pr_{\rho\leftarrow\boldsymbol{\rho}}[E_{T,d}|\rho(x_1)=1]\Pr_{\rho\leftarrow\boldsymbol{\rho}}[\rho(x_1)=1] \\
&\quad + \Pr_{\rho\leftarrow\boldsymbol{\rho}}[E_{T,d}|\rho(x_1)=*]\Pr_{\rho\leftarrow\boldsymbol{\rho}}[\rho(x_1)=*] \\
&= \Pr_{\rho'\leftarrow\boldsymbol{\rho'}}[E_{T_0,d}]q + \Pr_{\rho'\leftarrow\boldsymbol{\rho'}}[E_{T_1,d}]q + \Pr_{\rho'\leftarrow\boldsymbol{\rho'}}[E_{T_0,d-1}\bigcup E_{T_1,d-1}]p \\
&\le \Pr_{\rho'\leftarrow\boldsymbol{\rho'}}[E_{T_0,d}]q + \Pr_{\rho'\leftarrow\boldsymbol{\rho'}}[E_{T_1,d}]q + \Pr_{\rho'\leftarrow\boldsymbol{\rho'}}[E_{T_0,d-1}]p + \Pr_{\rho'\leftarrow\boldsymbol{\rho'}}[E_{T_1,d-1}]p \qquad (1)
\end{aligned}$$

where the second line follows from Lemma 14 and the fact that the subtrees at x_1 do not contain x_1, and the last line follows from the union bound. Further, rewriting the above inequality in terms of $\gamma_{d,n}(t_0,t)$ and using Lemma 13, we can write,

$$\begin{aligned}
\gamma_{d,n}(t_0,t) &\le q\gamma_{d,n_0}(t_0-1,t) + q\gamma_{d,n_1}(t,t) + p\gamma_{d-1,n_0}(t_0-1,t) + p\gamma_{d-1,n_1}(t,t) \\
&\le q\gamma_{d,n}(t_0-1,t) + q\gamma_{d,n}(t,t) + p\gamma_{d-1,n}(t_0-1,t) + p\gamma_{d-1,n}(t,t).
\end{aligned}$$

The parameters n and t above are implicit, and we can rewrite the recurrence succinctly as

$$\gamma_d(t_0) \le q\gamma_d(t_0-1) + q\gamma_d(t) + p\gamma_{d-1}(t_0-1) + p\gamma_{d-1}(t) \qquad (2)$$

and for every integer d, we set

$$\gamma_d(0) = 0. \qquad (3)$$

We get the following recurrence for γ.

▶ **Lemma 15.** *After m iterations, the recurrence is,*

$$\gamma_d(t_0) \le \sum_{i=0}^{m}\binom{m}{i}q^{m-i}p^i\gamma_{d-i}(t_0-m) + \sum_{i=1}^{m}q^i\gamma_d(t) + \sum_{j=1}^{m}p^j\gamma_{d-j}(t)\left(\sum_{i=0}^{m-j}\binom{j+i}{i}q^i\right) \qquad (4)$$

4.2 Upper bound on γ

Setting $t_0 = t$ and $m = t$ in equation 4 and using 3, we get,

$$\frac{1-2q+q^{t+1}}{1-q}\gamma_d(t) \le \sum_{j=1}^{t}p^j\gamma_{d-j}(t)\left(\sum_{i=0}^{t-j}\binom{j+i}{i}q^i\right).$$

Using the induction hypothesis, for $\mu = \kappa 2^t$, we have that $\gamma_{d-c}(t) \le (\mu p)^{d-c}$ for all p, c, d, t where $p \le \frac{1}{\mu}$. Note that γ is a probability and since $p \le \frac{1}{\mu}$, it is also valid when $c > d$. [1] Then we have,

$$\frac{1-2q+q^{t+1}}{1-q}\gamma_d(t) \le (\mu p)^d\left(\sum_{j=1}^{t}\sum_{i=0}^{t-j}\left(\frac{1}{\mu}\right)^j\binom{j+i}{i}q^i\right). \qquad (5)$$

The next lemma is important, but the proof is relegated to the full version [5].

[1] In fact, whenever $d < t$, we can indeed get much fewer terms in the summation, and get a better bound, although asymptotically it does not make a difference.

▶ **Lemma 16.** *For* $\mu = 4.2^t$,

$$\left(\sum_{j=1}^{t}\sum_{i=0}^{t-j}\left(\frac{1}{\mu}\right)^j\binom{j+i}{i}q^i\right)\left(\frac{1-q}{1-2q+q^{t+1}}\right) \leq 1.$$

Using Lemma 16 in equation 5, we get that $\gamma_d(t) \leq (4p2^t)^d$, which concludes the proof of the resilience upper bound in Theorem 1.

5 Resilience lower bound

We now prove the lower bound on resilience in Theorem 1 which is the main contribution of this work. We prove the bound for t-clipped xor Tree Tribes or $\Xi_t(r)$, by induction on d. However, we cannot use $d = 0$ as the base case, and we discuss this when we do the inductive step. The base case will be $d = 1$, which we solve next. The proofs from this section are given in the full version [5].

5.1 The case for $d = 1$

This base case will turn out the most interesting. Here, we want to show the following.

$$\mathrm{R}_1(\Xi_t(r)|\boldsymbol{\rho}) \geq c_0 p 2^t.$$

If r is small, i.e., if the function has fewer than $\sim 2^t$ variables, then the number of variables assigned $*$ by $\rho \leftarrow \boldsymbol{\rho}$ will be low on average, and the event $\mathrm{DT}_{\mathrm{depth}}(\Xi_t(r)|\boldsymbol{\rho}) \geq 1$ would be extremely unlikely. Thus, we would like to show the following.

▶ **Lemma 17.** *For some universal constants* c_0 *and* c_p, *for* $r \in \Omega(t2^t)$ *and* $0 \leq p \leq c_p 2^{-t}$, *if* $\boldsymbol{\rho}$ *is a random* p*-restriction, then*

$$R_1(\Xi_t(r)|\boldsymbol{\rho}) \geq c_0 p 2^t.$$

Note that in Lemma 17, we require that the decision tree with the *smallest* depth that can compute $\Xi_t(r)|\boldsymbol{\rho}$ has depth greater than 1 with good probability, which means that we require that *any* decision tree representing $\Xi_t(r)|\boldsymbol{\rho}$ must query at least one variable. This will be made possible by a simple observation.

▶ **Lemma 18.** $DT_{depth}(\Xi_t(r)|\boldsymbol{\rho}) \geq 1$ *if and only if there is a path in* $\Xi_t(r)|\boldsymbol{\rho}$ *from the root to a leaf that evaluates to 0 and to a leaf that evaluates to 1.*

▶ **Definition 19.** Let $\Xi_t(r)$ be a t-clipped xor tribe on r levels and $\boldsymbol{\rho}$ a random p-restriction. Letting the parameters p and t be implicit, let $P_0(r)$, $P_1(r)$ and $P_*(r)$ be defined as follows.

$$\begin{aligned} P_0(r) &= \Pr_{\rho}[\Xi_t(r)|\boldsymbol{\rho} \equiv 0], \\ P_1(r) &= \Pr_{\rho}[\Xi(r)|\boldsymbol{\rho} \equiv 1], \\ P_*(r) &= \mathrm{R}_1(\Xi_t(r)|\boldsymbol{\rho}) \\ &= 1 - P_0(r) - P_1(r). \end{aligned} \tag{6}$$

Given Lemma 18, the proof for Lemma 17 proceeds as follows:

Step 1. We write the exact expressions for $P_0(r)$ and $P_1(r)$ as a polynomial in p by using counting arguments and recursion.

Step 2. In the next step, we reason about the constant coefficients of $P_0(r)$ and $P_1(r)$. We derive the expressions for $[1]P_0(r)$ and $[1]P_1(r)$, using which we show:

$$[1]P_0(r) + [1]P(r) = 1.$$

Step 3. In the third step, we first compute the recursive expressions for $[p]P_0(r)$ and $[p]P_1(r)$ and show the the following two claims.

▶ **Lemma 20.** *For every* $r \geq 1$, $-2.2^t \leq [p]P_0(r) \leq 0$ *and* $-2.2^t \leq [p]P_1(r) \leq 0$.

▶ **Lemma 21.** *For* $r \in \Omega(t2^t)$, $[p](P_0(r) + P_1(r)) \leq -c_1 2^t$ *where* $c_1 = \frac{1}{3}$.

The proofs of all the above lemmas are given in the full version [5]. Lemma 21 stated above is an important one, and it states there there is sufficient mass, $\sim 2^t$ in the coefficient of p in $P_0(r) + P_1(r)$. Combining lemma 20 and 21, we get that

$$-4.2^t \leq [p](P_0(r) + P_1(r)) \leq -c_1 2^t.$$

Step 4. In the equation at the end of step 3 above, we almost have sufficient information to conclude Lemma 17, however, we need to reason that higher powers of p in $P_0(r) + P_1(r)$ cannot substantially affect the coefficient of p. We do so by showing that the absolute value of $[\uparrow p^2](P_0(r) + P_1(r))$ is bounded by $O(2^{2t})$. Here we will use the fact that $p \leq c_p 2^{-t}$ where c_p is some universal constant. This lemma is also important, and its proof is given in the full version [5].

▶ **Lemma 22.** *For every* $r \geq 1$, *for* $0 \leq p \leq p_{\max} = \frac{1}{200.2^t}$,

$$G_2(P_0(r) + P_1(r)) \leq 30.2^{2t}.$$

Step 5. Finally we prove Lemma 17.

Proof of Lemma 17. The probability that the decision tree with r levels has depth more than or equal to 1, i.e. it does not become constantly 0 or 1 after being hit by a random restriction is given by

$$P_*(r) = 1 - P_0(r) - P_1(r).$$

For $r \in \Omega(t2^t)$, we can write

$$\begin{aligned} P_*(r) &= 1 - [1](P_0(r) + P_1(r)) - ([p]P_0(r) + [p]P_1(r))\,p - \left([\uparrow p^2](P_0(r) + P_1(r))\right)p^2 \\ &= -([p]P_0(r) + [p]P_1(r))\,p - \left([\uparrow p^2](P_0(r) + P_1(r))\right)p^2 \\ &\geq c_1 2^t p - p_{\max} 2.30.2^{2t} p \\ &= p2^t\left(\frac{1}{3} - \frac{60}{200}\right) \\ &\geq \frac{1}{30}p2^t \end{aligned}$$

where in the second line we used the fact that $[1]P_0(r) + [1]P(r) = 1$, and in the third line we used Lemmas 21 and 22. Note that we get $c_0 = \frac{1}{30}$. ◀

5.2 Inductive step

We remark that $d = 0$ cannot be taken as the base case for induction. In a decision tree T with x_1 as the root and having subtrees T_0 and T_1 out of the 0 and 1 edges respectively, if $\rho \leftarrow \boldsymbol{\rho}$ assigns $*$ to x_1, then $T_0|\rho$ or $T_1|\rho$ having depth greater than 0 does not imply that T has depth greater than 1. If both $T_0|\rho$ and $T_1|\rho$ are constant functions, we would additionally require that they evaluate to *different* values. Thus we cannot use $d = 0$ as the base case.

Doing the inductive step recursively is tricky. This is because we already need about $\sim t2^t$ levels in the tree for the base case to work. Thus, any induction must take care of the fact that once the number of levels are too few, the base case would not hold.

Let us write the recurrence first. Let $\gamma_d(r)$ be the probability that $\Xi_t(r)$ has depth greater than or equal to d under the action of a random restriction.

▶ **Lemma 23.** *Let $\mu = 1 - \gamma_{d-1}(r-1)$. Then,*

$$\gamma_d(r) \geq \sum_{k=1}^{t-1} q \sum_{i=1}^{k} \binom{k}{i} q^{k-i} p^i (1-\mu^{i+1}) + \sum_{i=0}^{t} \binom{t}{i} q^{t-i} p^i (1-\mu^i) + \sum_{k=0}^{t-1} q^{k+1} \gamma_d(r-1). \quad (7)$$

Note that we would use the above recurrence for m steps, since we need to pick up sufficient probability mass from each step. As such, we would want the inductive hypothesis to hold for *all* the m steps. The inductive steps will be different for the cases $t \geq 2$ and $t = 1$, shown in the following lemma.

▶ **Lemma 24.** *For $r \in \Omega(dt2^t)$, $\gamma_d(r) \geq (c_0 p 2^t)^d$.*

In the proof of Lemma 24, we use $O(2^t)$ steps to take the induction from d to $d-1$ levels, and the final $d = 1$ case can be carried out with $O(t2^t)$ levels. Thus, the maximum depth for which our conclusions hold is

$$dt2^t \leq O(r) \leq c_d \frac{\log n}{\log t}$$

or

$$d \leq c_d \left(\frac{\log n}{2^t t \log t} \right).$$

This concludes the proof of Theorem 1.

▶ Remark. If we unroll the induction and see how it worked, for depth d, we find some vertex that is unset by $\boldsymbol{\rho}$ and is connected to a tree that has depth $d-1$ under the action of ρ. For depth $d-1$, we again find an unset variable connected to a tree of depth $d-2$. This carries on until the last step, where we want to find a vertex that has depth greater than or equal to 1, i.e., has a path to both a 0-leaf and a 1-leaf. Thus, as described in the introduction, the whole proof essentially finds a *"path with a split"*.

References

1 Miklos Ajtai. Sigma 1-formulae on finite structures. *Annals of pure and applied logic*, 24(1):1–48, 1983.

2 Merrick Furst, James B Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Theory of Computing Systems*, 17(1):13–27, 1984.

3 Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 6–20. ACM, 1986.

4 Johan Håstad. On the correlation of parity and small-depth circuits. *SIAM Journal on Computing*, 43(5):1699–1708, 2014.

5 Jenish C. Mehta. Tree tribes and lower bounds for switching lemmas. *CoRR*, 2017. URL: `https://arxiv.org/abs/1703.00043`.

6 Toniann Pitassi, Benjamin Rossman, Rocco A Servedio, and Li-Yang Tan. Poly-logarithmic frege depth lower bounds via an expander switching lemma. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 644–657. ACM, 2016.

7 Alexander A Razborov. An equivalence between second order bounded domain bounded arithmetic and first order bounded arithmetic, 1993.

8 Alexander A Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, 1997.

Projection Theorems Using Effective Dimension

Neil Lutz
Department of Computer and Information Science,
University of Pennsylvania, 3330 Walnut Street, Philadelphia, PA 19104, USA
nlutz@cis.upenn.edu

Donald M. Stull[1]
Inria Nancy-Grand Est, 615 rue du jardin botanique, 54600 Villers-les-Nancy, France
donald.stull@inria.fr

Abstract

In this paper we use the theory of computing to study fractal dimensions of projections in Euclidean spaces. A fundamental result in fractal geometry is Marstrand's projection theorem, which shows that for every analytic set E, for almost every line L, the Hausdorff dimension of the orthogonal projection of E onto L is maximal.

We use Kolmogorov complexity to give two new results on the Hausdorff and packing dimensions of orthogonal projections onto lines. The first shows that the conclusion of Marstrand's theorem holds whenever the Hausdorff and packing dimensions agree on the set E, even if E is not analytic. Our second result gives a lower bound on the packing dimension of projections of arbitrary sets. Finally, we give a new proof of Marstrand's theorem using the theory of computing.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic

Keywords and phrases algorithmic randomness, geometric measure theory, Hausdorff dimension, Kolmogorov complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.71

1 Introduction

The field of fractal geometry studies the fine-grained structure of irregular sets. Of particular importance are fractal dimensions, especially the Hausdorff dimension, $\dim_H(E)$, and packing dimension, $\dim_P(E)$, of sets $E \subseteq \mathbb{R}^n$. Intuitively, these dimensions are alternative notions of size that allow us to quantitatively classify sets of measure zero. The books of Falconer [8] and Mattila [23] provide an excellent introduction to this field.

A fundamental problem in fractal geometry is determining how projection mappings affect dimension [9, 24]. Here we study orthogonal projections of sets onto lines. Let e be a point on the unit $(n-1)$-sphere S^{n-1}, and let L_e be the line through the origin and e. The *projection of E onto L_e* is the set

$$\mathrm{proj}_e E = \{e \cdot x : x \in E\},$$

where $e \cdot x$ is the usual dot product, $\sum_{i=1}^{n} e_i x_i$, for $e = (e_1, \ldots, e_n)$ and $x = (x_1, \ldots, x_n)$. We restrict our attention to lines through the origin because translating the line L_e will not affect the Hausdorff or packing dimension of the projection.

Notice that $\mathrm{proj}_e E \subseteq \mathbb{R}$, so the Hausdorff dimension of $\mathrm{proj}_e E$ is at most 1. It is also simple to show that $\dim_H(\mathrm{proj}_e E)$ cannot exceed $\dim_H(E)$ [8]. Given these bounds, it is natural to ask whether $\dim_H(\mathrm{proj}_e E) = \min\{\dim_H(E), 1\}$. Choosing E to be a line

[1] Research supported in part by National Science Foundation Grants 1247051 and 1545028.

orthogonal to L_e shows that this equality does not hold in general. However, a fundamental theorem due to Marstrand [21] states that, if $E \subseteq \mathbb{R}^2$ is analytic, then for *almost all* $e \in S^1$, the Hausdorff dimension of $\mathrm{proj}_e E$ is maximal. Subsequently, Mattila [22] showed that the conclusion of Marstrand's theorem also holds in higher-dimensional Euclidean spaces.

▶ **Theorem 1** ([21, 22]). *Let $E \subseteq \mathbb{R}^n$ be an analytic set with $\dim_H(E) = s$. Then for almost every $e \in S^{n-1}$,*

$$\dim_H(\mathrm{proj}_e E) = \min\{s, 1\}\,.$$

In recent decades, the study of projections has become increasingly central to fractal geometry [9]. The most prominent technique has been the potential theoretic approach of Kaufman [14]. While this is a very powerful tool in studying the dimension of a set, it requires that the set be analytic. We will show that techniques from theoretical computer science can circumvent this requirement in some cases.

Our approach to this problem is rooted in the effectivizations of Hausdorff dimension [16] by J. Lutz and of packing dimension by Athreya et al. [1]. The original purpose of these effective dimension concepts was to quantify the size of complexity classes, but they also yield geometrically meaningful definitions of dimension for *individual points* in $\mathbb{R}^n$ [18]. More recently, J. Lutz and N. Lutz established a bridge from effective dimensions back to classical fractal geometry by showing that the Hausdorff and packing dimensions of a set $E \subseteq \mathbb{R}^n$ are characterized by the corresponding effective dimensions of the individual points in E, taken relative to an appropriate oracle [17].

This result, a *point-to-set principle* (Theorem 7 below), allows researchers to use tools from algorithmic information theory to study problems in classical fractal geometry. Although this connection has only recently been established, there have been several results demonstrating the usefulness of the point-to-set principle: J. Lutz and N. Lutz [17] applied it to give a new proof of Davies' theorem [4] on the Hausdorff dimension of Kakeya sets in the plane; N. Lutz and Stull [20] applied it to the dimensions of points on lines in $\mathbb{R}^2$ to give improved bounds on generalized Furstenberg sets; and N. Lutz [19] used it to show that a fundamental bound on the Hausdorff dimension of intersecting fractals holds for arbitrary sets.

In this paper, we use algorithmic information theory, via the point-to-set principle, to study the Hausdorff and packing dimensions of orthogonal projections onto lines. Given the statement of Theorem 1, it is natural to ask whether the requirement that E is analytic can be removed. Without further conditions, it cannot; Davies [5] showed that, assuming the continuum hypothesis, there are non-analytic sets for which Theorem 1 fails. Indeed, Davies constructed a set $E^* \subseteq \mathbb{R}^2$ such that $\dim_H(E^*) = 1$ but $\dim_H(\mathrm{proj}_e E^*) = 0$ for every $e \in S^1$.

Our first main theorem shows that if the Hausdorff and packing dimensions of E agree, then we can remove the requirement that E is analytic.

▶ **Theorem 2.** *Let $E \subseteq \mathbb{R}^n$ be any set with $\dim_H(E) = \dim_P(E) = s$. Then for almost every $e \in S^{n-1}$,*

$$\dim_H(\mathrm{proj}_e E) = \min\{s, 1\}\,.$$

Our second main theorem applies to projections of *arbitrary* sets. Davies' construction precludes any non-trivial lower bound on the Hausdorff dimension of projections of arbitrary sets, but we are able to give a lower bound on the *packing* dimension.

▶ **Theorem 3.** *Let $E \subseteq \mathbb{R}^n$ be any set with $\dim_H(E) = s$. Then for almost every $e \in S^{n-1}$,*

$$\dim_P(\mathrm{proj}_e E) \geq \min\{s, 1\}\,.$$

Lower bounds on the packing dimension of projections have been extensively studied for restricted classes sets such as Borel and analytic [6, 7, 10, 12, 26]. To the best of our knowledge, our result is the first non-trivial lower bound of this type for arbitrary sets. It is known that the analogue of Marstrand's theorem for packing dimension does *not* hold [13].

Our other contribution is a new proof of Marstrand's projection theorem (Theorem 1). In addition to showing the power of theoretical computer science in geometric measure theory, this proof introduces a new technique for further research in this area. We show that the assumption that E is analytic allows us to use an earlier, restricted point-to-set principle due to J. Lutz [16] and Hitchcock [11]. While less general than that of J. Lutz and N. Lutz, it is sufficient for this application and involves a simpler oracle. Informally, this allows us to reverse the order of quantifiers in the statement of Theorem 1. This will be both beneficial for further research, as well as clarifying the role of the analytic assumption of E.

2 Preliminaries

We begin with a brief description of algorithmic information quantities and their relationships to Hausdorff and packing dimensions.

2.1 Kolmogorov Complexity in Discrete and Continuous Domains

The *conditional Kolmogorov complexity* of a binary string $\sigma \in \{0,1\}^*$ given a binary string $\tau \in \{0,1\}^*$ is the length of the shortest program π that will output σ given τ as input. Formally, the conditional Kolmogorov complexity of σ given τ is

$$K(\sigma \mid \tau) = \min_{\pi \in \{0,1\}^*} \{\ell(\pi) : U(\pi, \tau) = \sigma\} ,$$

where U is a fixed universal prefix-free Turing machine and $\ell(\pi)$ is the length of π. Any π that achieves this minimum is said to *testify* to, or be a *witness* to, the value $K(\sigma \mid \tau)$. The *Kolmogorov complexity* of a binary string σ is $K(\sigma) = K(\sigma \mid \lambda)$, where λ is the empty string. These definitions extend naturally to other finite data objects, e.g., vectors in $\mathbb{Q}^n$, via standard binary encodings; see [15] for details.

One of the most useful properties of Kolmogorov complexity is that it obeys the *symmetry of information*. That is, for every $\sigma, \tau \in \{0,1\}^*$,

$$K(\sigma, \tau) = K(\sigma) + K(\tau \mid \sigma, K(\sigma)) + O(1) .$$

Kolmogorov complexity can be naturally extended to points in Euclidean space, as we now describe. The *Kolmogorov complexity* of a point $x \in \mathbb{R}^m$ at *precision* $r \in \mathbb{N}$ is the length of the shortest program π that outputs a *precision-r* rational estimate for x. Formally, this is

$$K_r(x) = \min \{K(p) \,:\, p \in B_{2^{-r}}(x) \cap \mathbb{Q}^m\} ,$$

where $B_\varepsilon(x)$ denotes the open ball of radius ε centered on x. The *conditional Kolmogorov complexity of x at precision r given $y \in \mathbb{R}^n$ at precision $s \in \mathbb{R}^n$* is

$$K_{r,s}(x \mid y) = \max \left\{ \min\{K_r(p \mid q) \,:\, p \in B_{2^{-r}}(x) \cap \mathbb{Q}^m\} \,:\, q \in B_{2^{-s}}(y) \cap \mathbb{Q}^n \right\} .$$

When the precisions r and s are equal, we abbreviate $K_{r,r}(x \mid y)$ by $K_r(x \mid y)$. Given any positive real as a precision parameter, we round up to the next integer; for example, $K_r(x)$ denotes $K_{\lceil r \rceil}(x)$ whenever $r \in (0, \infty)$.

We will need the following technical lemmas which show that versions of the symmetry of information hold for Kolmogorov complexity in $\mathbb{R}^n$. The first Lemma 4 was proved in our previous work [20].

▶ **Lemma 4** ([20]). *For every $m, n \in \mathbb{N}$, $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, and $r, s \in \mathbb{N}$ with $r \geq s$,*

i. $|K_r(x \mid y) + K_r(y) - K_r(x, y)| \leq O(\log r) + O(\log\log \|y\|)$.

ii. $|K_{r,s}(x \mid x) + K_s(x) - K_r(x)| \leq O(\log r) + O(\log\log \|x\|)$.

▶ **Lemma 5.** *Let $m, n \in \mathbb{N}$, $x \in \mathbb{R}^m$, $z \in \mathbb{R}^n$, $\varepsilon > 0$ and $r \in \mathbb{N}$. If $K^x_r(z) \geq K_r(z) - \varepsilon r$, then the following hold for all $s \leq r$.*

i. $|K^x_s(z) - K_s(z)| \leq \varepsilon r - O(\log r)$.

ii. $|K_{s,r}(x \mid z) - K_s(x)| \leq \varepsilon r - O(\log r)$.

Proof. We first prove item (i). By Lemma 4(ii),

$$\begin{aligned}
\varepsilon r &\geq K_r(z) - K^x_r(z) \\
&\geq K_s(z) + K_{r,s}(z \mid z) - (K^x_s(z) + K^x_{r,s}(z \mid z)) - O(\log r) \\
&\geq K_s(z) - K^x_s(z) + K_{r,s}(z \mid z) - K^x_{r,s}(z \mid z) - O(\log r)\,.
\end{aligned}$$

Rearranging, this implies that

$$\begin{aligned}
K_s(z) - K^x_s(z) &\leq \varepsilon r + K^x_{r,s}(z \mid z) - K_{r,s}(z \mid z) + O(\log r) \\
&\leq \varepsilon r + O(\log r)\,,
\end{aligned}$$

and the proof of item (i) is complete.

To prove item (ii), by Lemma 4(i) we have

$$\begin{aligned}
\varepsilon r &\geq K_r(z) - K_r(z \mid x) \\
&\geq K_r(z) - (K_r(z, x) - K_r(x)) - O(\log r) \\
&\geq K_r(z) - (K_r(z) + K_r(x \mid z) - K_r(x)) - O(\log r) \\
&= K_r(x) - K_r(x \mid z) - O(\log r)\,.
\end{aligned}$$

Therefore, by Lemma 4(ii),

$$\begin{aligned}
K_s(x) - K_{s,r}(x \mid z) &= K_r(x) - K_{r,s}(x \mid x) - (K_r(x \mid z) - K_{r,s,r}(x \mid x, z)) \\
&\leq \varepsilon r + O(\log r) + K_{r,s,r}(x \mid x, z) - K_{r,s}(x \mid x) \\
&\leq \varepsilon r + O(\log r)\,,
\end{aligned}$$

and the proof is complete. ◀

2.2 Effective Hausdorff and Packing Dimensions

J. Lutz [16] initiated the study of effective dimensions by effectivizing Hausdorff dimension using betting strategies called *gales*, which generalize martingales. Subsequently, Athreya et al. defined effective packing dimension, also using gales [1]. Mayordomo showed that effective Hausdorff dimension can be characterized using Kolmogorov complexity [25], and Mayordomo and J. Lutz [18] showed that effective packing dimension can also be characterized in this way. In this paper, we use these characterizations as definitions. The *effective Hausdorff dimension* and *effective packing dimension* of a point $x \in \mathbb{R}^n$ are

$$\dim(x) = \liminf_{r\to\infty} \frac{K_r(x)}{r} \quad \text{and} \quad \mathrm{Dim}(x) = \limsup_{r\to\infty} \frac{K_r(x)}{r}\,.$$

Intuitively, these dimensions measure the density of algorithmic information in the point x. J. Lutz and N. Lutz [17] generalized these definitions by defining the *lower* and *upper conditional dimension* of $x \in \mathbb{R}^m$ given $y \in \mathbb{R}^n$ as

$$\dim(x \mid y) = \liminf_{r\to\infty} \frac{K_r(x \mid y)}{r} \quad \text{and} \quad \mathrm{Dim}(x \mid y) = \limsup_{r\to\infty} \frac{K_r(x \mid y)}{r}\,.$$

2.3 The Point-to-Set Principle

By letting the underlying fixed prefix-free Turing machine U be a universal *oracle* machine, we may *relativize* the definitions in this section to an arbitrary oracle set $A \subseteq \mathbb{N}$. The definitions of $K^A(\sigma|\tau)$, $K^A(\sigma)$, $K^A_r(x)$, $K^A_r(x \mid y)$, $\dim^A(x)$, $\mathrm{Dim}^A(x)$ $\dim^A(x \mid y)$, and $\mathrm{Dim}^A(x \mid y)$ are then all identical to their unrelativized versions, except that U is given oracle access to A. We will frequently consider the complexity of a point $x \in \mathbb{R}^n$ *relative to a point* $y \in \mathbb{R}^m$, i.e., relative to an oracle set A_y that encodes the binary expansion of y is a standard way. We then write $K^y_r(x)$ for $K^{A_y}_r(x)$.

The following *point-to-set principles* show that the classical notions of Hausdorff and packing dimension of a set can be characterized by the effective dimension of its individual points. The first point-to-set principle we use here, which applies to a restricted class of sets, was implicitly proven by J. Lutz [16] and Hitchcock [11].

A set $E \subseteq \mathbb{R}^n$ is a $\mathbf{\Sigma}^0_2$ set if it is a countable union of closed sets. The computable analogue of $\mathbf{\Sigma}^0_2$ is the class Σ^0_2, consisting of sets $E \subseteq \mathbb{R}^n$ such that there is a uniformly computable sequence $\{C_i\}_{i\in\mathbb{N}}$ satisfying

$$E = \bigcup_{i=0}^{\infty} C_i \,,$$

and each set C_i is *computably closed*, meaning that its complement is the union of a computably enumerable set of open balls with rational radii and centers. We will use the fact that every $\mathbf{\Sigma}^0_2$ set is Σ^0_2 relative to some oracle.

▶ **Theorem 6** ([16, 11]). *Let $E \subseteq \mathbb{R}^n$ and $A \subseteq \mathbb{N}$ be such that E is a Σ^0_2 set relative to A. Then*

$$\dim_H(E) = \sup_{x\in E} \dim^A(x) \,.$$

J. Lutz and N. Lutz [17] showed that the Hausdorff and packing dimension of *any* set $E \subseteq \mathbb{R}^n$ is characterized by the corresponding effective dimensions of individual points, relativized to an oracle that is optimal for the set E.

▶ **Theorem 7** (Point-to-set principle [17]). *Let $n \in \mathbb{N}$ and $E \subseteq \mathbb{R}^n$. Then*

$$\dim_H(E) = \min_{A\subseteq\mathbb{N}} \sup_{x\in E} \dim^A(x), \text{ and}$$
$$\dim_P(E) = \min_{A\subseteq\mathbb{N}} \sup_{x\in E} \mathrm{Dim}^A(x) \,.$$

It is worth noting that the point-to-set principle is taking the *minimum* over all oracles, not simply the *infimum*.

3 Bounding the Complexity of Projections

In this section, we will focus on bounding the Kolmogorov complexity of a projected point *at a given precision*. In Section 4, we will use these results in conjunction with the point-to-set principle to prove our main theorems.

We begin by giving intuition of the main idea behind this lower bound. We will show that under certain conditions, given an approximation of $e \cdot z$ and e, we can compute an approximation of the original point z. Informally, these conditions are the following.

1. The complexity $K_r(z)$ of the original point is small.
2. If $e \cdot w = e \cdot z$, then either $K_r(w)$ is large, or w is close to z.

Assuming that both conditions are satisfied, we can recover z from $e \cdot z$ by enumerating over all points u of low complexity such that $e \cdot u = e \cdot z$. By our assumption, any such point u must be a good approximation of z. We now formally state and prove this lemma.

▶ **Lemma 8.** *Suppose that* $z \in \mathbb{R}^n$ $e \in S^{n-1}$, $r \in \mathbb{N}$, $\delta \in \mathbb{R}_+$, *and* $\varepsilon, \eta \in \mathbb{Q}_+$ *satisfy* $r \geq \log(2\|z\| + 5) + 1$ *and the following conditions.*

i. $K_r(z) \leq (\eta + \varepsilon)\, r$.

ii. *For every* $w \in B_1(z)$ *such that* $e \cdot w = e \cdot z$,

$$K_r(w) \geq (\eta - \varepsilon)\, r + (r - t)\delta\,,$$

whenever $t = -\log \|z - w\| \in (0, r]$.

Then for every oracle set $A \subseteq \mathbb{N}$,

$$K_r^{A,e}(e \cdot z) \geq K_r^{A,e}(z) - \frac{n\varepsilon}{\delta} r - K(\varepsilon) - K(\eta) - O_z(\log r)\,.$$

Proof. Suppose z, e, r, δ, ε, η, and A satisfy the hypothesis.

Define an oracle Turing machine M that does the following given oracle (A, e) and input $\pi = \pi_1\pi_2\pi_3\pi_4\pi_5$ such that $U^A(\pi_1) = q \in \mathbb{Q}$, $U(\pi_2) = h \in \mathbb{Q}^n$, $U(\pi_3) = s \in \mathbb{N}$, $U(\pi_4) = \zeta \in \mathbb{Q}$, and $U(\pi_5) = \iota \in \mathbb{Q}$.

For every program $\sigma \in \{0,1\}^*$ with $\ell(\sigma) \leq (\iota + \zeta)s$, in parallel, M simulates $U(\sigma)$. If one of the simulations halts with some output $p = (p_1, \ldots, p_n) \in \mathbb{Q}^n \cap B_{2^{-1}}(h)$ such that $|e \cdot p - q| < 2^{-s}$, then $M^{A,e}$ halts with output p. Let c_M be a constant for the description of M.

Let π_1, π_2, π_3, π_4, and π_5 testify to $K_r^{A,e}(e \cdot z)$, $K_1(z)$, $K(r)$, $K(\varepsilon)$, and $K(\eta)$, respectively, and let $\pi = \pi_1\pi_2\pi_3\pi_4\pi_5$. Let σ be a program of length at most $(\eta+\varepsilon)r$ such that $\|p - z\| \leq 2^{-r}$, where $U(\sigma) = p$. Note that such a program must exist by condition (i) of our hypothesis. Then it is easily verified that

$$|e \cdot z - e \cdot p| \leq 2^{-r}\,.$$

Therefore $M^{A,e}$ is guaranteed to halt on π.

Let $M^{A,e}(\pi) = p = (p_1, \ldots, p_n) \in \mathbb{Q}^n$. Another routine calculation shows that there is some

$$w \in B_{2^{\gamma - r}}(p) \subseteq B_{2^{-1}}(p) \subseteq B_{2^0}(z)$$

such that $e \cdot w = e \cdot z$, where γ is a constant depending only on z and e. Then,

$$\begin{aligned} K_r^{A_e}(w) &\leq |\pi| + c_M \\ &\leq K_r^{A,e}(e \cdot z) + K_1(z) + K(r) + K(\varepsilon) + K(\eta) + c_M \\ &= K_r^{A,e}(e \cdot z) + K(\varepsilon) + K(\eta) + O(\log r)\,. \end{aligned}$$

Rearranging this yields

$$K_r^{A,e}(e \cdot z) \geq K_r^{A,e}(w) - K(\varepsilon) - K(\eta) - O(\log r)\,. \tag{1}$$

Let $t = -\log \|z - w\|$. If $t \geq r$, then the proof is complete. If $t < r$, then $B_{2^{-r}}(p) \subseteq B_{2^{1-t}}(z)$, which implies that $K_r^{A,e}(w) \geq K_{t-1}^{A,e}(z)$. Therefore,

$$K_r^{A,e}(w) \geq K_r^{A,e}(z) - n(r - t) - O(\log r)\,. \tag{2}$$

We now bound $r - t$. By our construction of M,

$$\begin{aligned}(\eta + \varepsilon)r &\geq K(p) \\ &\geq K_r(w) - O(\log r)\,.\end{aligned}$$

By condition (ii) of our hypothesis, then,

$$(\eta + \varepsilon)r \geq (\eta - \varepsilon)r + \delta(r - t)\,,$$

which implies that

$$r - t \leq \frac{2n\varepsilon}{\delta} r + O(\log r)\,.$$

Combining this with inequalities (1) and (2) concludes the proof. ◀

With the above lemma in mind, we wish to give a lower bound on the complexity of points w such that $e \cdot w = e \cdot z$. Our next lemma gives a bound based on the complexity, relative to z, of the direction $e \in S^{n-1}$. This is based on the observation that we can solve for $e = (e_1, \ldots, e_n)$ given w, z and $e_3, \ldots, e_n$. This follows from solving the system of two equations

$$\begin{aligned}e \cdot (z - w) &= 0 \\ e_1^2 + \ldots + e_n^2 &= 1\,.\end{aligned}$$

This suggests that

$$K_r^{z,e_3,\ldots,e_n}(e) \leq K_r^{z,e_3,\ldots,e_n}(w)\,.$$

However, for our purposes, we must be able to recover (an approximation of) e given *approximations* of w and z. Intuitively, the following lemma shows that we can algorithmically compute an approximation of e whose error is linearly correlated with the distance between w and z. We can then bound the complexity of w using a symmetry of information argument.

▶ **Lemma 9.** *Let $z \in \mathbb{R}^n$, $e \in S^{n-1}$, and $r \in \mathbb{N}$. Let $w \in \mathbb{R}^n$ such that $e \cdot z = e \cdot w$. Then there are numbers $i, j \in \{1, \ldots, n\}$ such that*

$$K_r(w) \geq K_t(z) + K_{r-t,r}^{e-\{e_i,e_j\}}(e \mid z) + O(\log r)\,,$$

where $t = -\log \|z - w\|$.

Proof. Let z, w, e, and r be as in the statement of the lemma. We first choose i so that $|z_i - w_i|$ is maximal. We then choose j so that

$$\begin{aligned}&\operatorname{sgn}((z_i - w_i)e_i) \neq \operatorname{sgn}((z_j - w_j)e_j), \text{ and} \\ &|z_j - w_j| > 0\,,\end{aligned}$$

where sgn denotes the sign. Note that such a j must exist since $(z - w) \cdot e = 0$. For the sake of removing notational clutter, we will assume, without loss of generality, that $i = 1$ and $j = 2$.

We first show that

$$K_{r-t,r}^{e_3,\ldots,e_n}(e_2 \mid z) \leq K_r(w \mid z) + O(1)\,. \tag{3}$$

As mentioned in the informal discussion preceding this lemma, note that

$$e_2 = \frac{-b + (-1)^h \sqrt{b^2 - 4ac}}{2a}\,, \tag{4}$$

where

- $h \in \{0,1\}$,
- $a = (z_1 - w_1)^2 + (w_2 - z_2)^2$,
- $b = 2(w_2 - z_2)\sum_{i=3}^{n}(w_i - z_i)e_i$, and
- $c = (\sum_{i=3}^{n}(w_i - z_i)e_i)^2 + (z_1 - w_1)^2\sum_{i=3}^{n} e_i^2 - 1$.

With this in mind, let M be the Turing machine such that, whenever $q = (q_1, \ldots, q_n) \in \mathbb{Q}^n$ and $U(\pi, q) = p = (p_1, \ldots, p_n) \in Q^2$ with $p_1 \neq q_1$,

$$M^{e_3,\ldots,e_n}(\pi, q, j) = \frac{-b' + (-1)^h\sqrt{b'^2 - 4a'c'}}{2a'},$$

where

- $h \in \{0,1\}$,
- $a' = (q_1 - p_1)^2 + (p_2 - q_2)^2$,
- $b' = 2(p_2 - q_2)\sum_{i=3}^{n}(p_i - q_i)d_i$, and
- $c' = (\sum_{i=3}^{n}(p_i - q_i)d_i)^2 + (q_1 - p_1)^2\sum_{i=3}^{n} d_i^2 - 1$, and
- $d = (d_3, \ldots, d_n) \in \mathbb{Q}^{n-2}$ is an nr-approximation of $(e_3, \ldots, e_n)$.

Let $q \in B_{2^{-r}}(z) \cap \mathbb{Q}^n$, and π_q testify to $\hat{K}_r(w \mid q)$. It tedious but straightforward (Lemma 10) to verify that

$$|M^{e_3,\ldots,e_n}(\pi_q, q, h) - e_2| \leq 2^{\alpha+t-r},$$

where α is a constant depending only on e. Hence, inequality (3) holds. Since

$$K_s^{e_3,\ldots,e_n}(e_2) = K_s^{e_3,\ldots,e_n}(e) + O(1)$$

holds for every s, we see that

$$K_{r-t,r}^{e_3,\ldots,e_n}(e \mid z) \leq K_r(w \mid z) + O(1). \tag{5}$$

To complete the proof, we note that

$$\begin{aligned} K_r(w \mid z) &\leq K_{r,t}(w \mid z) + O(\log r) \\ &= K_{r,t}(w \mid w) + O(\log r) \\ &= K_r(w) - K_t(w) + O(\log r) \\ &= K_r(w) - K_t(z) + O(\log r). \end{aligned}$$

The lemma follows from rearranging the above inequality, and combining inequality (5). ◀

The previous lemma uses the following technical lemma, whose proof is omitted due to space considerations.

▶ **Lemma 10.** *Let $z, w \in \mathbb{R}^n$, $e \in S^{n-1}$, and $r \in \mathbb{N}$ such that $e \cdot z = e \cdot w$. Let $q = (q_1, \ldots, q_n) \in \mathbb{Q}^n$ and $p = (p_1, \ldots, p_n) \in \mathbb{Q}^n$ be r-approximations of z and w, respectively. Then*

$$\left|\frac{-b + \sqrt{b^2 - 4ac}}{2a} - \frac{-b' + \sqrt{b'^2 - 4a'c'}}{2a'}\right| \leq 2^{-r+t+\alpha},$$

where a, b, c, a', b' and c' are as defined in Lemma 9, $t = -\log\|z - w\|$ and α is a constant depending only on e.

Finally, to satisfy the condition that $K_r(z)$ is small, we will use an oracle to "artificially" decrease the complexity of z at precision r. We will achieve this by applying the following lemma due to N. Lutz and Stull.

▶ **Lemma 11** ([20]). *Let $n, r \in \mathbb{N}$, $z \in \mathbb{R}^n$, and $\eta \in \mathbb{Q} \cap [0, \dim(z)]$. Then there is an oracle $D = D(n, r, z, \eta)$ and a constant $k \in \mathbb{N}$ depending only on n, z and η satisfying*

i. *For every $t \leq r$,*

$$K_t^D(z) = \min\{\eta r, K_t(z)\} + k \log r \,.$$

ii. *For every $m, t \in \mathbb{N}$ and $y \in \mathbb{R}^m$,*

$$K_{t,r}^D(y \mid z) = K_{t,r}(y \mid z) + k \log r \,,$$

and

$$K_t^{z,D}(y) = K_t^z(y) + k \log r \,.$$

4 Projection Theorems

The main results of the previous section gave us sufficient conditions for strong lower bounds on the complexity of $e \cdot z$ at a given precision, and methods to ensure that the conditions are satisfied. The following theorem encapsulates these results so that we may apply them in the proof of our main theorems.

▶ **Theorem 12.** *Let $z \in \mathbb{R}^n$, $e \in S^{n-1}$, $A \subseteq \mathbb{N}$, $\eta' \in \mathbb{Q} \cap (0,1) \cap (0, \dim(z))$, $\varepsilon' > 0$, and $r \in \mathbb{N}$. Assume the following are satisfied.*

1. *For every $s \leq r$, and $i, j \in \{1, \ldots, n\}$, $K_s^{e-\{e_i, e_j\}}(e) \geq s - \log(s)$.*
2. *$K_r^{A,e}(z) \geq K_r(z) - \varepsilon' r$.*

Then,

$$K_r^{A,e}(e \cdot z) \geq \eta' r - \varepsilon' r - \frac{2n\varepsilon'}{1-\eta'} r - K(2\varepsilon') - K(\eta') - O_z(\log r) \,.$$

Proof. Assume the hypothesis, and let $\eta = \eta'$, $\varepsilon = 2\varepsilon'$ and $\delta = 1 - \eta'$. Let $D_r = D(n, r, z, \eta')$ be the oracle as defined in Lemma 11.

First assume that the conditions of Lemma 8, relative to D_r, hold for z, e, r, η, ε and δ. Then we may apply Lemma 8, which, when combined item (2) and Lemma 11, yields

$$\begin{aligned} K_r^{A,D_r,e}(e \cdot z) &\geq K_r^{A,D_r,e}(z) - \frac{n\varepsilon}{\delta} r - K(\varepsilon) - K(\eta) - O_z(\log r) \\ &\geq K_r^{D_r}(z) - \varepsilon' r - \frac{n\varepsilon}{\delta} r - K(\varepsilon) - K(\eta) - O_z(\log r) \\ &= \eta' r - \varepsilon' r - \frac{2n\varepsilon'}{1-\eta'} r - K(\varepsilon') - K(\eta') - O_z(\log r) \,. \end{aligned}$$

Therefore, to complete the proof, it suffices to show that the conditions of Lemma 8, relative to D_r, hold.

Item (i) of Lemma 8 holds by our construction of D_r. To see that condition (ii) holds, let $w \in B_1(z)$ such that $e \cdot w = e \cdot z$. By Lemma 9, for some $i, j \in \{1, \ldots, n\}$,

$$K_r^{D_r}(w) \geq K_t^{D_r}(z) + K_{r-t,r}^{D_r, e-\{e_i, e_j\}}(e \mid z) + O(\log r) \,,$$

where $t = -\log \|z - w\|$. Therefore, by condition (2) of the hypothesis and Lemma 5,

$$K_r^{D_r}(w) \geq K_t^{D_r}(z) + K_{r-t,r}^{D_r, e-\{e_i, e_j\}}(e) - \varepsilon' r - O(\log r) \,.$$

By combining this with condition (1) of the present lemma and Lemma 11,

$$\begin{aligned} K_r^{D_r}(w) &\geq K_t^{D_r}(z) + K_{r-t,r}^{D_r, e-\{e_i,e_j\}}(e) - \varepsilon' r - O(\log r) \\ &\geq \eta' t + r - t - \varepsilon' r - O(\log r) \\ &= t(\eta' - 1) + r(1 - \varepsilon') - O(\log r) \\ &\geq (\eta - \varepsilon) r + \delta(r - t)\,. \end{aligned}$$

Hence, the conditions of Lemma 8 are satisfied and the proof is complete. ◀

4.1 Projection Theorems For Non-Analytic Sets

Our first main theorem shows that if the Hausdorff and packing dimensions of E are equal, the conclusion of Marstrand's theorem holds. Essentially this assumption guarantees, for every oracle A and direction e, the existence of a point $z \in E$ such that $\dim^{A,e}(z) \geq \dim_H(E) - \varepsilon$. This allows us to use Theorem 12 at all sufficiently large precisions r.

▶ **Theorem 2.** *Let $E \subseteq \mathbb{R}^n$ be any set with $\dim_H(E) = \dim_P(E) = s$. Then for almost every $e \in S^{n-1}$,*

$$\dim_H(\mathrm{proj}_e E) \geq \min\{s, 1\}\,.$$

Proof. Let $E \subseteq \mathbb{R}^n$ be any set with $\dim_H(E) = \dim_P(E) = s$. By the point-to-set principle, there is an oracle $B \subseteq \mathbb{N}$ testifying to $\dim_H(E)$ and $\dim_P(E)$. Let $e \in S^{n-1}$ be any point which is random relative to B. That is, let e be any point such that

$$K_r^{B,e-\{e_i,e_j\}}(e) \geq r - \log r\,,$$

for every $i, j \in \{1, \ldots, n\}$. Note that almost every point satisfies this requirement. Let $A \subseteq \mathbb{N}$ be the oracle testifying to $\dim_H(\mathrm{proj}_e E)$. Then, by the point-to-set principle, it suffices to show that for every $\varepsilon > 0$ there is a $z \in E$ such that

$$\dim^A(e \cdot z) \geq \min\{s, 1\} - \varepsilon\,.$$

To that end, let $\eta' \in \mathbb{Q} \cap (0,1) \cap (0,s)$ and $\varepsilon' > 0$. By the point-to-set principle, there is a $z_{\varepsilon'} \in E$ such that

$$\begin{aligned} s - \frac{\varepsilon'}{4} &\leq \dim^{A,B,e}(z_{\varepsilon'}) \\ &\leq \dim^B(z_{\varepsilon'}) \\ &\leq \mathrm{Dim}^B(z_{\varepsilon'}) \\ &\leq s\,. \end{aligned} \tag{6}$$

We now show that the conditions of Theorem 12 are satisfied, relative to B, for all sufficiently large $r \in \mathbb{N}$. We first note that, by inequality (6) and the definition of effective dimension,

$$\begin{aligned} sr - \frac{\varepsilon'}{4} r - \frac{\varepsilon'}{4} r &\leq K_r^{A,B,e}(z_{\varepsilon'}) \\ &\leq K_r^B(z_{\varepsilon'}) + O(1) \\ &\leq sr + \frac{\varepsilon'}{2} r\,, \end{aligned}$$

for all sufficiently large r. Hence, for all such r,

$$K_r^{A,B,e}(z_{\varepsilon'}) \geq K_r^B(z_{\varepsilon'}) - \varepsilon' r \,. \tag{7}$$

Thus the conditions of Theorem 12, relative to B, are satisfied.

We may therefore apply Theorem 12, resulting in

$$K_r^{A,B,e}(e \cdot z_{\varepsilon'}) \geq \eta' r - \varepsilon' r - \frac{2n\varepsilon'}{1-\eta'} r - K(\varepsilon') - K(\eta') - O_{z_{\varepsilon'}}(\log r) \,.$$

Hence,

$$\begin{aligned}
\dim^A(e \cdot z_{\varepsilon'}) &\geq \dim^{A,B,e}(e \cdot z_{\varepsilon'}) \\
&= \liminf_{r\to\infty} \frac{K_r^{A,B,e}(e \cdot z_{\varepsilon'})}{r} \\
&\geq \liminf_{r\to\infty} \frac{\eta' r - \varepsilon' r - \frac{2n\varepsilon'}{1-\eta'} r - K(\varepsilon') - K(\eta') - O_{z_{\varepsilon'}}(\log r)}{r} \\
&= \eta' - \varepsilon' - \frac{2n\varepsilon'}{1-\eta'} \,.
\end{aligned}$$

Since η' was chosen arbitrarily,

$$\dim^A(e \cdot z) \geq \min\{s, 1\} - \frac{\varepsilon'}{4} \,.$$

As ε' was chosen arbitrarily, by the point-to-set principle,

$$\begin{aligned}
\dim_H(\mathrm{proj}_e E) &\geq \sup_{z\in E} \dim^A(e \cdot z) \\
&\geq \sup_{\varepsilon>0} \dim^A(e \cdot z_{\varepsilon'}) \\
&= \min\{s, 1\} \,,
\end{aligned}$$

and the proof is complete. ◀

Our second main theorem gives a lower bound for the *packing* dimension of a projection for *general* sets. The proof of this theorem again relies on the ability to choose, for every (A, e), a point z whose complexity is unaffected relative to (A, e). This cannot be assumed to hold for every precision r. However, by the point-to-set principle, we can show that this can be done for infinitely many precision parameters r.

▶ **Theorem 3.** *Let $E \subseteq \mathbb{R}^n$ be any set with $\dim_H(E) = s$. Then for almost every $e \in S^{n-1}$,*

$$\dim_P(\mathrm{proj}_e E) \geq \min\{s, 1\} \,.$$

Proof. Let $E \subseteq \mathbb{R}^n$ be any set with $\dim_H(E) = s$. By the point-to-set principle, there is an oracle $B \subseteq \mathbb{N}$ testifying to $\dim_H(E)$ and $\dim_P(E)$. Let $e \in S^{n-1}$ be any point which is random relative to B. Note that almost every point satisfies this requirement. Let $A \subseteq \mathbb{N}$ be the oracle testifying to $\dim_P(\mathrm{proj}_e E)$. Then, by the point-to-set principle, it suffices to show that for every $\varepsilon > 0$ there is a $z \in E$ such that

$$\mathrm{Dim}^A(e \cdot z) \geq \min\{s, 1\} - \varepsilon \,.$$

To that end, let $\eta' \in \mathbb{Q} \cap (0, 1) \cap (0, s)$ and $\varepsilon' > 0$. By the point-to-set principle, there is a $z_{\varepsilon'} \in E$ such that

$$s - \frac{\varepsilon'}{4} \leq \dim^{A,B,e}(z_{\varepsilon'}) \leq \dim^B(z_{\varepsilon'}) \leq s \,. \tag{8}$$

We now show that the conditions of Theorem 12 are satisfied, relative to B, for infinitely many $r \in \mathbb{N}$. We first note that, by equation (8),

$$\begin{aligned} sr - \frac{\varepsilon'}{4}r - \frac{\varepsilon'}{4}r &\leq K_r^{A,B,e}(z_{\varepsilon'}) \\ &\leq K_r^B(z_{\varepsilon'}) + O(1) \\ &\leq sr + \frac{\varepsilon'}{2}r\,, \end{aligned}$$

for infinitely many r. Hence, for all such r,

$$K^{A,B,e}(z_{\varepsilon'}) \geq K^B(z_{\varepsilon'}) - \varepsilon r\,. \tag{9}$$

Thus the conditions of Theorem 12, relative to B, are satisfied for infinitely many $r \in \mathbb{N}$.

We may therefore apply Theorem 12, resulting in

$$K_r^{A,B,e}(e \cdot z_{\varepsilon'}) \geq \eta' r - \varepsilon' r - \frac{2n\varepsilon'}{1-\eta'}r - K(\varepsilon') - K(\eta') - O_{z_{\varepsilon'}}(\log r)\,,$$

for infinitely many $r \in \mathbb{N}$. Hence,

$$\begin{aligned} \mathrm{Dim}^A(e \cdot z_{\varepsilon'}) &\geq \mathrm{Dim}^{A,B,e}(e \cdot z_{\varepsilon'}) \\ &= \limsup_{r\to\infty} \frac{K_r^{A,B,e}(e \cdot z_{\varepsilon'})}{r} \\ &\geq \limsup_{r\to\infty} \frac{\eta' r - \varepsilon' r - \frac{2n\varepsilon'}{1-\eta'}r - K(\varepsilon') - K(\eta') - O_{z_{\varepsilon'}}(\log r)}{r} \\ &= \eta' - \varepsilon' - \frac{2n\varepsilon'}{1-\eta'}\,. \end{aligned}$$

Since η' was chosen arbitrarily

$$\mathrm{Dim}^A(e \cdot z) \geq \min\{s, 1\} - \frac{\varepsilon'}{4}\,.$$

As ε' was chosen arbitrarily, by the point-to-set principle

$$\begin{aligned} \dim_P(\mathrm{proj}_e E) &\geq \sup_{z \in E} \mathrm{Dim}^A(e \cdot z) \\ &\geq \sup_{\varepsilon > 0} \mathrm{Dim}^A(e \cdot z_{\varepsilon'}) \\ &= \min\{s, 1\}\,, \end{aligned}$$

and the proof is complete. ◀

4.2 Marstrand's Projection Theorem

We now give a new, algorithmic information theoretic proof of Marstrand's projection theorem. Recall that

▶ **Theorem 1.** *Let $E \subseteq \mathbb{R}^n$ be analytic with $\dim_H(E) = s$. Then for almost every $e \in S^{n-1}$,*

$$\dim_H(\mathrm{proj}_e E) = \min\{s, 1\}\,.$$

Note the order of the quantifiers. To use the point-to-set principle, we must first choose a direction $e \in S^{n-1}$. We then must show that for every oracle A and $\varepsilon > 0$, there is some $z \in E$ such that

$$\dim^A(e \cdot z) \geq \dim_H(E) - \varepsilon \,.$$

In order to apply Theorem 12, we must guarantee that (A, e) does not significantly change the complexity of z. To ensure this, we will use the point-to-set principle of J. Lutz and Hitchcock (Theorem 6). While this result is less general than the principle of J. Lutz and N. Lutz, the oracle characterizing the dimension of a Σ^0_2 set is easier to work with.

To take advantage of this, we use the following lemma.

▶ **Lemma 13.** *Let $E \subseteq \mathbb{R}^n$ be analytic with $\dim_H(E) = s$. Then there is a $\mathbf{\Sigma}^0_2$ set $F \subseteq E$ such that $\dim_H(F) = s$.*

Proof. It is well known that if $E \subseteq \mathbb{R}^n$ is analytic, then for every $\varepsilon \in (0, s]$, there is a compact subset $E_\varepsilon \subseteq E$ such that $\dim_H(E_\varepsilon) = s - \varepsilon$ (see e.g. Bishop and Peres [2]). Thus, the set

$$F = \bigcup_{i=\lceil 1/s \rceil}^{\infty} E_{1/i}$$

is a $\mathbf{\Sigma}^0_2$ set with $\dim_H(F) = s$. ◀

We will also use the following observation, which is a consequence of the well-known fact from descriptive set theory that Σ classes are closed under computable projections.

▶ **Observation 14.** *Let $E \subseteq \mathbb{R}^n$ and $A \subseteq \mathbb{N}$ be such that E is a Σ^0_2 set relative to A. Then for every $e \in S^{n-1}$, $\mathrm{proj}_e\, E$ is a Σ^0_2 set relative to (A, e).*

Finally, we must ensure that e does not significantly change the complexity of z. For this, we will use the following definition and theorem due to Calude and Zimand [3]. We rephrase their work in terms of points in Euclidean space. Let $n \in \mathbb{N}$, $z \in \mathbb{R}^n$ and $e \in S^{n-1}$. We say that z and e are *independent* if, for every $r \in \mathbb{N}$, $K^e_r(z) \geq K_r(z) - O(\log r)$ and $K^z_r(e) \geq K_r(e) - O(\log r)$.

▶ **Theorem 15** ([3])**.** *For every $z \in \mathbb{R}^n$, for almost every $e \in S^{n-1}$, z and e are independent.*

With these ingredients we can give a new proof Marstrand's projection theorem using algorithmic information theory.

Proof of Theorem 1. Let $E \subseteq \mathbb{R}^n$ be analytic with $\dim_H(E) = s$. By Lemma 13, there is a $\mathbf{\Sigma}^0_2$ set $F \subseteq E$ such that $\dim_H(F) = s$. Let $A \subseteq \mathbb{N}$ be an oracle such that F is Σ^0_2 relative to A. Using Theorem 6, for every $k \in \mathbb{N}$ we may choose a point $z_k \in F$ such that

$$\dim^A(z_k) \geq s - 1/k \,.$$

Let $e \in S^{n-1}$ be a point such that, for every $k \in \mathbb{N}$, the following hold.

- For every r and $t < r$, $K^{A, z_k, e_3 \ldots, e_n}_t(e) \geq t - O(1)$.
- For every r, $K^{A,e}_r(z_k) \geq K^A_r(z_k) - O(\log r)$.

A basic fact of algorithmic randomness states that almost every e satisfies the first item. By Theorem 15, almost every e satisfies the second item. So almost every e satisfies these requirements.

Fix $k \in \mathbb{N}$. Let $\eta' \in \mathbb{Q} \cap (0,1) \cap (0, \dim^A(z_k))$ and $\varepsilon' > 0$. It is clear, by our choices of e and z_k, that the conditions of Theorem 12 are satisfied for all sufficiently large r. We may therefore apply Theorem 12, resulting in

$$K_r^{A,e}(e \cdot z_k) \geq \eta' r - \varepsilon' r - \frac{2n\varepsilon'}{1-\eta'} r - K(2\varepsilon') - K(\eta') - O_z(\log r)\,.$$

Hence,

$$\begin{aligned}
\dim^{A,e}(e \cdot z_k) &= \liminf_{r\to\infty} \frac{K_r^{A,e}(e \cdot z_k)}{r} \\
&\geq \liminf_{r\to\infty} \frac{\eta' r - \varepsilon' r - \frac{2n\varepsilon'}{1-\eta'} r - K(2\varepsilon') - K(\eta') - O_z(\log r)}{r} \\
&= \eta' - \varepsilon' - \frac{2n\varepsilon'}{1-\eta'}\,.
\end{aligned}$$

Since both η' and ε' were chosen independently and arbitrarily, we see that

$$\begin{aligned}
\dim^{A,e}(e \cdot z_k) &\geq \dim^{A,e}(z_k) \\
&\geq \min\{s, 1\} - 1/k\,.
\end{aligned}$$

As k was chosen arbitrarily, Observation 14 and Theorem 6 give

$$\begin{aligned}
\dim_H(\mathrm{proj}_e\, E) &\geq \dim_H(\mathrm{proj}_e\, F) \\
&= \sup_{z\in F} \dim^{A,e}(e \cdot z) \\
&\geq \sup_{k\in\mathbb{N}} \dim^{A,e}(e \cdot z_k) \\
&= \min\{s, 1\}\,,
\end{aligned}$$

and the proof is complete. ◀

References

1. Krishna B. Athreya, John M. Hitchcock, Jack H. Lutz, and Elvira Mayordomo. Effective strong dimension in algorithmic information and computational complexity. *SIAM J. Comput.*, 37(3):671–705, 2007.
2. Christopher J. Bishop and Yuval Peres. *Fractals in probability and analysis*, volume 162 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 2017.
3. Cristian S. Calude and Marius Zimand. Algorithmically independent sequences. *Inf. Comput.*, 208(3):292–308, 2010.
4. Roy O. Davies. Some remarks on the Kakeya problem. *Proc. Cambridge Phil. Soc.*, 69:417–421, 1971.
5. Roy O. Davies. Two counterexamples concerning Hausdorff dimensions of projections. *Colloq. Math.*, 42:53–58, 1979.
6. K. J. Falconer and J. D. Howroyd. Projection theorems for box and packing dimensions. *Math. Proc. Cambridge Philos. Soc.*, 119(2):287–295, 1996.
7. K. J. Falconer and J. D. Howroyd. Packing dimensions of projections and dimension profiles. *Math. Proc. Cambridge Philos. Soc.*, 121(2):269–286, 1997.

8 Kenneth Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. Wiley, third edition, 2014.

9 Kenneth Falconer, Jonathan Fraser, and Xiong Jin. Sixty years of fractal projections. In *Fractal geometry and stochastics V*, volume 70 of *Progr. Probab.*, pages 3–25. Birkhäuser-/Springer, Cham, 2015.

10 Kenneth J. Falconer and Pertti Mattila. The packing dimension of projections and sections of measures. *Math. Proc. Cambridge Philos. Soc.*, 119(4):695–713, 1996.

11 John M. Hitchcock. Correspondence principles for effective dimensions. *Theory of Computing Systems*, 38(5):559–571, 2005.

12 J. D. Howroyd. Box and packing dimensions of projections and dimension profiles. *Math. Proc. Cambridge Philos. Soc.*, 130(1):135–160, 2001.

13 Maarit Järvenpää. On the upper Minkowski dimension, the packing dimension, and orthogonal projections. *Ann. Acad. Sci. Fenn. Ser. A I Math. Dissertationes*, page 34, 1994.

14 Robert Kaufman. On Hausdorff dimension of projections. *Mathematika*, 15:153–155, 1968.

15 Ming Li and Paul M.B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, third edition, 2008.

16 Jack H. Lutz. Dimension in complexity classes. *SIAM J. Comput.*, 32(5):1236–1259, 2003.

17 Jack H. Lutz and Neil Lutz. Algorithmic information, plane Kakeya sets, and conditional dimension. *ACM Transactions on Computation Theory*, 10(2):7:1–7:22, 2018.

18 Jack H. Lutz and Elvira Mayordomo. Dimensions of points in self-similar fractals. *SIAM J. Comput.*, 38(3):1080–1112, 2008.

19 Neil Lutz. Fractal intersections and products via algorithmic dimension. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, pages 58:1–58:12, 2017.

20 Neil Lutz and Donald M. Stull. Bounding the dimension of points on a line. In *Theory and Applications of Models of Computation - 14th Annual Conference, TAMC 2017, Bern, Switzerland, April 20-22, 2017, Proceedings*, pages 425–439, 2017.

21 J. M. Marstrand. Some fundamental geometrical properties of plane sets of fractional dimensions. *Proc. London Math. Soc. (3)*, 4:257–302, 1954.

22 Pertti Mattila. Hausdorff dimension, orthogonal projections and intersections with planes. *Ann. Acad. Sci. Fenn. Ser. A I Math.*, 1(2):227–244, 1975.

23 Pertti Mattila. *Geometry of sets and measures in Euclidean spaces: fractals and rectifiability*. Cambridge University Press, 1999.

24 Pertti Mattila. Recent progress on dimensions of projections. In *Geometry and analysis of fractals*, volume 88 of *Springer Proc. Math. Stat.*, pages 283–301. Springer, Heidelberg, 2014.

25 Elvira Mayordomo. A Kolmogorov complexity characterization of constructive Hausdorff dimension. *Inf. Process. Lett.*, 84(1):1–3, 2002.

26 Tuomas Orponen. On the packing dimension and category of exceptional sets of orthogonal projections. *Ann. Mat. Pura Appl. (4)*, 194(3):843–880, 2015.

Polynomial-Time Equivalence Testing for Deterministic Fresh-Register Automata

Andrzej S. Murawski
University of Oxford, UK

Steven J. Ramsay
University of Bristol, UK

Nikos Tzevelekos
Queen Mary University of London, UK

Abstract

Register automata are one of the most studied automata models over infinite alphabets. The complexity of language equivalence for register automata is quite subtle. In general, the problem is undecidable but, in the deterministic case, it is known to be decidable and in NP. Here we propose a polynomial-time algorithm building upon automata- and group-theoretic techniques. The algorithm is applicable to standard register automata with a fixed number of registers as well as their variants with a variable number of registers and ability to generate fresh data values (fresh-register automata). To complement our findings, we also investigate the associated inclusion problem and show that it is PSPACE-complete.

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

Keywords and phrases automata over infinite alphabets, language equivalence, bisimilarity, computational group theory

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.72

Funding Supported by EPSRC grants EP/J019577, EP/P004172.

1 Introduction

Register automata [9, 15] are one of the simplest models of computation over infinite alphabets. They operate on an infinite domain of data by storing data values in a finite number of registers, where the values are available for future comparisons or updates. The automata can also recognise when a data value does not appear in any of the registers. Fresh-register automata [20] are an extension of register automata that can, in addition, generate data values not seen so far.

In recent years, register-based automata have appeared in a variety of contexts, ranging from database query languages [18] and programming language semantics [14] to run-time verification [7]. Since the very beginning, there has been great interest in extending learning algorithms to register automata [16, 4, 1, 5, 12], driven by applications in verification [11] and system modelling [21].

Register automata are closely related to nominal automata [3], which constitute a nominal counterpart of finite-state machines. Their closure properties and associated decision problems have first been studied in [9, 15]. One of the most fundamental and applicable decision problems is that of language equivalence, not least due to connections with query equivalence,

program equivalence and learning. Unfortunately, it turns out that the equivalence problem for register automata is in general undecidable [15]. Fortunately, it is decidable in the *deterministic* case (by reduction to emptiness using closure properties [9]).

Our paper presents the first polynomial-time algorithm for the problem. The algorithm is actually applicable to a wider class of automata, namely fresh-register automata with a variable number of registers.

To begin with, we exploit the observation that in the deterministic setting, language equivalence and bisimilarity are closely related. Secondly, because in our setting only different values can be stored in different registers [9], we take advantage of symbolic representations of bisimulation relations based on partial permutations. The proposed algorithm attempts to build such a bisimulation relation incrementally. To avoid potential exponential blow-ups, the candidate relations are stored in a concise fashion through generators of symmetric groups. Thanks to the fact that group membership testing works in polynomial-time [6] and subgroup chains can only have linear length [2], we can prove that the process of refining the candidate will terminate in polynomial time. Consequently, the equivalence problem for our variant of fresh-register automata is in P, which improves upon the best upper bound known so far, namely, NP [13].

A natural question is whether the polynomial-time bound could have been obtained via the associated inclusion problem. We give a negative answer to this question by showing that the inclusion problem in our setting is PSPACE-complete.

2 Automata

Let $\mathcal{D}$ be an infinite set (alphabet). Its elements will be called *data values* (in process algebra, the term *names* is used instead). We shall work with a deterministic model of register automata over $\mathcal{D}$. As in [9], we require that different registers contain different data values. To allow for more flexible use of registers, the number of available registers will be allowed to vary according to the current state. Register content can be both erased and created. Creation can be local (new element is guaranteed not to occur in any register) or global (new element is guaranteed not to have been encountered in the whole run). We give the formal definition below. In Remark 5 we discuss the motivation behind various restrictions and their relevance to polynomial-time complexity.

▶ **Definition 1.** Given a natural number r, we write $[1, r]$ for the set $\{i \in \mathbb{N} \mid 1 \leq i \leq r\}$. An r-register assignment is an *injective* function from a subset of $[1, r]$ to $\mathcal{D}$. An **r-deterministic fresh-register automaton** (r-DFRA) is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \mu, \delta, F \rangle$, where:

- Σ is a finite alphabet of *tags*;
- Q is a finite set of states, $q_0 \in Q$ is *initial* and $F \subseteq Q$ contains *final* states;
- $\mu : Q \to \mathcal{P}([1, r])$ is the *availability* function indicating which registers are filled at each state, we require $\mu(q_0) = \emptyset$;
- $\delta = \delta_{old} + \delta_{fresh}$ is the transition function, where $\delta_{old} : Q \times \Sigma \times [1, r] \rightharpoonup Q$ controls the use of existing register values and $\delta_{fresh} : Q \times \Sigma \rightharpoonup Q \times [1, r] \times \{\bullet, \circledast\}$ indicates when fresh values are created and how fresh they are.

To preserve the meaning of μ, we insist that $\delta_{old}(q, t, i) = q'$ implies $i \in \mu(q)$ and $\mu(q) \supseteq \mu(q')$ and $\delta_{fresh}(q, t) = (q', i, x)$ implies $\mu(q) \cup \{i\} \supseteq \mu(q')$. Note the use of $\supseteq$ instead of $=$. This allows for register erasures during computation. We shall write $q \xrightarrow{t,i} q'$ for $\delta_{old}(q, t, i) = q'$ and $q \xrightarrow{t,i^x} q'$ for $\delta_{fresh}(q, t) = (q', i, x)$.

Next we formalise how to obtain a labelled transition system for a given r-DFRA.

▶ **Definition 2.** A **labelled transition system** (LTS) over $\mathcal{A}ct$ is a tuple $\mathcal{S} = (\mathcal{A}ct, \mathbb{C}, \rightarrow)$, where $\mathbb{C}$ is a set of *configurations*, $\mathcal{A}ct$ is a set of *action labels*, and $\rightarrow \subseteq \mathbb{C} \times \mathcal{A}ct \times \mathbb{C}$. We write $\kappa \xrightarrow{\ell} \kappa'$ for $(\kappa, \ell, \kappa') \in \rightarrow$. $\mathcal{S}$ is *deterministic* if $\kappa \xrightarrow{\ell} \kappa_1$ and $\kappa \xrightarrow{\ell} \kappa_2$ imply $\kappa_1 = \kappa_2$.

An r-DFRA induces a deterministic LTS as follows.

▶ **Definition 3.** Given an r-DFRA $\mathcal{A} = \langle \Sigma, Q, q_0, \mu, \delta, F \rangle$, we define its set of configurations:

$$\mathbb{C}_{\mathcal{A}} = \{(q, \rho, H) \mid q \in Q,\ \rho : \mu(q) \to \mathcal{D} \text{ is injective, } \mathsf{rng}(\rho) \subseteq H \subseteq_{fin} \mathcal{D}\}$$

We refer to H as *history*. Let $\mathcal{S}(\mathcal{A})$ be the LTS $\langle \Sigma \times \mathcal{D}, \mathbb{C}_{\mathcal{A}}, \rightarrow_{\mathcal{A}} \rangle$, where $\rightarrow_{\mathcal{A}}$ is defined in the following way: a configuration (q_1, ρ_1, H_1) can make a transition to a configuration (q_2, ρ_2, H_2) reading input (t, d), written $(q_1, \rho_1, H_1) \xrightarrow{(t,d)} (q_2, \rho_2, H_2)$, if one of the conditions listed below is satisfied (the last two cases never overlap, because δ_{fresh} is a partial function).

- $d = \rho_1(i)$, $\delta_{old}(q_1, t, i) = q_2$, $\rho_2 = (\rho_1 \upharpoonright \mu(q_2))$ and $H_2 = H_1$
- $d \notin \mathsf{rng}(\rho_1)$, $\delta_{fresh}(q_1, t) = (q_2, i, \bullet)$, $\rho_2 = (\rho_1[i \mapsto d] \upharpoonright \mu(q_2))$ and $H_2 = H_1 \cup \{d\}$
- $d \notin H_1$, $\delta_{fresh}(q_1, t) = (q_2, i, \circledast)$, $\rho_2 = (\rho_1[i \mapsto d] \upharpoonright \mu(q_2))$ and $H_2 = H_1 \cup \{d\}$

Note that $\mathcal{S}(\mathcal{A})$ does not depend on the initial and final parameters q_0 and F.

▶ **Definition 4.** The configuration $\kappa_{\mathcal{A}}^{init} = (q_0, \emptyset, \emptyset)$ will be called *initial*. A sequence of configurations $\kappa_0, \cdots, \kappa_n$ such that $\kappa_0 = \kappa_{\mathcal{A}}^{init}$ and $\kappa_i \xrightarrow{t_i, d_i} \kappa_{i+1}$ $(i = 0, \cdots, n-1)$ is called a run on the data word $(t_0, d_0) \cdots (t_{n-1}, d_{n-1})$. A run is *accepting* if $\kappa_n = (q_n, \rho_n, H_n)$ and $q_n \in F$. We write $\mathcal{L}(\mathcal{A})$ for the set of words from $(\Sigma \times \mathcal{D})^*$ with accepting runs, and call it *the language of* $\mathcal{A}$.

▶ Remark 5. Our definition allows for a variable number of available registers, i.e. it is more permissive than that in [15, 19]. This flexible register regime makes it possible to express certain common computational scenarios more directly: in particular, data values can be discarded ("forgotten") as soon as they are no longer needed (cf. garbage collection). Our result shows that poly-time equivalence testing is still possible with this added flexibility. At the same time, the flexible number of registers simplifies the technical development: one can combine an r_1-DFRA and a r_2-DFRA into a single $\max(r_1, r_2)$-DFRA (see Remark 7) by taking the disjoint union of states and transitions.

We rely on injective register assignments, as in the original definition of Francez and Kaminski [9]. This restriction is important for poly-time complexity, as the presence of multiple copies of the same value in registers could be used to model binary memory content (e.g. 1 is represented by the same value in two registers and 0 by different values). Consequently, this would imply a PSPACE lower bound. The appeal of injectivity lies in the fact no expressivity is lost but the transition function has a particularly simple shape and one can define the deterministic variant without introducing any additional comparisons between registers. While the injective discipline may seem restrictive, it has proved a good match for several prominent formalisms that arise in programming language semantics, and does not limit expressivity (e.g. [1]). For example, one can show that the automata support elegant translations from the pi-calculus [19]. They are also a natural target when it comes to investigating the semantics of programs with unbounded data – this is one of the original motivations mentioned in [9], which was also exploited in our work on the ML programming language [14].

The explicit availability function μ guarantees that whenever a transition refers to existing register content, the relevant value will be available. Allowing for transitions that may block on unavailable values is known to lead to NP-hardness [17], already for emptiness in the

deterministic case. Our variant of automata makes it possible for the automaton to drop multiple data values from registers. Conversely, values can also be created but only one at a time. Of course, such single value creations can be combined to create multiple new values. However, the new values must also occur in labels. One can imagine adding a facility for spontaneous value creation, where locally or globally fresh values would be added to the registers without being present in labels. However, the resultant non-determinism could then be used to prove universality undecidable in the same way as for nondeterministic automata, e.g. the argument from [15] could be repeated by employing spontaneous value creation to guess the location of errors. Like in [1, 12], we assume that the registers are not filled at the beginning and are initialised through transitions.

▶ **Definition 6.** A relation $R \subseteq \mathbb{C} \times \mathbb{C}$ is called a *simulation* if, for all $(\kappa_1, \kappa_2) \in R$, if $\kappa_1 \xrightarrow{t,a} \kappa_1'$ then there is $\kappa_2 \xrightarrow{t,a} \kappa_2'$ such that $(\kappa_1', \kappa_2') \in R$. R is called a **bisimulation** if both R and R^{-1} are simulations. The union of all bisimulations is denoted $\sim$. Two configurations κ_1, κ_2 are bisimilar just if $\kappa_1 \sim \kappa_2$, i.e. there is some bisimulation R containing them.

In this paper we are concerned with the *language equivalence* problem for DFRA, i.e. the question whether, given r_1-DFRA $\mathcal{A}_1$ and r_2-DFRA $\mathcal{A}_2$, we have $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$. Our approach to the problem is bisimulation-oriented: language equivalence testing of $\mathcal{A}_1$ and $\mathcal{A}_2$ can be viewed as a bisimilarity problem for a single r-DFRA with $r = \max(r_1, r_2)$.

▶ Remark 7. We explain this reduction in a little more detail. First we transform $\mathcal{A}_i$ into $\mathcal{A}_i'$ as follows:

- remove all transitions leading to states from which it is impossible to reach a final state,
- add a new state f_i and designate it as the only final state,
- add transitions from former final states to the new final state on a new tag $t^{\$}$.

Suppose $\mathcal{S}(\mathcal{A}_i') = \langle \Sigma \times \mathcal{D}, \mathbb{C}_{\mathcal{A}_i'}, \rightarrow_{\mathcal{A}_i'} \rangle$ $(i = 1, 2)$ and consider the LTS $\mathcal{S}_{\mathcal{A}_1,\mathcal{A}_2} = \langle \Sigma \times \mathcal{D}, \mathbb{C}_{\mathcal{A}_1'} + \mathbb{C}_{\mathcal{A}_2'}, \rightarrow_{\mathcal{A}_1'} + \rightarrow_{\mathcal{A}_2'} \rangle$. Now language equivalence of the original automata is equivalent to checking whether $\kappa^{init}_{\mathcal{A}_1}$ and $\kappa^{init}_{\mathcal{A}_2}$ are bisimilar in $\mathcal{S}_{\mathcal{A}_1,\mathcal{A}_2}$. If $\mathcal{A}_i' = \langle \Sigma, Q_i, q_0^i, \mu_i, \delta_i, \{f_i\} \rangle$, then let $\mathcal{S}_{\mathcal{A}_1,\mathcal{A}_2} = \mathcal{S}_{\mathcal{A}'}$, where $\mathcal{A}'$ is the $\max(r_1, r_2)$-DFRA defined by $\langle \Sigma, Q_1 + Q_2, q', \mu_1 + \mu_2, \delta_1 + \delta_2, F' \rangle$ for any $q' \in Q_1 + Q_2$ and $F' \subseteq Q_1 + Q_2$. Note, the components q', F' can be chosen arbitrarily, because they do not contribute to the definition of bisimilarity over (the configuration graph of) $\mathcal{S}_{\mathcal{A}'}$.

3 Symbolic bisimulations

In this section we introduce symbolic representations of bisimulation relations, for configuration pairs with common history,[1] based on partial permutations. A *partial permutation* over $[1, n]$ is a bijection between two (possibly different) subsets of $[1, n]$. Let $\mathcal{IS}_n$ stand for the set of partial permutations over $[1, n]$ and $\mathcal{S}_X$ for the group of permutations over X. Let us consider the kind of possible scenarios that may arise in simulating transitions of a DFRA.

- A transition on a value already stored in a register can be matched by a transition on a stored value or a locally fresh transition, but never a globally fresh one.
- A globally fresh transition can be matched by a globally fresh transition or a locally fresh one, but never a transition on a stored value.
- A locally fresh transition can be matched by a transition on a stored value, a locally fresh transition or a globally fresh one.

[1] By Remark 7, it suffices to consider configuration pairs with common history.

The use of partial bijections will help us specify which cases may occur. Although we work with automata over r registers, we shall use partial permutations over $[1, n]$, where $n = 2r$. They will be used to express not only a matching between data values occurring in two sets of r registers (corresponding to two configurations that we examine for bisimilarity) but also to indicate which values forgotten by one set are still remembered by the other.

The number $2r$ may be surprising but it is needed to provide an accurate account of scenarios in which local freshness can be simulated by global freshness. Note that this is possible if the registers of the second configuration contain all the data values that have been forgotten by the first one (i.e. do not appear in its registers any more). Once the size of the history exceeds $2r$, this is no longer possible: because the first configuration has only r registers it will have forgotten more than r data values and, because the second configuration has only r registers, it cannot remember them all. Consequently, we only need to track matches between forgotten values and register content of the other configuration as long as the size of the history does not exceed $2r$. To keep track of such scenarios, it is convenient to imagine that there are $2r$ registers available and use partial permutations to match values in registers with values that were possibly forgotten until the size of the history is at most $2r$. Once that is exceeded, matchings between the real r registers suffice.

Given $\sigma \in \mathcal{IS}_r$ and $q_1, q_2 \in Q$, we write $\sigma \restriction (q_1, q_2)$ for $\sigma \cap (\mu(q_1) \times \mu(q_2))$. Next, in accordance with the use of $2r$ registers discussed above, we introduce notions that will allow us to represent configurations in which only a subset $S \subseteq [1, 2r]$ of the registers is available along with certain values that are not stored any longer. The data values occurring in registers S will occupy the same positions (as specified by S), for other values we impose the convention that they should reside in the leftmost register positions that are unoccupied.

▶ **Definition 8** (Notation). Given $S \subseteq T \subseteq [1, 2r]$, let $S \lhd T \in \mathcal{S}_{2r}$ be the permutation that shifts all elements in $T \setminus S$ to the left (inside the interval $[1, 2r]$) without interfering with S. Formally, if $T \setminus S$ is ordered as $[i_1, \cdots, i_k]$ then:

$$S \lhd T = (i_1\, i'_1); \cdots ; (i_k\, i'_k), \quad \text{where } i'_j \text{ is the } j\text{th smallest element in } [1, 2r] \setminus S.$$

Each $(i\, i')$ denotes a transposition and ; is the composition of permutations. For example, taking $S = \{3, 6\}$ and $T = \{1, 3, 4, 6, 7\}$, the permutation would be $S \lhd T = (1\,1); (4\,2); (7\,4)$ and, therefore, $(S \lhd T)(T) = \{1, 2, 3, 4, 6\}$.
Given $S \subseteq [1, 2r]$ and $h \leq 2r$ with $|S| \leq h$, we define $S^{\lhd h}$ to be the unique T satisfying $S \subseteq T \subseteq [1, 2r]$, $|T| = h$ and $T = (S \lhd T)(T)$. In other words, $S^{\lhd h}$ is obtained by adding $h - |S|$ smallest numbers from $[1, 2r] \setminus S$ to S. For instance, for $S = \{3, 6\}$: $S^{\lhd 2} = S$, $S^{\lhd 3} = \{1, 3, 6\}$, $S^{\lhd 4} = \{1, 2, 3, 6\}$, etc. Finally, given $\sigma \in \mathcal{IS}_{2r}$ and $S_1 \subseteq \mathsf{dom}(\sigma)$, $S_2 \subseteq \mathsf{rng}(\sigma)$:

- we write: $\sigma^{(S_1, S_2)\lhd} = (S_1 \lhd \mathsf{dom}(\sigma))^{-1}; \sigma; (S_2 \lhd \mathsf{rng}(\sigma))$,
- and extend the notation to $q_1, q_2 \in Q$ by: $\sigma^{(q_1, q_2)\lhd} = \sigma^{(\mu(q_1), \mu(q_2))\lhd}$.

Next we shall introduce a symbolic notion of simulation. Pairs of configurations will be represented by elements of $\mathcal{U}_0 = Q \times \mathcal{IS}_{2r} \times Q \times ([0, 2r] \cup \{\infty\})$: each pair is represented by the states it contains and a partial permutation representing the two register assignments (a matching between their common data values). In order to handle the interaction between the two kinds of fresh transitions we also introduce an additional element storing the size of the common history (∞ stands for "bigger than $2r$"). Below we define a subset $\mathcal{U}$ of $\mathcal{U}_0$ that characterises the elements compatible with availability information. Moreover, once the history becomes larger than $2r$, we reduce the matchings to r registers only (see above).

▶ **Definition 9.** Let $\mathcal{U}_0 = Q \times \mathcal{IS}_{2r} \times Q \times ([0, 2r] \cup \{\infty\})$ and:

$$\mathcal{U} = \{ \quad (q_1, \sigma, q_2, h) \in \mathcal{U}_0 \mid \quad h \leq 2r \implies (\mathsf{dom}(\sigma) = \mu(q_1)^{\lhd h} \wedge \mathsf{rng}(\sigma) = \mu(q_2)^{\lhd h}) \\ \wedge \quad h = \infty \implies (\sigma \in \mathcal{IS}_r \wedge \sigma \subseteq \mu(q_1) \times \mu(q_2)) \}$$

Given configurations κ_1, κ_2, with $\kappa_i = (q_i, \rho_i, H)$ for some common H, we define the set of symbolic representations of (κ_1, κ_2) by:

$$\mathsf{symb}(\kappa_1, \kappa_2) = \begin{cases} \{(q_1, \rho_1; \rho_2^{-1}, q_2, \infty)\} & |H| > 2r \\ \{(q_1, (\hat{\rho}_1; \hat{\rho}_2^{-1})^{\lhd(q_1,q_2)}, q_2, |H|) \mid \rho_i \subseteq \hat{\rho}_i \wedge \mathsf{rng}(\hat{\rho}_i) = H\} & |H| \leq 2r \end{cases}$$

The essence of the above representation is the abstracting away from the register assignments ρ_1, ρ_2 to a partial permutation $\sigma \in \mathcal{IS}_{2r}$. If the history is large, then σ is simply a matching between the common values of ρ_1 and ρ_2. If, on the other hand, H contains at most $2r$ elements then σ is obtained by extending each ρ_i to some $\hat{\rho}_i$ that stores the full history H, and these pairs $(\hat{\rho}_1, \hat{\rho}_2)$ are then represented by recording their indices containing matching values.

We proceed with defining symbolic bisimulations. The clauses (a)-(f) in the definition below cover all possible kinds of simulation scenarios. Partial bijections help to capture the conditions under which simulation is possible.

▶ **Definition 10.** Let $\mathcal{A} = \langle \Sigma, Q, q_0, \mu, \delta, F \rangle$ be an r-DFRA. A *symbolic simulation* on $\mathcal{A}$ is a relation $R \subseteq \mathcal{U}$, with elements $(q_1, \sigma, q_2, h) \in R$ written $q_1 \, R^h_\sigma \, q_2$, such that all $(q_1, \sigma, q_2, h) \in R$ satisfy the (FSyS) conditions in R. We say that a tuple (q_1, σ, q_2, h) satisfies the *fresh symbolic simulation conditions* (FSyS) *in* R if the following conditions hold, where (a-c) apply to $h \leq 2r$, and (d-e) to $h = \infty$:

(a) for all $q_1 \xrightarrow{t,i} q_1'$,
  1. if $\sigma(i) \in \mu(q_2)$ then there is some $q_2 \xrightarrow{t,\sigma(i)} q_2'$ with $q_1' \, R^h_{\sigma'} \, q_2'$ and $\sigma' = \sigma^{(q_1', q_2')\lhd}$,
  2. if $\sigma(i) = j' \in [1, 2r] \setminus \mu(q_2)$ then there is some $q_2 \xrightarrow{t,j^\bullet} q_2'$ with $q_1' \, R^h_{\sigma'} \, q_2'$ and $\sigma' = (\sigma; (j \, j'))^{(q_1', q_2')\lhd}$;

(b) for all $q_1 \xrightarrow{t,i^\bullet} q_1'$ and $i' \in \mathsf{dom}(\sigma) \setminus \mu(q_1)$,
  1. if $\sigma(i') \in \mu(q_2)$ then there is some $q_2 \xrightarrow{t,\sigma(i')} q_2'$ with $q_1' \, R^h_{\sigma'} \, q_2'$ and $\sigma' = ((i \, i'); \sigma)^{(q_1', q_2')\lhd}$,
  2. if $\sigma(i') = j' \in [1, 2r] \setminus \mu(q_2)$ then there is some $q_2 \xrightarrow{t,j^\bullet} q_2'$ with $q_1' \, R^h_{\sigma'} \, q_2'$ and $\sigma' = ((i \, i'); \sigma; (j \, j'))^{(q_1', q_2')\lhd}$;

(c) for all $q_1 \xrightarrow{t,\ell_i} q_1'$ with $\ell_i \in \{i^\bullet, i^\circledast\}$ there is some $q_2 \xrightarrow{t,\ell_j} q_2'$ with $\ell_j \in \{j^\bullet, j^\circledast\}$ and,
  1. if $h < 2r$ then $q_1' \, R^{h+1}_{\sigma'} \, q_2'$ with $\sigma' = ((i \, 2r); \sigma[2r \mapsto 2r]; (j \, 2r))^{(q_1', q_2')\lhd}$,
  2. if $h = 2r$ then $q_1' \, R^\infty_{\sigma'} \, q_2'$ with $\sigma' = \sigma[i \mapsto j] \restriction (q_1', q_2')$;

(d) for all $q_1 \xrightarrow{t,i} q_1'$,
  1. if $\sigma(i) \in \mu(q_2)$ then there is some $q_2 \xrightarrow{t,\sigma(i)} q_2'$ with $q_1' \, R^\infty_{\sigma'} \, q_2'$ and $\sigma' = \sigma \restriction (q_1', q_2')$,
  2. if $i \in \mu(q_1) \setminus \mathsf{dom}(\sigma)$ then there is some $q_2 \xrightarrow{t,j^\bullet} q_2'$ with $q_1' \, R^\infty_{\sigma'} \, q_2'$ and $\sigma' = \sigma[i \mapsto j] \restriction (q_1', q_2')$;

(e) for all $q_1 \xrightarrow{t,i^\bullet} q_1'$ and $j \in \mu(q_2) \setminus \mathsf{rng}(\sigma)$, there exists $q_2 \xrightarrow{t,j} q_2'$ with $q_1 \, R^\infty_{\sigma'} \, q_2'$ and $\sigma' = \sigma[i \mapsto j] \restriction (q_1', q_2')$;

(f) for all $q_1 \xrightarrow{t,\ell_i} q_1'$ with $\ell_i \in \{i^\bullet, i^\circledast\}$ there is some $q_2 \xrightarrow{t,\ell_j} q_2'$ with $\ell_j \in \{j^\bullet, j^\circledast\}$, $q_1' \, R^\infty_{\sigma'} \, q_2'$ and $\sigma' = \sigma[i \mapsto j] \restriction (q_1', q_2')$, and $\ell_i = i^\bullet \implies \ell_j = j^\bullet$.

Define now the inverse of R by $R^{-1} = \{\,(q_2, \sigma^{-1}, q_1, h) \mid (q_1, \sigma, q_2, h) \in R\,\}$ and call R a **symbolic bisimulation** if both R and R^{-1} are symbolic simulations. We let *s-bisimilarity*, denoted $\overset{\mathsf{s}}{\sim}$, be the union of all symbolic bisimulations.

In the rest of the paper, given $R \subseteq \mathcal{U}$ and $h \in [1, 2r] \cup \{\infty\}$, we shall write R^h for the projection of R on h: $R^h = \{(q, \sigma, q') \mid (q, \sigma, q', h) \in R\}$.

▶ Remark 11. To gain some further intuition about the definition above, let us consider the 2-DFRA configurations $\kappa_i = (q_i, \rho_i, H)$, $i = 1, 2$, where: $\mu(q_1) = \mu(q_2) = \{1, 2\}$, $\rho_1 = \{(1, a), (2, b)\}$, $\rho_2 = \{(1, a), (2, c)\}$ and $H = \{a, b, c\}$.
The pair (κ_1, κ_2) can be represented symbolically by $(q_1, \sigma, q_2, 3) \in \mathsf{symb}(\kappa_1, \kappa_2) \subseteq \mathcal{U}$, where $\sigma = \{(1, 1), (2, 3), (3, 2)\}$. This represents the fact that ρ_1, ρ_2 share the data value a in their first register and each have a private value in their second register.[2] The (FSYS) conditions express symbolically what it takes for κ_2 to simulate κ_1, i.e. what is needed for $(q_1, \sigma, q_2, 3)$ to belong to a (symbolic) simulation R. Let us look at two sample cases.

- Suppose $q_1 \xrightarrow{t,1} q_1'$. Then, $\kappa_1 \xrightarrow{(t,a)} \kappa_1'$ and, in order for κ_2 to match this, it must be the case that $q_2 \xrightarrow{t,1} q_2'$. This is imposed by Condition (a)1 of (FSYS).
- If $q_1 \xrightarrow{t,2} q_1'$ then $\kappa_1 \xrightarrow{(t,b)} \kappa_1'$. Then, κ_2 can only match a transition on b using a locally fresh transition (Condition (a)2), so we must have e.g. $q_2 \xrightarrow{t,1^\bullet} q_2'$, yielding some $\kappa_2 \xrightarrow{(t,b)} \kappa_2'$.

In each of the cases above, the (FSYS) conditions also stipulate that the resulting representation of (κ_1', κ_2') must also be in R. In the second case, assuming $\kappa_i' = (q_i', \rho_i', H)$ and $\mu(q_i') = \{1, 2\}$, we have that $\rho_1' = \{(1, a), (2, b)\}$ and $\rho_2' = \{(1, b), (2, c)\}$, and the pair (κ_1', κ_2') is represented by $(q_1', \sigma', q_2', 3)$ with $\sigma' = \{(1, 3), (2, 1), (3, 2)\}$. Because $\sigma' = \sigma; (3\ 1) = (\sigma; (3\ 1))^{(q_1', q_2')\triangleleft}$, the (FSYS) conditions require $(q_1', \sigma', q_2', 3) \in R$.

The importance of symbolic bisimulations lies in that they precisely represent actual bisimulations in a finite way. Below, we first show that the symbolic representations of pairs of configurations are well defined (the choice of extensions $\hat{\rho}_i$ for the case of $|H| \leq 2r$ does not matter for $\overset{\mathsf{s}}{\sim}$), and then prove the representation property.

▶ **Lemma 12.** *For any κ_1, κ_2 with $\kappa_i = (q_i, \rho_i, H)$ and $|H| \leq 2r$, either $\mathsf{symb}(\kappa_1, \kappa_2) \subseteq \overset{\mathsf{s}}{\sim}$ or $\mathsf{symb}(\kappa_1, \kappa_2) \cap \overset{\mathsf{s}}{\sim} = \emptyset$.*

▶ **Proposition 13.** For any κ_1, κ_2 with common history, $\kappa_1 \sim \kappa_2$ iff $\mathsf{symb}(\kappa_1, \kappa_2) \subseteq \overset{\mathsf{s}}{\sim}$.

Although finite, symbolic bisimulations are of exponential size in the worst case (with respect to the automaton size) because of including the partial bijections σ. Our equivalence-testing algorithm for r-DFRA will rely on representations of candidate symbolic bisimulations in a succinct way. In order to spell out in what sense these representations will capture subsets of $\mathcal{U}$ we need to introduce the following closure operations.

▶ **Definition 14.** Let $R \subseteq \mathcal{U}$. Then $Cl(R)$ is defined to be the smallest subset X of $\mathcal{U}$ such that $R \subseteq X$ and X is closed under the following rules.

$$\frac{S = \mu(q)^{\triangleleft h} \quad h \leq 2r}{(q, \mathsf{id}_S, q) \in X^h} \qquad \frac{}{(q, \mathsf{id}_{\mu(q)}, q) \in X^\infty} \qquad \frac{(q_1, \sigma, q_2) \in X^h}{(q_2, \sigma^{-1}, q_1) \in X^h}$$

$$\frac{(q_1, \sigma, q_2) \in X^\infty \quad \sigma \subseteq \sigma'}{(q_1, \sigma', q_2) \in X^\infty} \qquad \frac{(q_1, \sigma_1, q_2) \in X^h \quad (q_2, \sigma_2, q_3) \in X^h}{(q_1, \sigma_1; \sigma_2, q_3) \in X^h}$$

[2] E.g. the value b is in register 2 of ρ_1 but is not present in ρ_2. Seeing $\hat{\rho}_2$ as an expansion of ρ_2 to 3 registers (with register 3 containing forgotten values), we set $\hat{\rho}_2(3) = b$ and therefore $\sigma(2) = 3$.

The next lemma provides a handle on proving that closures $Cl(R)$ satisfy (FSyS) conditions.

▶ **Lemma 15.** *Let $R, P \subseteq \mathcal{U}$ with $R = R^{-1}$. If all $g \in R$ satisfy the* (FSyS) *conditions in P then all $g' \in Cl(R)$ satisfy the* (FSyS) *conditions in $Cl(P)$.*

▶ **Corollary 16.** $Cl(\overset{s}{\sim}) = \overset{s}{\sim}$.

Proof. It suffices to show the left-to-right inclusion. All elements in $\overset{s}{\sim}$ satisfy the (FSyS) conditions in $\overset{s}{\sim}$. Hence, by the previous lemma, all elements of $Cl(\overset{s}{\sim})$ satisfy the (FSyS) conditions in $Cl(\overset{s}{\sim})$. This implies that $Cl(\overset{s}{\sim})$ is a symbolic bisimulation. Thus, $Cl(\overset{s}{\sim}) \subseteq \overset{s}{\sim}$. ◀

▶ Remark 17. One may wonder to what extent our techniques apply to simulation rather than bisimulation. Although symbolic simulation can be related to simulation, our methods crucially exploit the fact that bisimilarity is symmetric. This is reflected in the top right rule of Definition 14, which introduces inverses, and enables us to develop a group-theoretic representation scheme in the next section.

4 Representation

Our algorithm for DFRA equivalence will rely on manipulating sets $\mathcal{H} \subseteq \mathcal{U}$ that, for positive instances, will ultimately converge to a symbolic bisimulation relation. We shall handle them through succinct representations based on group theory, whose shape is inspired by the structure of bisimulation relations [13]. The backbone of a generating system, to be defined next, is an equivalence relation $\diamond^h$ on states. As explained in Definition 19, the relation specifies which pairs of states may actually feature in tuples of the represented subset of $\mathcal{U}$.

▶ **Definition 18.** A *generating system* $\mathcal{R}$ consists of a set $\{\mathcal{R}^h \mid h \in [0, 2r] \cup \{\infty\}\}$, where each $\mathcal{R}^h = \langle \diamond^h, \{(q_C^h, X_C^h, G_C^h) \mid C \in Q/\diamond^h\}, \{\sigma_q^h \mid q \in Q\}\rangle$ satisfies the following constraints.

- $\diamond^h \subseteq Q \times Q$ is an equivalence relation.
- For any $\diamond^h$-equivalence class C:
  - q_C^h is a state from C (class representative);
  - $X_C^h = \mu(q_C^h)^{\triangleleft h}$ for $h \in [0, 2r]$ and $X_C^\infty \subseteq \mu(q_C^\infty)$;
  - $\emptyset \neq G_C^h \subseteq \mathcal{S}_{X_C^h}$.
- For any $q \in Q$, $C = [q]_{\diamond^h}$ and $h \in [0, 2r]$, we have $\sigma_q^h \in \mathcal{IS}_{2r}$ with $\mathsf{dom}(\sigma_q^h) = \mu(q_C^h)^{\triangleleft h}$ and $\mathsf{rng}(\sigma_q^h) = \mu(q)^{\triangleleft h}$. Moreover, $\sigma_q^\infty \in \mathcal{IS}_r$ and $\mathsf{dom}(\sigma_q^\infty) = X_C^\infty$. Finally, $\sigma_{q_C^h}^h = \mathsf{id}_{X_C^h}$.

Thus, at each level h, a generating system partitions the set of states into equivalence classes according to $\diamond^h$ and each class has a representative q_C^h, which is "connected" to each element of the class via σ_q^h. Each representative q_C^h is also equipped with a subset $X_C^h \subseteq [0, 2r]$ and a set G_C^h of permutations (generators) from $\mathcal{S}_{X_C^h}$.

▶ **Definition 19.** Let $\mathcal{R}$ be a generating system. The subset of $\mathcal{U}$ represented by $\mathcal{R}$, written $\mathsf{Gen}(\mathcal{R})$, is defined to be $Cl(\mathcal{H}_\mathcal{R})$, where $\mathcal{H}_\mathcal{R} = \bigcup_{h=0}^{2r} \mathcal{H}_\mathcal{R}^h \cup \mathcal{H}_\mathcal{R}^\infty$ and, for any $h \in [0, 2r] \cup \{\infty\}$, we take $\mathcal{H}_\mathcal{R}^h = \{\, (q_C^h, g_C^h, q_C^h, h) \mid C \in Q/\diamond^h,\ g_C^h \in G_C^h \,\} \cup \{\, (q_C^h, \sigma_q^h, q, h) \mid q \in Q, C = [q]_{\diamond^h} \,\}$.

▶ **Example 20.** The representation system $\mathcal{R}_{init}$ is defined by the following components.

- $\diamond^h = \{(q, q) \mid q \in Q\}$. Note that $[q]_{\diamond^h} = \{q\}$.
- For any equivalence class $C = \{q\}$ we have: $q_C^h = q$, $X_C^h = \mu(q)^{\triangleleft h}$ ($h \in [0, 2r]$), $X_C^\infty = \mu(q)$, $G_C^h = \{\mathsf{id}_{X_C^h}\}$.
- For any q, $\sigma_q^h = \mathsf{id}_{X_C^h}$.

Note that $\mathsf{Gen}(\mathcal{R}_{init}) = Cl(\emptyset)$.

Next we examine how generating systems can be employed in algorithms. We are particularly interested in membership testing and a special kind of updates.

4.1 Membership

The next lemma reduces testing for membership in $\mathsf{Gen}(\mathcal{R})$ to the classic problem of group membership testing [6]. Given $G \subseteq \mathcal{S}_X$, we let $\mathsf{Sub}(G)$ be the subgroup of $\mathcal{S}_X$ spanned by G.

▶ **Lemma 21.** *Let $\mathcal{R}$ be a generating system, $u = (q_1, \sigma, q_2, h) \in \mathcal{U}$ and $\overline{\sigma} = \sigma^h_{q_1}; \sigma; (\sigma^h_{q_2})^{-1}$. Then $u \in \mathsf{Gen}(\mathcal{R})$ if and only if $q_1 \diamond^h q_2$ and $\overline{\sigma} \in \mathsf{Sub}(G^h_C)$, where $C = [q_1]_{\diamond^h} = [q_2]_{\diamond^h}$.*

4.2 Update

Suppose $\mathsf{Gen}(\mathcal{R}) = Cl(\mathcal{H})$. We explain how, given $u = (q_1, \sigma, q_2, h) \in \mathcal{U}$, one can update $\mathcal{R}$ to $\mathcal{R}'$ so that $\mathsf{Gen}(\mathcal{R}') = Cl(\mathcal{H} \cup \{u\})$. Of course, if $u \in \mathsf{Gen}(\mathcal{R})$ then it suffices to take $\mathcal{R}' = \mathcal{R}$. Thus, let us assume $u \notin \mathsf{Gen}(\mathcal{R})$. By Lemma 21, this corresponds to the following cases, where $\overline{\sigma} = \sigma^h_{q_1}; \sigma; (\sigma^h_{q_2})^{-1}$.

1. $q_1 \diamond^h q_2$ and either (a) or (b) holds, where $C = [q_1]_{\diamond^h} = [q_2]_{\diamond^h}$:
   - **(a)** $\overline{\sigma} \in \mathcal{S}_{X^h_C} \setminus \mathsf{Sub}(G^h_C)$,
   - **(b)** $\overline{\sigma} \notin \mathcal{S}_{X^h_C}$, i.e. $\mathsf{dom}(\overline{\sigma}) \subsetneq X^h_C$.
2. $q_1 \diamond^h q_2$ does not hold.

Observe that 1.(b) will never arise for $h \neq \infty$ due to the definitions of $\mathcal{U}$ and $\mathcal{R}$. Note also that, for $h \neq \infty$, X^h_C is uniquely determined by q^h_C. However, this is not the case for X^∞_C.

Before we explain how to tackle each case, we introduce several technical lemmas that examine how partial permutations interact. They will inform the performance of updates based on modifying X^∞_C.

▶ **Lemma 22.** *Given $I \subseteq \mathcal{IS}_r$, let $\chi_I = \{\sigma \mid \sigma = \sigma_1^{\epsilon_1}; \cdots; \sigma_k^{\epsilon_k},\ k > 0,\ \sigma_i \in I,\ \epsilon_i \in \{1, -1\}\}$ and $\mathcal{D}_I = \{\mathsf{dom}(\sigma) \mid \sigma \in \chi_I\}$. Then χ_I is closed under composition and inversion, and $\mathcal{D}_I$ is closed under intersection.*

▶ **Lemma 23.** *Given $I \subseteq \mathcal{IS}_r$, let $B_I = \bigcap_{X \in \mathcal{D}_I} X$ be called the* base *of I. Then we have:*

1. *$B_I \in \mathcal{D}_I$ and $\mathsf{id}_{B_I} \in \chi_I$.*
2. *Given $\sigma \in \mathcal{IS}_r$ and $X \subseteq [1, r]$, let us write $\sigma \restriction X$ for $\mathsf{id}_X; \sigma$. Then, for any $\sigma \in I$, $\sigma \restriction B_I \in \chi_I$ and $\sigma \restriction B_I$ is a permutation of B_I.*

Next we show that, given I, the base B_I can be calculated via graph reachability.

▶ **Lemma 24.** *Let $I \subseteq \mathcal{IS}_r$. Consider the undirected graph $G_I = (V, E)$ with $V = [1, r]$, where $\{j_1, j_2\} \in E$ iff there exists $\sigma \in I$ such that $\sigma(j_1) = j_2$ or $\sigma(j_2) = j_1$. We shall call $v \in [1, r]$* endangered *if there exists $\sigma \in I$ such that $v \notin \mathsf{dom}(\sigma)$ or $v \notin \mathsf{rng}(\sigma)$. For any $i \in [1, r]$, $i \in B_I$ if and only if no endangered vertex is reachable from i in G_I.*

4.3 Update implementation

Finally, we are ready to return to the main issue of representation update. We discuss the three cases (1.(a), 1.(b) and 2.) in turn. Recall that $u = (q_1, \sigma, q_2, h) \in \mathcal{U}$ and $\overline{\sigma} = \sigma^h_{q_1}; \sigma; (\sigma^h_{q_2})^{-1}$.

1. **(a)** Here we have $\overline{\sigma} \in \mathcal{S}_{X^h_C} \setminus \mathsf{Sub}(G^h_C)$. To update the system in order to represent σ, it suffices to add $\overline{\sigma}$ to G^h_C without changing anything else.

1. **(b)** Here we have $\mathsf{dom}(\overline{\sigma}) \subsetneq X^h_C$ and $h = \infty$. In order to capture $\overline{\sigma}$, we replace X^∞_C with B_I, where $I = G^\infty_C \cup \{\overline{\sigma}\}$, and set $G^\infty_C = \{\sigma \restriction B_I \mid \sigma \in I\}$. Note that, by Lemma 23, all the elements are permutations, as required. Similarly to G^∞_C, we replace σ^∞_q with $\sigma^\infty_q \restriction B_I$ for each $q \in C$. Other elements of the system remain the same.

```
1  i=0; R_0 = R_init; Δ = {u_0}; Δ_0 = ∅;
2  while (Δ is not empty) do {
3      u = Δ.get();
4      if u ∉ Gen(R_i) {
5          if one-step test fails for u return NO;
6          Δ.add(succ-set(u));
7          Δ_{i+1} = Δ_i.add({u});
8          R_{i+1} = R_i updated with u;
9          i=i+1;
10     }
11 }
12 return YES
```

Figure 1 Bisimilarity checking algorithm.

2. This case is the hardest as we need to merge two different equivalence classes, namely, $C_1 = [q_1]_{\diamond^h}$ and $C_2 = [q_2]_{\diamond^h}$ into a single one $C = C_1 \cup C_2$ (formally, this is a change to $\diamond^h$). For the new class C, we take $q^h_C = q^h_{C_1}$.
   Next we discuss $X^h_{q_C}$. Given $\tau \in G^h_{q_{C_2}}$, let $\widehat{\tau} = \overline{\sigma}; \tau; (\overline{\sigma})^{-1}$ and consider $I = G^h_{q_{C_1}} \cup \{\widehat{\tau} \mid \tau \in G^h_{q_{C_2}}\}$. We shall set $X^h_{q_C}$ to B_I. Note that, if $h \neq \infty$, all elements of I will have the same domains, so in this case $X^h_{q_C}$ will not change. As before, we set $G^h_C = \{\sigma \restriction B_I \mid \sigma \in I\}$. We also modify σ^h_q, but only for $q \in C_1 \cup C_2$. If $q \in C_1$, we take $\sigma^h_q \restriction B_I$ instead of σ^h_q. For $q \in C_2$, we need to take the change of representative into account and take $(\overline{\sigma}; \sigma^h_q) \restriction B_I$ instead of σ^h_q.
   (For this to be a correct choice, we need to show that $\mathsf{dom}(\overline{\sigma}; \sigma^h_q) \supseteq B_I$. This is indeed so, because $\mathsf{dom}(\overline{\sigma}; \sigma^h_q) = \mathsf{dom}(\overline{\sigma}; \mathsf{id}_{X^h_{q_{C_2}}})$, by $\mathsf{dom}(\sigma^h_q) = X^h_{q_{C_2}}$, and $\mathsf{dom}(\overline{\sigma}; \mathsf{id}_{X^h_{q_{C_2}}}) = \mathsf{dom}(\overline{\sigma}; \tau) \supseteq \mathsf{dom}(\overline{\sigma}; \tau; \overline{\sigma}^{-1}) = \mathsf{dom}(\widehat{\tau}) \supseteq B_I$ for any $\tau \in G^h_{q_{C_2}}$.)

Recall that we work under the assumption that $\mathsf{Gen}(\mathcal{R}) = Cl(\mathcal{H})$ and let us write $\mathcal{R}'$ for the updated representation system. In each of the above cases, the modifications contribute to $\mathcal{H}_{\mathcal{R}'}$ only elements from $Cl(\mathcal{H} \cup \{u\})$. This is completely clear for 1.(a). For 1.(b) and 2., we need to appeal to Lemma 23 ($\sigma \restriction B_I \in \chi_I$) and the use of composition/inversion during construction. Consequently, $\mathsf{Gen}(\mathcal{R}') \subseteq Cl(\mathcal{H} \cup \{u\})$.

Conversely, $Cl(\mathcal{H} \cup \{u\}) \subseteq \mathsf{Gen}(\mathcal{R}')$, because all elements of $\mathcal{R}$ as well as u have been integrated into $\mathcal{R}'$, either directly or through composition and reductions to X^∞_C. Thanks to the defining rules for Cl (notably, closure under composition and inclusion), such changes preserve representability.

5 Algorithm

Finally, we present the algorithm for deciding whether two configurations $\kappa_i = (q_i, \rho_i, H)$ are bisimilar. Let $u_0 = (q_1, \sigma, q_2, h)$ be an arbitrary element of $\mathsf{symb}(\kappa_1, \kappa_2)$. By Lemma 12 and Proposition 13, bisimilarity of κ_1, κ_2 amounts to checking whether u_0 belongs to a symbolic bisimulation. Our algorithm will determine whether or not this is the case.

The algorithm is presented in Figure 1. It is similar in flavour to the classic Hopcroft-Karp algorithm for DFA [8], which maintains *sets of pairs of states*. In contrast, we work with *sets of elements from the set* $\mathcal{U}$, i.e. four-tuples (q_1, σ, q_2, h). As subsets of $\mathcal{U}$ may have exponential size, we do not store them explicitly. Instead we take advantage of the representation systems developed in the previous section.

Starting from u_0, the algorithm maintains generating systems $\mathcal{R}_i$, beginning with $\mathcal{R}_{init}$. We assume the availability of a data structure Δ for storing multisets of elements of $\mathcal{U}$ (e.g. a queue), equipped with emptiness testing, a *get* method that removes an occurrence of an element u from Δ and returns it as a result, and an *add* method that extends Δ with the elements listed as its argument .

▶ Remark 25. Each of the conditions for (FSYS) relies on finding a matching transition satisfying an extra constraint spelt out in terms of R^h. If (FSYS) fails for u or u^{-1} because no potential transition exists, we shall say that the *one-step* test fails for $u \in \mathcal{U}$. Note that we are not concerned whether the extra constraint is satisfied – we only check if a transition with the specified source and label exists.
Because we work with deterministic automata, the availability of a transition implies uniqueness. Consequently, if u passes the one-step test, the (FSYS) rules for u and u^{-1} deliver a unique set of conditions that need to be checked in order for (FSYS) to be satisfied (for u and u^{-1}). Formally, these conditions can be captured as a subset of $\mathcal{U}$ and we shall call them the *successor set* of u, written $\mathsf{succ\text{-}set}(u)$. In the code above the membership test ($u \notin \mathsf{Gen}(\mathcal{R}_i)$) is performed as specified in Section 4.1, while the extension of $\mathcal{R}_i$ with u follows Section 4.2.

The correctness arguments rely on the following invariants.

▶ **Lemma 26.** *The loop satisfies the following invariants.*

(a) *For any $i \geq 0$, $\mathsf{Gen}(\mathcal{R}_i) = Cl(\Delta_i)$ and, for all $v \in \Delta_i$, v, v^{-1} satisfy the* (FSYS) *conditions in $Cl(\Delta_i \cup \Delta)$.*

(b) *For any symbolic bisimulation relation R, if $u_0 \in R$ then $\Delta \subseteq R$.*

▶ **Theorem 27** (Partial Correctness)**.** *When the Algorithm returns YES, there exists a symbolic bisimulation containing u_0. When the Algorithm returns NO, no symbolic bisimulation can contain u_0.*

Proof. When the Algorithm returns YES, Δ is empty. Consequently, Lemma 26 (a) implies that each element of $\Delta_i \cup \Delta_i^{-1}$ satisfies the (FSYS) conditions in $Cl(\Delta_i)$, so $Cl(\Delta_i)$ is a symbolic bisimulation relation by Lemma 15.

- If $u_0 \notin \mathsf{Gen}(\mathcal{R}_{init})$ then $i > 0$ and $u_0 \in \Delta_0 \subseteq \Delta_i$. Thus, $u_0 \in Cl(\Delta_i)$.
- If $u_0 \in \mathsf{Gen}(\mathcal{R}_{init})$ then the Theorem is also true, because $\mathsf{Gen}(\mathcal{R}_{init})$ is a symbolic bisimulation.

Thus, in each case, there exists a symbolic bisimulation containing u_0. The NO case follows immediately from Lemma 26 (b). ◀

Next we argue why the algorithm terminates and its complexity is polynomial. To that end, it will be useful to introduce the following measure on representation systems.

▶ **Definition 28.** Given $\mathcal{R}$, let $m_{\mathcal{R}} : ([0, 2r] \cup \{\infty\}) \times Q \to \mathbb{N} \times \mathcal{P}(\mathcal{IS}_{2r})$ be defined as follows.

$$m_{\mathcal{R}}(h, q) = (|Q/\diamond^h| + |X_{[q]_{\diamond^h}}|, \mathsf{Sub}(G^h_{[q]_{\diamond^h}}))$$

Given $(n_1, H_1), (n_2, H_2) \in \mathbb{N} \times \mathcal{P}(\mathcal{IS}_{2r})$, let $(n_1, H_1) \leq (n_2, H_2)$ stand for $n_1 < n_2$ or ($n_1 = n_2$ and $H_1 \supseteq H_2$). For $\mathcal{R}_1, \mathcal{R}_2$, we then write $m_{\mathcal{R}_1} \leq m_{\mathcal{R}_2}$ iff for all (h, q), $m_{\mathcal{R}_1}(h, q) \leq m_{\mathcal{R}_2}(h, q)$.

▶ **Lemma 29.** *Given a representation system $\mathcal{R}$ and $u \in \mathcal{U}$, let $\mathcal{R}'$ be its extension by u constructed in Section 4.2. Then $m_{\mathcal{R}'} \lneq m_{\mathcal{R}}$.*

▶ **Theorem 30.** *The Algorithm terminates.*

Proof. We argue by contradiction. Observe that, if the Algorithm does not terminate, there can be no bound on the number of times that elements are added to the queue. This will generate an infinite sequence of generating systems $\mathcal{R}_0, \mathcal{R}_1, \cdots, \mathcal{R}_i, \mathcal{R}_{i+1}, \cdots$, where each $\mathcal{R}_{i+1}$ extends $\mathcal{R}_i$ according to Section 4.2. By Lemma 29, $m_{\mathcal{R}_0} \gneq m_{\mathcal{R}_1} \gneq \cdots \gneq m_{\mathcal{R}_i} \gneq \cdots$. Given that the first components (numbers) in $m_{\mathcal{R}i}(h, q)$ are bounded by $|Q| + 2r$, for this to happen, we would need to have an infinite chain of subgroups of $\mathcal{S}_X$ for some $X \subseteq [1, 2r]$. This contradicts the bound from [2]. ◀

Following a similar pattern of reasoning, we can establish a bound on the number of generating systems that can be produced by the Algorithm, which happens to correspond to the value of i. We have already observed that the integers in the first component of $m_{\mathcal{R}i}(h, q)$ are bounded by $|Q| + 2r$. Consequently, that particular component can decrease $|Q| + 2r$ times for $h \in [0, 2r]$ and $|Q|$ times for $h = \infty$ (the sets X^h_C are not modified in this case). As for the second component, the bound on the number of times it can change is $2r + O(1)$ [2]. Because the decreases may occur for any q, h, the overall bound on i is $\underbrace{|Q|(2r+1)(|Q|+2r)2r}_{h\in[0,2r]} + \underbrace{|Q||Q|2r}_{h=\infty} = O(|Q|^2r^2 + |Q|r^3) + O(|Q|^2r)$. Each increase of i is accompanied by the addition of one-step successors to the queue. There are $O(r)$ such successors and their generation can take $O(r)$ steps due to rearrangements on permutations. Consequently, the handling of each element of u may require $O(r^2)$ steps ($O(r)$ steps for $h = \infty$). This does not take group membership tests into account, for which there exist polynomial-time algorithms [6]. Thus, the complexity can be conservatively bounded by $O(|Q|^2r^5p(r))$ steps, where $p(r)$ refers to the complexity of membership testing for $\mathcal{S}_{2r}$ (which bounds those for $\mathcal{S}_X$, where $X \subseteq [1, 2r]$). Note that for $h = \infty$, the complexity is $O(|Q|^2r^2p(r))$. Knuth [10] reports on an algorithm for which $p(r) = O(r^5 + mr^2)$, where m is the number of membership queries, adding that it runs considerably faster in practice.

▶ **Theorem 31.** *The language equivalence problem for r-DFRA is in PTIME.*

A natural question for further study is whether the problem is PTIME-complete. It is certainly NL-hard, by reduction from DFA.

Implementation

An implementation of our algorithm is available from `http://github.com/stersay/deq`. Although we leave a full analysis of our empirical results to a future publication, it is worth mentioning that initial case studies indicate that the high-degree of r in the worst case is not a hindrance in practice. For example, in comparing two encodings of automata simulating finite stack machines (considered previously by [12]), bisimulations for automata with $r \leq 1500$ can be computed in less than one minute.

6 Inclusion

Equivalence can often be attacked by reduction to the associated inclusion problem. As we explain next, for DFRA this route would not yield a PTIME bound.

▶ **Theorem 32.** *The inclusion problem for r-DFRA is in PSPACE-complete.*

Proof. For membership in PSPACE, we first note that inclusion can be reduced to simulation. Now observe that if there is a winning strategy for Attacker over the infinite alphabet then there will be one if $2r + 1$ letters are used. This is because $2r + 1$ letters are sufficient to

simulate the effect of attacks that rely on global freshness: with $2r + 1$ letters available it is always possible to choose a letter that is not stored in either set of the r-registers and, thus, attacks based on global freshness can be simulated. Consequently, failures of inclusion can be detected by guessing the relevant word using $2r + 1$ letters on the understanding that for globally fresh transitions we need to choose a letter not occurring in any of the $2r$ registers. To this end, polynomial space is needed to keep track of the current content of both sets of registers.

We can show PSPACE-hardness already for DFRA without global freshness, which we refer to as DRA. Because DRA can be complemented easily, we actually show that the equivalent problem of DRA intersection emptiness is PSPACE-hard. This is done by reduction from non-emptiness of deterministic linear-bounded Turing machines. The main difficulty in the argument is to represent the tape through registers. This seems impossible at first given that a register assignment must contain different data values. We overcome this by constructing two $(n + 1)$-DRA $\mathcal{A}_1$, $\mathcal{A}_2$ such that whenever they synchronise on a data word, their register assignments ρ_1, ρ_2 represent the content of n tape cells as follows: 0 in the ith cell is represented by $\rho_1(i) = \rho_2(i)$, and 1 by $\rho_1(i) \notin \mathsf{rng}(\rho_2)$. The $(n + 1)$th register plays a technical role that helps us to maintain the representation. The position of the head and state of the machine are maintained in the state of the automata. ◀

References

1. F. Aarts, P. Fiterau-Brostean, H. Kuppens, and F. W. Vaandrager. Learning register automata with fresh value generation. In *Proceedings of ICTAC*, volume 9399 of *Lecture Notes in Computer Science*, pages 165–183. Springer, 2015.
2. L. Babai. On the length of subgroup chains in the symmetric group. *Communications in Algebra*, 14(9):1729–1736, 1986.
3. M. Bojańczyk, B. Klin, and S. Lasota. Automata theory in nominal sets. *LMCS*, 10(3), 2014.
4. B. Bollig, P. Habermehl, M. Leucker, and B. Monmege. A robust class of data languages and an application to learning. *Logical Methods in Computer Science*, 10(4), 2014.
5. S. Cassel, F. Howar, B. Jonsson, and B. Steffen. Active learning for extended finite state machines. *Formal Asp. Comput.*, 28(2):233–263, 2016.
6. M. L. Furst, J. E. Hopcroft, and E. M. Luks. Polynomial-time algorithms for permutation groups. In *Proceedings of FOCS*, pages 36–41. IEEE Computer Society, 1980.
7. R. Grigore, D. Distefano, R. L. Petersen, and N. Tzevelekos. Runtime verification based on register automata. In *Proceedings of TACAS*, LNCS. Springer, 2013.
8. J. E. Hopcroft and R. M. Karp. A linear algorithm for testing equivalence of finite automata. Technical Report 114, Cornell University, 1971.
9. M. Kaminski and N. Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.
10. D. E. Knuth. Efficient representation of perm groups. *Combinatorica*, 11(1):33–43, 1991.
11. M. Leucker. Learning meets verification. In *Proceedings of FMCO*, volume 4709 of *Lecture Notes in Computer Science*, pages 127–151, 2007.
12. J. Moerman, M. Sammartino, A. Silva, B. Klin, and M. Szynwelski. Learning nominal automata. In *Proceedings of POPL*, pages 613–625. ACM, 2017.
13. A. S. Murawski, S. J. Ramsay, and N. Tzevelekos. Bisimilarity in fresh-register automata. In *Proceedings of LICS*, pages 156–167, 2015.
14. A. S. Murawski and N. Tzevelekos. Algorithmic nominal game semantics. In *Proceedings of ESOP*, volume 6602 of *Lecture Notes in Computer Science*, pages 419–438. Springer-Verlag, 2011.

15 F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

16 H. Sakamoto. *Studies on the Learnability of Formal Languages via Queries*. PhD thesis, Kyushu University, 1998.

17 H. Sakamoto and D. Ikeda. Intractability of decision problems for finite-memory automata. *Theor. Comput. Sci.*, 231(2):297–308, 2000.

18 T. Schwentick. Automata for XML - A survey. *J. Comput. Syst. Sci.*, 73(3):289–315, 2007.

19 N. Tzevelekos. Full abstraction for nominal general references. *Logical Methods in Computer Science*, 5(3), 2009.

20 N. Tzevelekos. Fresh-register automata. In *Proceedings of POPL*, pages 295–306. ACM Press, 2011.

21 F. W. Vaandrager. Model learning. *Commun. ACM*, 60(2):86–95, 2017.

On W[1]-Hardness as Evidence for Intractability

Ralph Christian Bottesch
University of Innsbruck, Innrain 52, 6020 Innsbruck, Austria
ralph.bottesch@uibk.ac.at

Abstract

The central conjecture of parameterized complexity states that FPT $\neq$ W[1], and is generally regarded as the parameterized counterpart to P $\neq$ NP. We revisit the issue of the plausibility of FPT $\neq$ W[1], focusing on two aspects: the difficulty of proving the conjecture (assuming it holds), and how the relation between the two classes might differ from the one between P and NP.

Regarding the first aspect, we give new evidence that separating FPT from W[1] would be considerably harder than doing the same for P and NP. Our main result regarding the relation between FPT and W[1] states that the closure of W[1] under relativization with FPT-oracles is precisely the class W[P], implying that either FPT is not low for W[1], or the W-Hierarchy collapses. This theorem also has consequences for the A-Hierarchy (a parameterized version of the Polynomial Hierarchy), namely that unless W[P] is a subset of some level A[t], there are structural differences between the A-Hierarchy and the Polynomial Hierarchy. We also prove that under the unlikely assumption that W[P] collapses to W[1] in a specific way, the collapse of any two consecutive levels of the A-Hierarchy implies the collapse of the entire hierarchy to a finite level; this extends a result of Chen, Flum, and Grohe (2005).

Finally, we give weak (oracle-based) evidence that the inclusion W[t] $\subseteq$ A[t] is strict for $t > 1$, and that the W-Hierarchy is proper. The latter result answers a question of Downey and Fellows (1993).

2012 ACM Subject Classification Theory of computation → Complexity classes

Keywords and phrases Parameterized complexity, Relativization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.73

Related Version A full version of the paper is available at `https://arxiv.org/abs/1712.05766`.

Funding This work was supported by the ERC Consolidator Grant QPROGRESS 615307 for the majority of its duration, and by the Austrian Science Fund (FWF) project Y757 at the time of publication.

Acknowledgements I thank Harry Buhrman, Sándor Kisfaludi-Bak, and Ronald de Wolf for helpful discussions. I am especially grateful to Ronald de Wolf and Leen Torenvliet for helpful comments on drafts of the paper.

1 Introduction

The central conjecture of parameterized complexity theory states that FPT $\neq$ W[1]. The complexity class FPT is a generalization of P, and it also contains this class in the sense that regardless of which parameter we associate with the instances of a problem in P, the resulting *parameterized problem* is in FPT. This inclusion is strict, as FPT also contains parameterized versions of problems that are provably not in P. The class W[1] can be regarded as a parameterized counterpart to NP. It can be defined in different ways, all

of them quite technical, but the most common definition is in terms of a parameterized version of a particular NP-complete problem (much like NP can be defined in terms of a Boolean circuit satisfiability problem). However, W[1] is not known or believed to contain *all* parameterized versions of problems in NP, and by defining complexity classes in terms of parameterizations of other NP-complete problems, one actually obtains a large set of seemingly distinct parameterized analogues of NP, some of which we list here:

$$\mathsf{A}[1] = \mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \mathsf{W}[3] \subseteq \ldots \mathsf{W}[t] \ldots \subseteq \mathsf{W}[\mathsf{P}] \subseteq \mathsf{para\text{-}NP}.$$

Among these, the most interesting classes are W[1] (a.k.a. A[1]) and W[P], due to having many natural complete problems.

The basic intuition for why W[1] (and hence all classes in the above sequence) should differ from FPT is the same as for P ≠ NP, namely that we do not know of any way to efficiently simulate nondeterministic computations deterministically. This intuition is often used to justify considering the W[1]-hardness of a problem as evidence for its intractability. But because FPT is strictly larger than P, while W[1] does not appear to capture all of the complexity of NP, it seems that proving the central conjecture of parameterized complexity theory may be harder than separating P and NP. We investigate qualitative differences between the two conjectures, as well as the more general question of whether FPT occupies the same place within W[1] as P does within NP. We start by giving a brief summary of some relevant prior results.

That the central parameterized conjecture is at least as strong as its classical counterpart is easy to prove: If NP = P, then, as noted above, every parameterized version of every problem in NP(= P) must be in FPT, hence W[1] = FPT (and, in fact, para-NP = W[P] = ... = FPT). Thus we have that FPT ≠ W[1] ⇒ P ≠ NP. The converse of this implication is not known to hold, but Downey and Fellows [9] were the first to observe that a collapse of W[1] to FPT would at least imply the existence algorithms with sub-exponential running time for the NP-complete problem 3Sat. This would contradict the *Exponential Time Hypothesis (ETH)*, first introduced by Imagliazzo, Paturi, and Zane [13], which states that for some constant $c > 0$, 3Sat can not be solved in time $O^*(2^{cn})$ by deterministic Turing machines (TMs). This conjecture has enjoyed much popularity recently, because, assuming ETH, for many problems it is possible to prove a complexity lower bound that matches that of the best known algorithm up to lower-order factors (see [14] for a survey of such results). Nevertheless, one should keep in mind that ETH is a much stronger statement than P ≠ NP, since it rules out not only the existence of polynomial-time algorithms for 3Sat, but also of those that run in up to exponential-time (for some bases). Putting all of these facts together, we have:

$$\text{ETH} \Longrightarrow \mathsf{FPT} \neq \mathsf{W}[1] \Longrightarrow \ldots \Longrightarrow \mathsf{FPT} \neq \mathsf{W}[\mathsf{P}] \Longrightarrow \mathsf{FPT} \neq \mathsf{para\text{-}NP} \Longrightarrow \mathsf{P} \neq \mathsf{NP}.$$

The above sequence relates parameterized complexity conjectures to two classical ones, but it does not say which of them are closer in strength to ETH and which are closer to P ≠ NP. The only known fact here is that FPT ≠ para-NP ⇔ P ≠ NP (see [11, Corollary 2.13]), but there is strong evidence suggesting that all of the other parameterized conjectures listed above are considerably stronger than P ≠ NP (although possibly still weaker than ETH[1]). First, Downey and Fellows [8] construct an oracle relative to which P and NP differ while W[P]

[1] There is, in fact, a subclass of W[1], called M[1], of which it is known that FPT ≠ M[1] is equivalent to ETH (see [10]). The similarities between M[1] and W[1] can be seen as a further indication that the conjecture FPT ≠ W[1] is nearly as strong as ETH, but, evidently, both FPT ≠ M[1] and M[1] ≠ W[1] are wide open conjectures.

collapses to FPT, so we know that any proof of the implication $\mathsf{P} \neq \mathsf{NP} \Rightarrow \mathsf{FPT} \neq \mathsf{W[P]}$ can not be as simple as the proof of the converse implication sketched above. More importantly, $\mathsf{FPT} \neq \mathsf{W[P]}$ can be related much more precisely to other classical complexity conjectures.

How strong the assumption $\mathsf{FPT} \neq \mathsf{W[P]}$ is, can be elegantly expressed in terms of *limited nondeterminism*. If f is a poly-time-computable function, denote by $\mathsf{NP}[f(n)]$ the class of problems that can be solved by a nondeterministic TM in polynomial-time by using at most $O(f(n))$ bits of nondeterminism (n denotes the size of the input). Note that $\mathsf{NP}[\log n] = \mathsf{P}$, since a deterministic TM can cycle through all possible certificates of length $O(\log n)$ in polynomial-time. A remarkable theorem of Cai, Chen, Downey, and Fellows [6] states that $\mathsf{FPT} \neq \mathsf{W[P]}$ holds *if and only if* for *every* poly-time-computable non-decreasing unbounded function h, we have that $\mathsf{P} \neq \mathsf{NP}[h(n) \log n]$ (see [11, Theorem 3.29] for a proof of the theorem in this form). The class of functions referred to in this theorem contains functions with very slow growth, such as the iterated logarithm function, $\log^*$. In fact, there is no poly-time-computable non-decreasing unbounded function that has the slowest growth, because if some function h satisfies these conditions, then so does $\log^* h$. It is not even intuitively clear whether P is different from NP when the amount of allowed nondeterminism is arbitrarily close to trivial. At the very least, the fact that an infinite number of increasingly strong separations must hold in order for W[P] to not collapse to FPT, suggests that separating these two classes is much farther out of our reach than a separation of P and NP.

The evidence we have seen so far indicates that proving a separation of W[1] and FPT may be harder than proving $\mathsf{P} \neq \mathsf{NP}$. But assuming that both conjectures hold, it is meaningful to ask whether the internal structure of W[1] resembles that of NP, and there is indeed some positive evidence in this direction. For example, a parameterized version of Cook's Theorem connects Boolean circuit satisfiability to W[1]-completeness (see [9]), a parameterized version of Ladner's Theorem states that if $\mathsf{FPT} \neq \mathsf{W[1]}$, then there is an infinite hierarchy of problems with different complexities within W[1] (see [9]), and the machine-based characterizations of this class, due to Chen, Flum, and Grohe [7], establish that W[1] can indeed be defined in terms of nondeterministic computing machines. Nevertheless, there are also previously unexplored ways in which W[1] may not behave the same way as NP.

Our main goal in this work is to provide further evidence that the classes FPT and W[1] are close not only in the sense of being difficult to separate, but also in the sense that the relationship between the two differs from that of P and NP, in a way that indicates that FPT is larger within W[1] than P is within NP (assuming the latter pair does not collapse). These results contrast with those in [4], where we showed how certain theorems about FPT and the levels of the A-Hierarchy can be proved in the same way as for their classical counterparts.

1.1 Summary of our results

The difficulty of separating W[P] from FPT. Assuming that we could prove a separation of the form $\mathsf{P} \neq \mathsf{NP}[h(n) \log n]$ for a particular, slow-growing function h, how much progress would we have made towards proving the separation where $h(n)$ is replaced by $\log h(n)$? Intuitively, the difficulty of proving non-equality should increase when a function with a slower growth is chosen. On the other hand, if $\mathsf{FPT} \neq \mathsf{W[P]}$ holds, then all such classical separations hold as well (by the above-mentioned theorem of Cai *et al.* [6]), and therefore any one of them implies the others. It is not clear, however, whether a proof of $\mathsf{FPT} \neq \mathsf{W[P]}$ with $\mathsf{P} \neq \mathsf{NP}[h(n) \log n]$ as a hypothesis would be significantly simpler than a proof from scratch. We show that this is unlikely to be the case, by proving (Theorem 9) that for any poly-time-computable non-decreasing unbounded function h, there exists a computable oracle O_h such that:

$$\mathsf{P}^{O_h} \neq \mathsf{NP}[h(n) \log n]^{O_h}, \text{ but } \mathsf{W[P]}^{O_h} = \mathsf{FPT}^{O_h}.$$

Theorem 9 is an improvement over the above-mentioned oracle construction of Downey and Fellows [8][2]. It is weak as a barrier result, since the relativization barrier has been repeatedly overcome in the last three decades, but nevertheless the theorem succinctly expresses how much harder the conjecture $\mathsf{FPT} \neq \mathsf{W}[\mathsf{P}]$ is compared to classical questions regarding nondeterministic vs. deterministic computation: No matter how small the amount of nondeterminism that provably yields a class strictly containing P, we will always be a non-trivial proof step away from separating $\mathsf{W}[\mathsf{P}]$ (or $\mathsf{W}[1]$) from FPT.

The structure of W[1] and its relation to FPT. The class $\mathsf{A}[1](=\mathsf{W}[1])$[3] can be characterized in terms of random access machines that perform *tail-nondeterministic* computations [7]. Such computations consist of two phases: 1. a (deterministic) FPT-computation; 2. a short nondeterministic computation that can use any data computed in phase 1. Tail-nondeterministic machines that perform only the second phase of the computation (without a longer deterministic computation preceding it), can not solve every problem in FPT, but, paradoxically, they can solve many problems that are complete for $\mathsf{A}[1]$ (we give an example in Section 4). As we will see, this simple observation has important consequences for the structure of this class.

A first consequence is that giving an $\mathsf{A}[1]$-machine very restricted oracle access to even a tractable (FPT) problem, may increase its computational power, because then the use of nondeterminism can be combined with the ability to solve instances of an FPT-problem via the oracle. Thus, FPT-computations appear to constitute a non-trivial computational resource for $\mathsf{A}[1]$ (unlike P-computations for NP). Somewhat suprisingly, we can actually identify the complexity class resulting from endowing $\mathsf{A}[1]$ with FPT-oracles, if a suitable, highly restricted type of oracle access is used. We have (Theorem 12, Corollary 13) that:

$$\mathsf{A}[1]^{\mathsf{FPT}} = \mathsf{W}[\mathsf{P}] \quad \text{and} \quad \forall t \geq 1 : \mathsf{W}[t]^{\mathsf{FPT}} = \mathsf{W}[\mathsf{P}],$$

where we used the common notation $\mathcal{C}_1^{\mathcal{C}_2} := \bigcup_{Q \in \mathcal{C}_2} \mathcal{C}_1^Q$. This means that either $\mathsf{W}[\mathsf{P}] = \mathsf{W}[1]$, in which case $\mathsf{W}[\mathsf{P}]$ is smaller than generally believed, or FPT is larger within $\mathsf{W}[1]$ than P is within NP.

Putting the known and new facts together, Theorem 9 and the result of Cai *et. al.* [6] mentioned in the introduction indicate that $\mathsf{W}[\mathsf{P}]$ is likely to be closer to FPT than any class $\mathsf{NP}[h(n) \log n]$ is to P (see Figure 1). The case for this figure being accurate is further strengthened by Theorem 12 and Corollary 13, which exhibit another way in which at least two of the classes FPT, $\mathsf{W}[1]$, and $\mathsf{W}[\mathsf{P}]$ are close.

Theorem 12 and the observation preceding it also have consequences for the A-Hierarchy, which is a parameterized analogue of PH. Although they share some essential properties [7, 4], a corollary of Theorem 12 is that, unless some unlikely inclusions between complexity classes occur, the two hierarchies have structural differences that indicate that consecutive levels of the A-Hierarchy are closer to each other than the corresponding levels of PH (see

[2] Actually, Downey and Fellows [8] use a different computational model to define and relativize $\mathsf{W}[\mathsf{P}]$, so the two results, although in the same spirit, may not be directly comparable at a technical level.

[3] $\mathsf{W}[1]$ and $\mathsf{A}[1]$ coincide as complexity classes, but in [7], Chen, Flum, and Grohe give two machine-based characterizations, one which can be generalized to get the levels of the W-Hierarchy, and one which generalizes to the levels of the A-Hierarchy. The machine model for $\mathsf{A}[1]$ is easier to handle when working with oracles, so we typically use this model when relativizing this class, and write "$\mathsf{A}[1]$" to emphasize this fact. However, oracle $\mathsf{W}[1]$-machines can also be defined so that our theorems hold for this model as well (see Section 3).

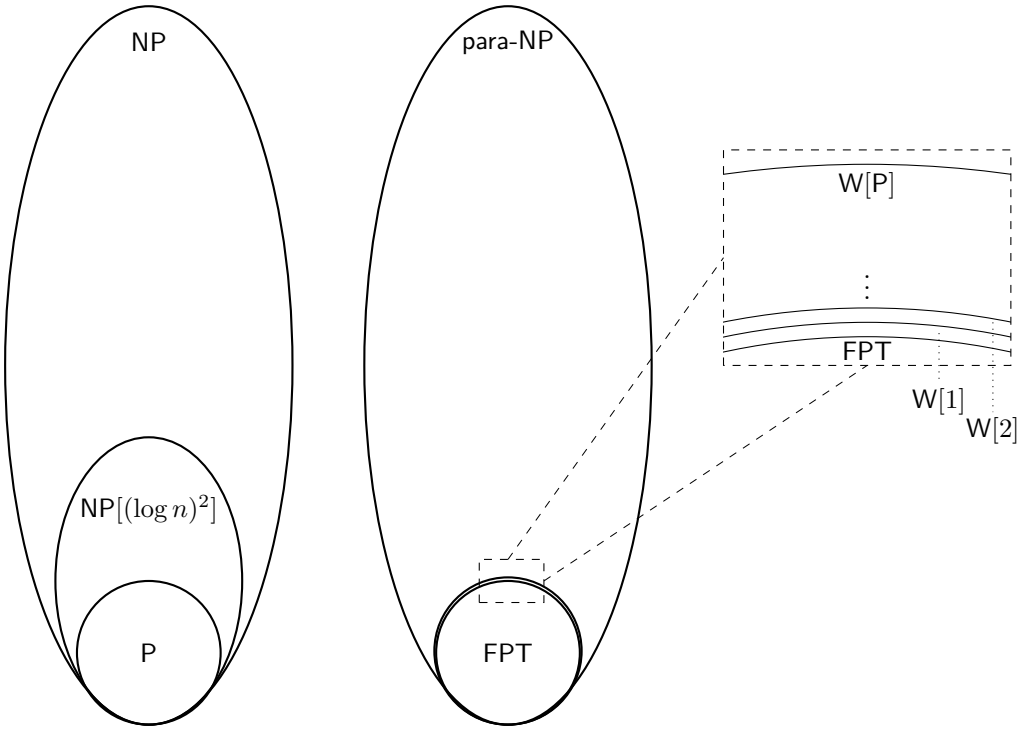


Figure 1 The mutual closeness of the parameterized complexity classes, compared to that of their classical analogues, as suggested by [6, 8], Theorems 9 and 12, and Corollary 13. Regardless of which class $\mathsf{NP}[h(n)\log n]$ we choose to represent between P and NP on the left side (whether it is $\mathsf{NP}[(\log n)^2]$ as in the picture, $\mathsf{NP}[\log^* n \log n]$, or something even smaller), it will be much larger compared to P than $\mathsf{W}[\mathsf{P}]$ is compared to FPT.

Section 4). Conversely, using a similar idea as in the proof of Theorem 12, we can show (Theorem 15) that if $\mathsf{W}[\mathsf{P}]$ were to collapse to $\mathsf{W}[1]$ in a specific way, we would get a *downward separation* theorem for the A-Hierarchy (i.e., that if two levels collapse, the entire hierarchy collapses to the smaller of the two). Proving such a theorem for the A-Hierarchy has been a long-standing open problem in parameterized complexity theory, and although our theorem falls short of this goal (since it requires an unlikely collapse to occur), it marks the first progress on this front in over a decade (since [7]).

Level-by-level relativized separations of the W- and the A-Hierarchy. We also give some evidence that certain collapses do not occur. The only relations that are known to hold between the classes $\mathsf{W}[t]$ and $\mathsf{A}[t]$ are that $\mathsf{W}[1] = \mathsf{A}[1]$ and that $\mathsf{W}[t] \subseteq \mathsf{A}[t]$ for $t \geq 2$. We show that in a relativized setting, the known inclusions can be made strict and some unexpected inclusions can be ruled out.

Separations of complexity classes relative to oracles count only as very weak evidence that the unrelativized versions of the classes are distinct, due to the fact that such oracles can in some cases be constructed even when two classes coincide (the most famous example being $\mathsf{IP} = \mathsf{PSPACE}$ [16] – see [12] for an oracle separating the two). Nevertheless, there are a few reasons why level-by-level relativized separations for the W- and the A-Hierarchy are interesting: First, since it is generally assumed that these hierarchies are proper and distinct, we should expect to have at least this weak form of evidence supporting the assumption.

Second, we have seen a number of results which suggest that the levels of these hierarchies are in various ways close to each other, so proving even relativized separations between them may be non-trivial. Finally, relativization in the parameterized setting is still mostly unexplored, and although the proofs of the following theorems rely on standard diagonalization arguments, the details of the machine models and how they are allowed to access oracles require special care in order to make the arguments work.

In Section 5 we show (Theorem 16, Corollary 17) that there exists a computable parameterized oracle O such that

$$\forall t \geq 2 : \mathsf{A}[t]^O \not\subset \mathsf{W}[t]^O.$$

Note that this is a *single* oracle relative to which all inclusions are simultaneously made strict. Also note that, although we use machine-based characterizations of classes $\mathsf{A}[t]$ and $\mathsf{W}[t]$ which result in distinct characterizations of the class $\mathsf{A}[1] = \mathsf{W}[1]$, fortunately, this oracle does not appear to separate $\mathsf{A}[1]$ from $\mathsf{W}[1]$. Such a separation would have suggested that the strict inclusions are mere artifacts of the machine models used.

Finally, we give evidence which suggests that the W-Hierarchy is not contained within any finite level of the A-Hierarchy (Theorem 18): For all $t \geq 1$, there exists a computable parameterized oracle O_t such that

$$\mathsf{W}[t+1]^{O_t} \not\subset \mathsf{A}[t]^{O_t}.$$

Since it holds that $\mathsf{W}[t]^{O_t} \subseteq \mathsf{A}[t]^{O_t}$, each oracle O_t separates two consecutive levels of the W-Hierarchy. This answers a question of Downey and Fellows [8], although we do not have a single oracle that simultaneously separates the entire hierarchy.

2 Preliminaries

We assume familiarity with standard facts and notations from both classical and parameterized complexity theory, and refer to [2] and to [11] for the necessary background in the respective branches. Since the characterizations of various parameterized complexity classes in terms of computing machines [7] are less well known, we give a brief overview of the main definitions.

Many parameterized complexity classes can only be naturally characterized in terms of *random access machines (RAMs)*, which can store entire integers in each of their registers, perform the operations addition, subtraction, and division by 2 on integers in unit time, and can access any part of their memory in constant time (see [15] or the introduction of [7]). The input of a RAM can be a sequence of non-negative integers, and we allow the instances of problems to be encoded in this way whenever we are working only with RAMs (as opposed to TMs). Since the size of a sequence of non-negative integers is calculated as the sum of the length of the binary representations of the individual numbers, RAMs have no significant computational advantage over TMs [15, Theorem 2.5]. However, this encoding does make a difference when considering oracle RAMs, because the query instances will also be encoded in this fashion.

We give two examples of definitions of complexity classes in terms of RAMs. It is not difficult to see that these are equivalent to the standard (TM-based) definitions (see [11]). Note that we use the Downey-Fellows definition of parameterized problems [9], where the parameter value, encoded in unary, is given together with the input.

▶ **Definition 1.** Let Q be a parameterized problem. We say that $Q \in \mathsf{FPT}$ if and only if there exists a RAM M, a computable function f, and a constant $c \geq 0$, such that for every input (x, k) with $x \in \mathbb{N}^*$ and $k \geq 0$, M runs in time $f(k)(|x| + k)^c$ and accepts if $(x, k) \in Q$, otherwise it rejects. The class $\mathsf{para\text{-}NP}$ is defined similarly, except with RAMs which can nondeterministically guess, in unit time, positive integers of size upper-bounded by the $f(k)(|x| + k)^c$ (the bound on the running time).

We also collect several useful definitions and notations in the following:

▶ **Definition 2.** Let $\mathcal{C}_1, \mathcal{C}_2$ be complexity classes, where $\mathcal{C}_1$ is defined in terms of computing machines that can be given access to an oracle, and let $P_0, P_1 \subseteq \{0,1\}^*$ be classical languages. We define: $\mathcal{C}_1^{\mathcal{C}_2} := \bigcup_{P \in \mathcal{C}_2} \mathcal{C}_1^P$ and $P_0 \oplus P_1 := \{0x \mid x \in P_0\} \cup \{1x \mid x \in P_1\}$. We say that P_0 is *low for* $\mathcal{C}_1$ if $\mathcal{C}_1^{P_0} = \mathcal{C}_1$, and we say that $\mathcal{C}_2$ is *low for* $\mathcal{C}_1$ if $\mathcal{C}_1^{\mathcal{C}_2} = \mathcal{C}_1$.

2.1 The A-Hierarchy and the W-Hierarchy

The following classes are defined in terms of *alternating random access machines (ARAMs)*, which are RAMs that can nondeterministically guess, in unit time, integers of size bounded by the running time of the machine on a given input, either in the existential or the universal mode (see [7]).

▶ **Definition 3** ([7])**.** For each $t \geq 1$, let $\mathsf{A}[t]$ be the class of parameterized problems that are solved by some ARAM A which, for some computable functions f and h, and a constant $c \geq 0$, satisfies the following conditions on every input (x, k):

1. A runs in time at most $f(k)(|x| + k)^c$;
2. throughout the computation, the values in A's registers do not exceed $f(k)(|x| + k)^c$;
3. all nondeterministic guesses are made during the last $h(k)$ steps of the computation;
4. the first nondeterministic guess is existential and the machine alternates at most $t - 1$ times between existential and universal guesses.

The class $\mathsf{co\text{-}A}[1]$ is defined in terms of ARAMs which satisfy conditions 1–3, but only make universal nondeterministic guesses (one can verify, just as in the classical setting, that a problem is in $\mathsf{co\text{-}A}[1]$ if and only if it is the complement of a problem in $\mathsf{A}[1]$). ARAMs satisfying conditions 1 and 2 are called *parameter-bounded* in [4], those satisfying conditions 3 and 4 are called, respectively, *tail-nondeterministic* and *t-alternating* [7].

The classes $\mathsf{W}[t]$ ($t \geq 1$) can be defined in terms of $\mathsf{A}[t]$-machines (parameter-bounded tail-nondeterministic t-alternating ARAMs) that are further restricted so that: 1. Every block of nondeterministic guess instructions of the same kind, except the first one, is made up of at most c' guess instructions, where c' is a constant that is independent of the input. 2. All nondeterministically guessed integers are placed in a special set of *guess registers*, which can not be read from directly, and can only be accessed via special instructions that use the guessed values as indices for accessing standard registers. We will not need further details regarding these machines, and therefore refer the reader to [7] or [5] for more complete definitions. We will, however, define oracle $\mathsf{W}[t]$-machines (Definition 10).

▶ **Definition 4** ([4])**.** An *oracle (A)RAM* is a machine with an additional set of registers called *oracle registers*, instructions that allow the machine to copy values from its standard registers to the oracle registers, as well as a QUERY instruction, the execution of which results in one of the values 1 or 0 being placed into the first standard register of the machine, depending on whether the instance encoded in the oracle registers at that time constitute a 'yes'- or a 'no'-instance of a problem for which the machine is said to have an oracle.

An oracle (A)RAM has *balanced oracle access* to a parameterized oracle, if there is a computable function g such that on every input (x, k), the machine queries the oracle only with instances whose parameter value is $\leq g(k)$ (in other words, the parameter values of the instances for which the oracle is called should be upper-bounded by some function of k, but may not depend on n, even though the machine may have time to construct such a query instance). An oracle (A)RAM has *tail-restricted oracle access*, if its access to the oracle is balanced and, furthermore, there is a computable function h such that the machine makes oracle queries only within the last $h(k)$ steps of the computation on input (x, k). Note that tail-restricted access is also balanced.

For a parameterized complexity class $\mathcal{C}$ that is defined in terms of (A)RAMs, we write $\mathcal{C}(O)$ if $\mathcal{C}$ has unrestricted access to the oracle O, $\mathcal{C}(O)_{bal}$ if it has balanced access, and $\mathcal{C}(O)_{tail}$ if it has tail-restricted access. If $\mathcal{C}$ is defined in terms of tail-nondeterministic ARAMs, we also write $\mathcal{C}^O$ instead of $\mathcal{C}(O)_{tail}$ (so $\mathsf{A}[1]^O$ means $\mathsf{A}[1]$ with tail-restricted access to O). Note that for $\mathcal{C}(O)$, the oracle can be either classical or parameterized, but for balanced or more restricted oracle access, it must be parameterized.

2.2 W[P] and the W[P]-Hierarchy

We define W[P] both in terms of TMs and in terms of RAMs, and use both definitions at different points in the paper.

▶ **Definition 5** ([7]). Let Q be a parameterized problem. We say that $Q \in \mathsf{W}[\mathsf{P}]$ if and only if there exists a nondeterministic TM M, computable functions f and h, and a constant $c \geq 0$, such that for any input (x, k) with $x \in \{0, 1, \#\}^*$ and $k \geq 0$, M runs in time $f(k)(|x| + k)^c$, uses at most $h(k)\lceil \log(|x| + k) \rceil$ nondeterministic bits, and accepts if and only if $(x, k) \in Q$.

The following problem is complete for W[P] under fpt-reductions.

p-WSatCircuit	
Input:	A Boolean circuit C with n input bits, $k \in \mathbb{N}$.
Parameter:	k
Problem:	Decide whether C has a satisfying assignment of weight k.

The class W[P] can also be defined in terms of RAMs [7]. One can also define a hierarchy that is similar to PH, except in terms of alternating nondeterminism that matches the nondeterminism of W[P].

▶ **Definition 6** ([4]). For each $t \geq 1$, let $\Sigma_t^{[P]}$ be the class of parameterized problems that are solved by some ARAM A which, for some computable functions f and h, and a constant $c \geq 0$, satisfies, on every input (x, k), conditions 1, 2, and 4 from Definition 3, as well as:

3'. *A* nondeterministically guesses at most $h(k)$ numbers throughout the computation.

We denote the class $\bigcup_{t=1}^{\infty} \Sigma_t^{[P]}$ by W[P]H, the W[P]-*Hierarchy*.

It is not difficult to see that $\mathsf{W}[\mathsf{P}] = \Sigma_1^{[P]}$. Note that we will use the term "W[P]-machine" to designate both the TMs from Definition 5 as well as the $\Sigma_1^{[P]}$-machines from Definition 6 (which are RAMs), but it should be clear from the context which type of machine is meant.

For each $t \geq 1$, the following generalizations of the problem p-WSatCircuit can easily be seen to be, respectively, complete problems for $\Sigma_t^{[P]}$.

p-AWSatCircuit$_t$	
Input:	A Boolean circuit C with n input bits, $k \in \mathbb{N}$, and a partition of the input variables of C into t sets $I_1, \ldots, I_t$.
Parameter:	k
Problem:	Decide whether there exists a set $J_1 \subseteq I_1$ of size k such that for all subsets $J_2 \subseteq I_2$ of size k there exists ... such that setting precisely the variables in $J_1 \cup \ldots \cup J_t$ to 'true' results in a satisfying assignment of C.

As in the case of the Polynomial Hierarchy, and unlike the case of the A-Hierarchy, it is known that the collapse of levels of the W[P]-Hierarchy would propagate upward:

▶ **Fact 7** (Corollary 17 of [4]). *If for any $t \geq 1$, $\Sigma_{t+1}^{[P]} = \Sigma_t^{[P]}$, then* $\mathsf{W[P]H} = \Sigma_t^{[P]}$.

▶ **Definition 8** ([4]). An oracle ARAM has *parameter-bounded oracle access* to a parameterized oracle, if its access to the oracle is balanced and, furthermore, there is a computable function g such that on every input (x, k), the machine makes at most $g(k)$ oracle queries.

If $\mathcal{C}$ is a class that is defined in terms of ARAMs, we write $\mathcal{C}(O)_{para}$ to denote that $\mathcal{C}$ has parameter-bounded access to the oracle O. If $\mathcal{C} = \Sigma_t^{[P]}$, for some $t \geq 1$, we may also write $\mathcal{C}^O$ to mean $\mathcal{C}(O)_{para}$ (so $\mathsf{W[P]}^O = \mathsf{W[P]}(O)_{para}$).

3 The difficulty of separating W[P] from FPT

In this section we show that there is likely no shortcut to proving $\mathsf{FPT} \neq \mathsf{W[P]}$ via any finite number of separations of the form $\mathsf{P} \neq \mathsf{NP}[h(n) \log n]$. Due to space restrictions, the proofs of the theorems in this section can be found in the full version of the paper.

To prove the theorem, we need to construct an oracle relative to which two conditions hold simultaneously: the collapse of one pair of complexity classes and the separation of another. One approach to achieving this is to construct the oracle in stages, and to work towards one goal in the odd-numbered stages and towards the other in the even-numbered ones, while ensuring that the two constructions do not interfere with each other (see [3, Theorem 5.1] for one example of an application of this technique). However, this approach does not always work, and in this case it fails because one pair of classes is parameterized (specifically, it is not possible to computably list all FPT- or W[P]-machines, but this appears to be necessary in this type of staged construction). To overcome this obstacle, we use an idea of Allender [1], who constructs an oracle with two parts: the first part is designed so as to ensure that one pair of classes collapses *regardless of what the second part of the oracle is*; the second part can then be freely used in a diagonalization argument to separate the remaining pair of classes.

▶ **Theorem 9.** *For every polynomial-time-computable non-decreasing unbounded function h, there exists a computable oracle B such that*

$$\mathsf{P}^B \neq \mathsf{NP}[h(n) \log n]^B, \quad \textit{but} \quad \mathsf{W[P]}(B) = \mathsf{FPT}(B).$$

For this result we have used unrestricted oracle access to relativize the parameterized complexity classes, rather than the parameter-bounded type that we argued is natural for W[P] [4]. This is because restricting the oracle access to being balanced (or more) makes it possible to collapse even para-NP to FPT, with the classical separation unchanged. Thus we would get an oracle relative to which NP and P differ while para-NP and FPT coincide, which is clearly an artifact of the restrictions placed on the oracle access, since we know that, unrelativized, a collapse of para-NP to FPT is equivalent to a collapse of NP to P (see [11]).

It seems reasonable to expect that if W[P] collapses to FPT relative to some oracle, then so should any class W[t]. But to show that this is indeed the case, we first need to define oracle W[t]-machines. Recall that a W[t]-machine is similar to an A[t]-machine, but the numbers it guesses nondeterministically are placed in a special set of *guess registers*, to which the machine has only limited access [7] (see also [5]). Naturally, an oracle W[t]-machine should then have three sets of registers: the standard registers, guess registers (which the machine can not read from directly), and oracle registers. For such machines, the usual way to read from or write to the oracle registers is very limiting, because the machines' nondeterminism would only weakly be able to influence the query instances. For example, nondeterministically guessed numbers could not be written to the oracle registers. The interaction between the nondeterminism of such machines and their ability to form query instances can be strengthened without allowing the W[t]-specific restrictions to be circumvented. We achieve this by making the oracle registers *write-only*, and adding instructions that allow the machine to copy values from the guess registers to the oracle registers and to use numbers from the guess registers to address oracle registers. In this way, the machine can still not read the guessed numbers directly or use them in arithmetic computations, but can nevertheless use them for oracle queries. In many cases, this allows oracle W[t]-machines to match A[t]-machines in the way the oracle is used.

▶ **Definition 10.** An *oracle* W[t]*-machine* is a W[t]-machine that, in addition to the standard registers $r_0, r_1, \ldots$, and guess registers $g_0, g_1, \ldots$ (to which the machine has only restricted access), also possesses a set of oracle registers $o_0, o_1, \ldots$. The contents of oracle registers are never read from and are only affected by the following new instructions:

SO_MOVE - copy the contents of standard register r_0 to oracle register o_{r_1};
GO_MOVE - copy the contents of guess register g_{r_0} to oracle register o_{r_1};
ADDR_GO_MOVE - copy the contents of register r_0 to $o_{g_{r_1}}$;
OO_MOVE - copy the contents of o_{r_0} to o_{r_1}.

Additionally, the machine has a QUERY instruction that places either the value 0 or 1 into r_0, depending on whether the contents of the oracle registers at the time when the instruction is executed represent a 'no'- or a 'yes'-instance of the problem to which the machine has oracle access.

Note that for such machines, it again makes sense to speak of unrestricted, balanced, parameter-bounded, or tail-restricted oracle access. With the above definition in mind, we can now prove Corollary 11, where oracle access is unrestricted.

▶ **Corollary 11.** *For any function h as in Theorem 9, and the corresponding oracle B, we have that* $\mathsf{FPT}(B) = \mathsf{W}[1](B) = \mathsf{A}[1](B) = \mathsf{W}[2](B) = \mathsf{A}[2](B) = \ldots = \mathsf{W}[\mathsf{P}](B)$.

4 The structure of W[1] and its relation to FPT

Under the assumption that $\mathsf{P} \neq \mathsf{NP}$, it is meaningful to ask whether the relation between the two classes is the same as the one between FPT and W[1]. So far, we have seen evidence that the two parameterized classes are closer to each other in the sense that proving a separation between them is more difficult than proving $\mathsf{P} \neq \mathsf{NP}$. In this section we look at other ways in which W[1] is closer to FPT than NP is to P.

In this section, the definitions of classes in terms of RAMs are used, instances of problems and oracles query instances are encoded as integer sequences, and oracles are parameterized.

Is FPT *low for* W[1]*?*

Given that FPT is the class of tractable problems in parameterized complexity, and that P-oracles add no computational power to NP (or to any class $\mathsf{NP}[h(n)\log n]$), one might expect FPT to also be low for W[1]. It turns out, however, that allowing tail-nondeterministic machines to make even tail-restricted queries to an FPT-oracle can increase their computational strength to that of W[P]. We prove this for A[1] first, since this machine model is more easily relativizable.

▶ **Theorem 12.** $\mathsf{A}[1]^{\mathsf{FPT}} = \mathsf{W}[\mathsf{P}]$. *Therefore,* FPT *is low for* A[1] *if and only if* $\mathsf{W}[\mathsf{P}] = \mathsf{A}[1]$ *and the* W*-Hierarchy collapses to its first level.*

Proof. We have that $\mathsf{A}[1]^{\mathsf{FPT}} \subseteq \mathsf{A}[1](\mathsf{FPT})_{bal} \subseteq \mathsf{W}[\mathsf{P}](\mathsf{FPT})_{bal} = \mathsf{W}[\mathsf{P}]$, with the final equality holding because a W[P]-machine can replace balanced oracle calls to FPT-problems by fpt-length computations.

To show that $\mathsf{W}[\mathsf{P}] \subseteq \mathsf{A}[1]^{\mathsf{FPT}}$, we define the following problem:

p-WSatCircuit-with-assignment	
Input:	A circuit C with n inputs, $k \in \mathbb{N}$, and vector $v \in \{0,1\}^n$ of weight k.
Parameter:	k.
Problem:	Decide whether v is a satisfying assignment for C.

Since the output of a circuit can be computed in time polynomial in its size, the above problem is obviously in FPT[4]. Any problem $Q \in \mathsf{W}[\mathsf{P}]$ can be solved by some A[1]-machine A with tail-restricted access to p-WSatCircuit-with-assignment as an oracle: First, A reduces in fpt-time the input instance (x,k) to an instance (y,k') of p-WSatCircuit, where k' depends computably only on k. Let m be the number of input bits of the circuit encoded in y. If $m < k'$, A rejects, otherwise it writes y, 0^m, and $1^{k'}$ to its oracle registers, thus forming a valid instance of p-WSatCircuit-with-assignment, except that the assignment vector has weight 0. Now A enters the nondeterministic phase of its computation by guessing k' pairwise distinct integers $i_1, \ldots, i_{k'} \in [m]$. It then modifies the assignment vector in the oracle registers by changing the zeroes at positions $i_1, \ldots, i_{k'}$ of the vector 0^m to 1, queries the oracle, and accepts if the answer is 'yes', otherwise it rejects.

It is easy to see that what this machine actually does is nondeterministically guess a satisfying assignment of the p-WSatCircuit-instance, if one exists, and delegate the verification to the oracle. The trick here is that the all-zero assignment vector must be written to the oracle registers deterministically, because during the nondeterministic phase at the end of the computation there may not be enough time to do so. Then the machine only needs to change the vector at k' positions to obtain an assignment with the right weight, which takes only $O(k')$ steps with random access memory. ◀

Note that the proof that a W[P]-machine can simulate A[1]-machines with FPT-oracles only works if the oracle access of the A[1]-machines is tail-restricted or at least parameter-restricted. On the other hand, the proof that $\mathsf{A}[1]^{\mathsf{FPT}} \supseteq \mathsf{W}[\mathsf{P}]$ only requires tail-restricted oracle access. We regard this as further evidence (in addition to the results from [4]) that tail-restricted oracle access is the natural type to consider for the class A[1].

[4] In fact, this problem is clearly in P, meaning that we can actually prove the stronger statement $\mathsf{W}[\mathsf{P}] \subseteq \mathsf{A}[1]^{\mathsf{P}}$. However, we choose FPT instead of P because the instance with which the oracle is queried will be fpt-sized, and because it is more natural to have A[1]-machines query a parameterized oracle, rather than a classical one.

Since $\mathsf{A}[1] \subseteq \mathsf{W}[t] \subseteq \mathsf{W}[\mathsf{P}]$ holds for all $t \geq 1$, and by Theorem 12 we have that $\mathsf{A}[1]^{\mathsf{FPT}} = \mathsf{W}[\mathsf{P}] = \mathsf{W}[\mathsf{P}](\mathsf{FPT})_{tail}$, it seems reasonable to expect that $\mathsf{W}[t]^{\mathsf{FPT}} = \mathsf{W}[\mathsf{P}]$ holds for all t as well. In order to prove that this is indeed the case, we need to use oracle $\mathsf{W}[t]$-machines (Definition 10), with tail-restricted access to the oracle. Then the proof is based on a combination of ideas from the proofs of Corollary 11 and Theorem 12.

▶ **Corollary 13.** *For every $t \geq 1$ it holds that $\mathsf{W}[t]^{\mathsf{FPT}} = \mathsf{W}[\mathsf{P}]$.*

In [4] we showed that for every $t \geq 1$ the class $\mathsf{A}[t+1]$ can be obtained as $\mathsf{A}[1]^{O_t}$, where O_t is a specific $\mathsf{A}[t]$-complete oracle, but we also observed that $\mathsf{A}[1]^{\mathsf{FPT}}$ does not appear to be a subset of $\mathsf{A}[t]$ for any t. Theorem 12 provides support for this intuition by identifying $\mathsf{A}[1]^{\mathsf{FPT}}$ as a class which is not known or believed to be a subset of any class $\mathsf{A}[t]$.

▶ **Corollary 14.** *For every $t \geq 1$ we have that if $\mathsf{A}[t+1] = \mathsf{A}[1]^{\mathsf{A}[t]}$, then $\mathsf{W}[\mathsf{P}] \subset \mathsf{A}[t+1]$. In particular, if $\mathsf{W}[\mathsf{P}] \not\subset \mathsf{A}[2]$, we have that $\mathsf{A}[1]^{\mathsf{A}[1]} \neq \mathsf{A}[2]$.*

Corollary 14 shows that the above-mentioned oracle characterization of the A-Hierarchy from [4] can probably not be improved significantly: Although it may be possible to obtain $\mathsf{A}[t+1]$ by providing $\mathsf{A}[1]$ with other $\mathsf{A}[t]$-complete oracles, it is unlikely that O_t can be replaced by the entire class $\mathsf{A}[t]$, for the somewhat counter-intuitive reason that $\mathsf{A}[t]$ contains all *tractable* problems. More importantly, Corollary 14 implies, assuming $\mathsf{W}[\mathsf{P}] \not\subset \mathsf{A}[t+1]$ and that PH is proper, that each class $\mathsf{A}[t+1]$ is closer to the class $\mathsf{A}[t]$ than Σ_{t+1}^P is to Σ_t^P, in the precise sense that $\mathsf{A}[t+1] \subsetneq \mathsf{A}[1]^{\mathsf{A}[t]}$, whereas $\Sigma_{t+1}^P = \mathsf{NP}^{\Sigma_t^P}$. The use of a highly restricted type of oracle access for the parameterized classes can only make this conclusion more legitimate.

A weak downward separation theorem for the A-Hierarchy.

It is a long-standing open problem whether the collapse of any class $\mathsf{A}[t+1]$ to $\mathsf{A}[t]$ would cause all higher levels of the A-Hierarchy to coincide with $\mathsf{A}[t]$ (or, equivalently, whether a separation of two classes $\mathsf{A}[t+1]$ and $\mathsf{A}[t]$ would imply that all levels below $\mathsf{A}[t]$ are distinct, whence the name "downward separation"). Given the similarities with PH, one might expect such a theorem to hold for the A-Hierarchy as well. Nevertheless, the proof of the downward separation theorem for the Polynomial Hierarchy does not appear to carry over directly to the parameterized setting. So far, the best result in this direction has been a theorem of Chen *et al.* [7], who showed that $\mathsf{W}[\mathsf{P}] = \mathsf{FPT}$ implies $\mathsf{FPT} = \mathsf{A}[1] = \mathsf{A}[2] = \ldots$. This result is already non-trivial, since $\mathsf{A}[t]$ is not known to be a subset of $\mathsf{W}[\mathsf{P}]$ for $t > 1$, and can be viewed as a parameterized version of $\mathsf{P} = \mathsf{NP} \Rightarrow \mathsf{PH} = \mathsf{P}$, except that the stronger collapse $\mathsf{W}[\mathsf{P}] = \mathsf{FPT}$ is required instead of $\mathsf{A}[1] = \mathsf{FPT}$ (in fact, the proof of the parameterized theorem in [7] is adapted from the proof of the corresponding classical theorem). Previously it was not known whether assuming a weaker collapse, for example $\mathsf{W}[\mathsf{P}] = \mathsf{A}[1]$, might also suffice to prove that $\forall t \geq 1 : \mathsf{A}[t] = \mathsf{A}[t+1] \Rightarrow \text{A-Hierarchy} = \mathsf{A}[t]$. In what follows we prove such a theorem.

Let $\mathsf{A}[1]_c$ be the class of parameterized problems Q such that there exists an $\mathsf{A}[1]$-machine that solves any instance (x, k) of Q in a number of steps depending only on k. This subclass of $\mathsf{A}[1]$ contains the problems that can be solved by $\mathsf{A}[1]$-machines *without* the need for a precomputation that runs in fpt-time. It is provably not closed under fpt-reductions, but contains many important $\mathsf{W}[1]$-complete problems, provided that the input is given in an appropriate format. For example, p-IndependentSet $\in \mathsf{A}[1]_c$, if the input graph is given in the form of an adjacency matrix, because then an $\mathsf{A}[1]$-machine can first guess k vertices (recall that a nondeterministic RAM can guess an integer between 1 and n in a single step; see Section 2.1) and use its random access memory to verify in $O(k^2)$ steps that none of the edges between two guessed vertices are in the graph. One can similarly show that p-ShortTMAcceptance and other $\mathsf{W}[1]$-complete problems are in $\mathsf{A}[1]_c$.

If W[P] were to collapse to A[1], then the W[P]-complete problem p-WSatCircuit would also be A[1]-complete, and therefore it would seem reasonable to expect that it is also in $\mathsf{A}[1]_c$, given an appropriate, efficiently computable encoding of the input. Thus, p-WSatCircuit $\in \mathsf{A}[1]_c$ seems only slightly less likely than $\mathsf{W}[\mathsf{P}] = \mathsf{A}[1]$ (although, strictly speaking, both p-WSatCircuit $\in \mathsf{A}[1]_c$ and p-WSatCircuit $\in \mathsf{FPT}$ (used by Chen *et al.* [7]) are strictly stronger assumptions than p-WSatCircuit $\in \mathsf{A}[1]$, and probably mutually incomparable). Under this assumption, we can prove the following:

▶ **Theorem 15.** *Assume that p-WSatCircuit $\in \mathsf{A}[1]_c$, meaning that there exists an* A[1]*-machine which solves any instance (x, k) of p-WSatCircuit in a number of steps depending computably on k alone. Then $\forall t \geq 1$ we have that $\mathsf{A}[t] = \mathsf{A}[t+1] \Rightarrow (\forall u \geq 1 : \mathsf{A}[t] = \mathsf{A}[t+u])$.*

Proof. We show that under the first assumption in the theorem statement, we have for every $t \geq 1$ that $\mathsf{A}[t+1] = \Sigma_{t+1}^{[P]}$. Since we already have a downward separation theorem for W[P]H (Fact 7), it follows that the desired conclusion holds for the A-Hierarchy.

First, we have for every $t \geq 1$ that $\Sigma_{t+1}^{[P]} \subseteq \mathsf{A}[t]^{p\text{-WSatCircuit}}$, by a similar proof as that of Theorem 12: To solve a problem $Q \in \Sigma_{t+1}^{[P]}$, an $\mathsf{A}[t]^{p\text{-WSatCircuit}}$-machine will first compute a reduction to the canonical $\Sigma_{t+1}^{[P]}$-complete problem p-AWSatCircuit$_{t+1}$, and, if t is odd, modify the resulting circuit so that its output is flipped. The machine then uses its t-alternating nondeterminism to guess the variables to set to 1 in the first t sets of the partition of the circuit's inputs, and hardwires this partial assignment into the circuit. The result is an instance of p-WSatCircuit, which can be solved with a single query to the oracle, and the oracle A[t]-machine now outputs the oracle's answer if t is odd, otherwise it outputs the opposite answer. It is easy to verify that this solves the problem Q.

Finally, we outline the proof that $\mathsf{A}[t]^{p\text{-WSatCircuit}} \subseteq \mathsf{A}[t+1]$, under the assumption that the algorithm for p-WSatCircuit mentioned in the theorem statement exists. This inclusion is proved in the same manner as $\mathsf{A}[1]^{p\text{-MC}(\Sigma_t[3])} \subseteq \mathsf{A}[t+1]$ [4, Theorem 13], which is itself a parameterized version of the proof of the well-known fact that $\mathsf{NP}^{\Sigma_t\mathrm{SAT}} \subseteq \Sigma_{t+1}^P$ (see [2, Section 5.5]). An $\mathsf{A}[t+1]$-machine can first perform the deterministic part of the oracle A[t]-machine's computation, and then use its $(t+1)$-alternating nondeterminism to guess the answers to the subsequent oracle queries of the simulated machine (existentially), all of its t-alternating nondeterministic guesses, as well as (suitably quantified) witnesses for the query instances. Oracle queries are then replaced by computations in which the guessed witnesses are used instead of nondeterministic guesses. The fact that evaluations of p-WSatCircuit-queries can be performed in this manner, is due to the assumption that this problem has a nondeterministic algorithm running in time dependent on k alone.

Since $\mathsf{A}[t+1] \subseteq \Sigma_{t+1}^{[P]}$ holds unconditionally, we conclude that the two classes are equal, which completes the proof. ◀

5 Level-by-level relativized separations of the W- and the A-Hierarchy

In this section we give oracle-based evidence that the main parameterized hierarchies do not collapse in unforeseen ways. We start by constructing a single oracle relative to which the inclusion of every W[t] in A[t] is strict, except for the first level. In fact, we accomplish this by proving the strongest possible relativized separation between co-nondeterminism and (existential) nondeterminism in the parameterized setting: the weakest co-nondeterministic class with tail-restricted oracle access, against the strongest nondeterministic class with unrestricted oracle access.

The proofs of the theorems in this section are based on standard diagonalization arguments that have been adapted to the parameterized setting, and can be found in the full version of the paper.

▶ **Theorem 16.** *There exists a computable oracle O such that* $\mathsf{co\text{-}A}[1]^O \not\subset \mathsf{para\text{-}NP}(O)$.

Since $\mathsf{co\text{-}A}[1]^O \subseteq \mathsf{A}[t]$ for all $t \geq 2$, and $\mathsf{W}[t]^O \subseteq \mathsf{para\text{-}NP}(O)$ for all $t \geq 1$, we immediately get the next corollary. Note, however, that the oracle constructed here does not appear to separate $\mathsf{A}[1]$ from $\mathsf{W}[1]$, since the separating problem in $\mathsf{co\text{-}A}[1]^O \setminus \mathsf{para\text{-}NP}(O)$ is not in $\mathsf{A}[1]^O$. Had a separation of two coinciding classes occured, this would have made the conclusion of Theorem 16 much less convincing.

▶ **Corollary 17.** *There exists a computable oracle O such that for every $t \geq 2$,* $\mathsf{W}[t]^O \subsetneq \mathsf{A}[t]^O$.

Finally, we show that the W-Hierarchy is not likely to be contained in any finite level of the A-Hierarchy.

▶ **Theorem 18.** *There exists for each $t \geq 1$ a computable oracle O_t such that* $\mathsf{W}[t+1]^{O_t} \not\subset \mathsf{A}[t]^{O_t}$*, where both machines have tail-restricted access to O_t, but the* $\mathsf{W}[t]$*-machine has the stronger type of oracle access mentioned in Section 3.*

As mentioned in the introduction, each oracle O_t also separates the classes $\mathsf{W}[t]$ and $\mathsf{W}[t+1]$ in the relativized setting, since $\mathsf{W}[t]^{O_t} \subseteq \mathsf{A}[t]^{O_t}$ but $\mathsf{W}[t+1]^{O_t} \not\subset \mathsf{A}[t]^{O_t}$.

6 Conclusions

Our results, together with the previously known theorems mentioned in the introduction, strongly indicate that if the central conjecture of parameterized complexity theory holds at all, proving it may be hard *even under the additional assumption of a separation between arbitrarily-weakly-nondeterministic polynomial-time and* P (and, in particular, that $\mathsf{P} \neq \mathsf{NP}$). Of course, the same also applies to the nowadays "standard" conjecture ETH. Additionally, we have seen that $\mathsf{W}[1]$ and FPT are in some ways unexpectedly close, unless much of what is generally assumed in parameterized complexity theory (such as the W-Hierarchy not collapsing) is false. All of this suggests that the hardness of a problem for up to $\mathsf{W}[\mathsf{P}]$ should not be treated as strong evidence of intractability, at least not with a similar level of confidence as when NP-hardness is considered evidence of computational intractability.

References

1. E. Allender. Limitations of the upward separation technique. *Mathematical Systems Theory*, 24(1):53–67, 1991.
2. S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge, 2009.
3. R.V. Book, C.B. Wilson, and M. Xu. Relativizing time, space, and time-space. *SIAM J. Comput.*, 11(3):571–581, 1982.
4. R.C. Bottesch. Relativization and Interactive Proof Systems in Parameterized Complexity Theory. In *12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*, volume 89, pages 9:1–9:12, 2018. URL: `http://drops.dagstuhl.de/opus/volltexte/2018/8571`.
5. J.F. Buss and T. Islam. Simplifying the Weft hierarchy. *Theoretical Computer Science*, 351(3):303–313, 2006.
6. L. Cai, J. Chen, R.G. Downey, and M.R. Fellows. On the structure of parameterized problems in NP. *Information and Computation*, 123:38–49, 1995.

7 Y. Chen, J. Flum, and M. Grohe. Machine-based methods in parameterized complexity theory. *Theoretical Computer Science*, 339:167–199, 2005.

8 R.G. Downey and M.R. Fellows. Fixed-parameter tractability and completeness III - Some structural aspects of the W hierarchy. In K. Ambos-Spies, S. Homer, and U. Schoning, editors, *Complexity Theory*, pages 166–191. Cambridge University Press, 1993.

9 R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer, Berlin, 1999.

10 R.G. Downey and M.R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

11 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, Berlin, 2006.

12 L. Fortnow and M. Sipser. Are there interactive protocols for co-NP languages? *Information Processing Letters*, 28(5):249–251, 1988.

13 R. Imagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

14 D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–71, 2011.

15 C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

16 A. Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

A Simple Augmentation Method for Matchings with Applications to Streaming Algorithms

Christian Konrad
Department of Computer Science, University of Bristol
Merchant Venturers Building, Woodland Road, BS8 1UB, United Kingdom
christian.konrad@bristol.ac.uk

Abstract

Given a graph G, it is well known that any maximal matching M in G is at least half the size of a maximum matching M^*. In this paper, we show that if G is bipartite, then running the Greedy matching algorithm on a sampled subgraph of G produces enough additional edges that can be used to augment M such that the resulting matching is of size at least $(2-\sqrt{2})|M^*| \approx 0.5857|M^*|$ (ignoring lower order terms) with high probability.

The main applications of our method lie in the area of data streaming algorithms, where an algorithm performs few passes over the edges of an n-vertex graph while maintaining a memory of size $O(n \operatorname{polylog} n)$. Our method immediately yields a very simple two-pass algorithm for Maximum Bipartite Matching (MBM) with approximation factor 0.5857, which only runs the Greedy matching algorithm in each pass. This slightly improves on the much more involved 0.583-approximation algorithm of Esfandiari et al. [ICDMW 2016]. To obtain our main result, we combine our method with a residual sparsity property of the random order Greedy algorithm and give a one-pass random order streaming algorithm for MBM with approximation factor 0.5395. This substantially improves upon the one-pass random order 0.505-approximation algorithm of Konrad et al. [APPROX 2012].

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms, Theory of computation → Graph algorithms analysis

Keywords and phrases Matchings, augmenting paths, streaming algorithms, random order

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.74

1 Introduction

Computing Large Matchings. Given a bipartite graph $G = (A, B, E)$, a matching $M \subseteq E$ in G is a subset of non-adjacent edges. In this paper, we address the Maximum Bipartite Matching (MBM) problem, which consists of finding a matching of maximum size. Many classic algorithms for MBM, such as the Hopcroft-Karp algorithm [20] or Edmonds' algorithm [11], as well as many more recent algorithms, first compute an arbitrary matching and then iteratively improve it by finding augmenting paths until it is of maximum size. A good starting point is a *maximal matching*, i.e., a matching that cannot be enlarged by adding an edge outside the matching to it, which is known to be of size at least $1/2$ times the size of a *maximum matching*, i.e., one of maximum size. A maximal matching is for example produced by the Greedy matching algorithm, which processes the edges of a graph in arbitrary order and adds the current edge to an initially empty matching if the resulting set is still a matching. For an integer $k \geq 1$, a $(2k+1)$-*augmenting path* $P = e_1, e_2, e_3, \ldots, e_{2k+1}$ with respect to a matching M is a path of odd length that alternates between edges outside M and edges contained in M such that both e_1 and e_{2k+1} are incident to vertices that are not matched in M. Since P contains $k+1$ edges outside M and k edges of M, removing the matched edges

in P from M and inserting the unmatched edges in P into M thus increases the size of M by 1. It is known that a non-maximum matching always admits an augmenting path, and, thus, repeatedly finding one and augmenting eventually yields a maximum matching.

To decrease the number of improvement steps required, one common approach is to compute a large *set of disjoint augmenting paths* and augment along each of them simultaneously. This approach is particularly beneficial when designing algorithms in restricted computational models such as the data streaming model (see below) or various distributed computational models, since typically the number of passes (streaming) or rounds (distributed algorithms) grows linearly with the number of augmentation rounds.

Our Results. In this paper, we give a new method that allows us to find a large fraction of disjoint 3-augmenting paths such that, when augmenting along those paths, the resulting matching is of size at least $(2-\sqrt{2})|M^*| - o(|M^*|) \approx 0.5857|M^*| - o(|M^*|)$ with high probability, where M^* is a maximum matching (**Theorem 8**). The strength of our method lies both in its simplicity and effectiveness: It only requires running the Greedy matching algorithm on a random subgraph to produce the necessary edges. Despite its simplicity, it outperforms other more complicated methods and yields improvements over the state-of-the-art one- and two-pass data streaming algorithms for matchings (see below). Our method can also be applied repeatedly and for example yields a 3-pass streaming algorithm that also outperforms the currently best 3-pass streaming algorithm known.

Applications to Data Streaming Algorithms. While our method can be applied in essentially all computational models that allow an implementation of the Greedy matching algorithm, it has been designed with the *data streaming model* in mind. Given an n-vertex graph $G = (V, E)$, a p-pass, s-space data streaming algorithm processing G performs p passes over the edges E of G (the edges may arrive in arbitrary, potentially adversarial order) while maintaining a memory of size s. Since many graph problems require space $\Omega(n \log n)$ (observe that storing a large matching already requires this amount of space) [32], research has focussed on the *semi-streaming model* [16], where a graph streaming algorithm is allowed to use space $O(n \operatorname{polylog} n)$. Concerning the MBM problem, Feigenbaum et al. [16] observed that the Greedy matching algorithm constitutes a one-pass $\frac{1}{2}$-approximation semi-streaming algorithm for MBM. Interestingly, despite intense research efforts, no better one-pass streaming algorithms are known, even if space $O(n^{2-\delta})$ is granted, for any $\delta > 0$, while lower bounds only rule out the existence of semi-streaming algorithms with approximation ratio larger than $1 - 1/e \approx 0.6321$ [22, 18]. Konrad et al. [26] studied minimal extensions to the one-pass semi-streaming model that allow us to improve on Greedy. They showed that approximation ratios strictly larger than $\frac{1}{2}$ can be obtained if either the edges of the input graph arrive in uniform random order, or a second pass is granted. More specifically, they gave a symbolic improvement showing that a $(\frac{1}{2} + 0.005)$-approximation can be obtained if edges arrive in random order, and a $(\frac{1}{2} + 0.02)$-approximation can be achieved if two passes are allowed. Their two-pass result has since been improved by Kale and Tirodkar [21] to $\frac{1}{2} + \frac{1}{16} = \frac{1}{2} + 0.0625$ and independently by Esfandiari et al. to $\frac{1}{2} + 0.083$ [14].

Our method for finding augmenting paths immediately yields a two-pass semi-streaming algorithm with approximation factor 0.5857 (**Theorem 9**), thus slightly improving over the algorithm of Esfandiari et al. [14]. Our algorithm has constant update time (i.e., the running time between two read operations from the stream) and does not need a post-processing step, while the algorithm of Esfandiari et al. requires the computation of a maximum matching in the post-processing step. Our main result is a one-pass random order semi-streaming

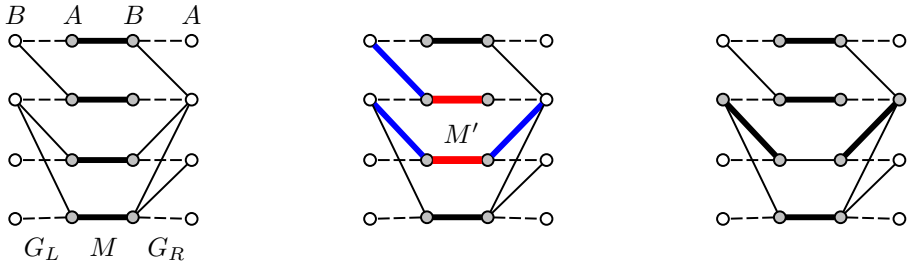


Figure 1 Left: Bipartite graph $G = (A, B, E)$ with maximal matching M. The dotted edges show a perfect matching in G. Matched vertices are grey, free vertices are white. Center: Subset $M' \subseteq M$ is highlighted in red. The blue edges are produced by the runs of Greedy on G'_L and G'_R. Observe that one 3-augmenting path is found. Right: M after the augmentation.

algorithm with approximation factor 0.5395 (**Theorem 16**), showing that more substantial improvements over $\frac{1}{2}$ than the symbolic improvement given by Konrad et al. [26] are possible in the random order scenario. This algorithm is obtained by combining our method for finding augmenting paths with a residual sparsity property of the random order Greedy matching algorithm (e.g. [25]) that has recently been exploited in various contexts [25, 1, 17].

Techniques. For illustration purposes, consider a bipartite graph $G = (A, B, E)$ that contains a *perfect matching* M^*, i.e., a matching that matches all vertices, and a maximal matching M with $|M| = \frac{1}{2}|M^*|$. It can be seen that $M \oplus M^* := (M \setminus M^*) \cup (M^* \setminus M)$ forms a set of $\frac{1}{2}|M^*|$ disjoint 3-augmenting paths. In other words, there exists a matching of size $\frac{1}{2}|M^*|$ in graph $G_L := G[A(M) \cup \overline{B(M)}]$, where $A(M)$ is the set of matched A-vertices, and $\overline{B(M)} := B \setminus B(M)$, and also one of size $\frac{1}{2}|M^*|$ in $G_R = G[\overline{A(M)} \cup B(M)]$, see Figure 1.

We now sample a random subset of edges $M' \subseteq M$ such that every edge $e \in M$ is included in M' with probability p. Using an argument by Konrad et al. [26], it follows that when running the Greedy matching algorithm on the subgraph $G'_L := G[A(M') \cup \overline{B(M)}] \subseteq G_L$, then in expectation a $\frac{1}{1+p}$ fraction of the vertices $A(M')$ is matched. Observe that if we chose $p = 1$, then half of the vertices get matched, which is what we expect from the Greedy matching algorithm. However, if we chose p substantially smaller than 1, then a large fraction of vertices of $A(M')$ is matched. We also apply this argument to subgraph $G'_R := G[B(M') \cup \overline{A(M)}] \subseteq G_R$, which thus allows us to find large matchings in both subgraphs G'_L and G'_R and in turn extract many 3-augmenting paths. Observe that this method directly yields a two-pass semi-streaming algorithm, by computing a maximal matching in the first pass, and augmenting it using the described method in the second pass.

The main shortcoming of this method is that the result by Konrad et al. [26] only holds in expectation, which would imply that our result also only holds in expectation. We therefore strengthen their result and prove that a similar version holds with high probability. Our proof models the execution of the algorithm with a Doob martingale and applies Azuma's inequality to obtain a concentration result. We then use our result and additional combinatorial arguments to bound the number of 3-augmenting paths found.

Our one-pass random order streaming algorithm combines our method for finding 3-augmenting paths with a *residual sparsity* property of the random order Greedy algorithm. We run Greedy on the first $\frac{1}{\log n}$ fraction of edges in the stream to produce a matching M. The residual sparsity property states that the residual graph $H = G[V \setminus V(M)]$ contains $O(n \operatorname{polylog} n)$ edges with high probability, which we then collect while processing the remaining edges in the stream. Our main argument is as follows: If $|M|$ is relatively small, then the residual graph H contains a sufficiently large matching. On the other hand, if $|M|$ is relatively large (close to a $\frac{1}{2}$-approximation), then we can use the remainder of the stream to find 3-augmenting paths using the method described above.

Comparison to Esfandiari et al. [14] and Kale and Tirodkar [21]. The two-pass streaming algorithms of Esfandiari et al. and Kale and Tirodkar proceed similarly in that they compute a maximal matching M in the first pass and then find additional edges in the second pass that are used to augment M. Their algorithms are in fact almost identical and only differ in the post-processing stage. With $G_L = G[A(M) \cup \overline{B(M)}]$ and $G_R = G[B(M) \cup \overline{A(M)}]$ being as above, they compute *incomplete semi-matchings* S_L in G_L and S_R in G_R, i.e., subsets of edges such that every vertex in $A(M)$ ($B(M)$) is matched at most once in S_L (resp. S_R) and every vertex $\overline{B(M)}$ (resp. $\overline{A(M)}$) is matched at most k times, for some integer k. Using a Greedy algorithm for computing S_L and S_R, it can be seen that a large fraction of vertices $A(M)$ (resp. $B(M)$) are matched in S_L (resp. S_R). This allows the extraction of multiple 3-augmenting paths. In Kale and Tirodkar, the extraction step is done greedily, which is efficient but leads to a worse approximation factor than in Esfandiari et al. Esfandiari et al. solve an optimization problem in a post-processing phase that allows the extraction of more 3-augmenting paths, which in turn leads to an improved approximation guarantee. Our method is much simpler in this regard, since our additional edges form matchings and it is thus straightforward to extract 3-augmenting paths.

Comparison to Konrad et al. [26]. The one-pass random order algorithm by Konrad et al. proceeds as follows: First, run Greedy on roughly the first third of the edges in the input stream and obtain a matching M. Konrad et al. prove that if Greedy on the entire input stream produces a matching that is close to a $\frac{1}{2}$-approximation, then the matching is built early on, i.e., $|M|$ is relatively large. They then use the remaining part of the stream for finding 3-augmenting paths. To this end, they compute a matching in G_L on roughly the next third of the edges, and then use the last third to compute a matching in G_R to complete the 3-augmenting paths. Their method only yields a marginal improvement over 1/2 and their result only holds in expectation.

Observe that we also argue that the matching M is large, which we achieve by exploiting the residual sparsity property of Greedy. While Konrad et al. have already processed a third of the edges at this stage, we have only processed a $\frac{1}{\log(n)}$ fraction, and there are thus more remaining edges to our disposal for finding 3-augmenting paths. Further, our method produces more 3-augmenting paths than the method proposed by Konrad et al.

Further Related Work. Matching problems are the most studied graph problems in the data streaming model. Besides the already mentioned works, algorithms have been designed for weighted matchings (e.g. [16, 29, 33, 9, 31]), multiple passes (e.g. [29, 12, 2]), insertion/deletion streams (e.g. [10, 6, 24, 7, 4, 30]), sparse graphs (e.g. [13, 8]), and other variants of the matching problem [27]. Regarding random order streams, Kapralov et al. [23] showed that the size of a maximum matching can be estimated within a poly-log factor using poly-log space, and a $(2/3 - \epsilon)$-approximation can be computed using $\tilde{O}(n^{3/2})$ space [3].

Outline. We proceed as follows. We first give notation and definitions in Section 2. We then present our method for finding a large set of disjoint 3-augmenting paths in Section 3. Implementation details when implementing our method in the adversarial order streaming model are then discussed in Section 4. In Section 5, we give our one-pass random order algorithm. Finally, we conclude in Section 6 with open problems.

2 Preliminaries

Notation. Let $G = (A, B, E)$ be a bipartite graph. We generally use n to denote the number of vertices, i.e., $n = |A| + |B|$, and $m = |E|$ to denote the number of edges. For a subset of vertices $U \subseteq A \cup B$ and a subset of edges $F \subseteq E$, we denote the vertex induced subgraph of G by vertices U by $G[U]$, and the edge induced subgraph of G by edges F by $G[F]$. Let M be a matching in G. We denote by $A(M)$ ($B(M)$) the vertices of A (resp. B) that are matched by M, and we write $V(M) = A(M) \cup B(M)$. Similarly, for an edge $e \in E$, we write $A(e)$ to denote its incident A-vertex, $B(e)$ to denote its B vertex, and $V(e) = \{A(e), B(e)\}$. The complement of a subset $A' \subseteq A$ ($B' \subseteq B$) is denoted by $\overline{A'} = A \setminus A'$ (resp. $\overline{B'} = B \setminus B'$).

The *matching number* of a graph G, i.e., the size of a maximum matching in G, is denoted $\mu(G)$. We write $opt(G)$ to denote an arbitrary but fixed maximum matching in G. For two sets X, Y, we write $X \oplus Y := (X \setminus Y) \cup (Y \setminus X)$ to denote their symmetric difference. For a graph G, $\Delta(G)$ denotes the maximum degree.

Concentration Bounds. In this paper, we will use two concentration bounds. The first one is Azuma's inequality for martingales (we refer the reader to [28] for the an introduction to martingales and Azuma's inequality), and the second is a Chernoff-type bound for weakly dependent random variables.

▶ **Theorem 1** (Azuma's Inequality ([5, 28])). *Suppose that $X_0, X_1, X_2, \ldots$ is a martingale and let $|X_i - X_{i-1}| \leq c_i$ for suitable constants c_i. Then:*

$$\mathbb{P}\left[|X_n - X_0| \geq t\right] \leq 2\exp\left(\frac{-t^2}{2\sum_{i=1}^{n} c_i^2}\right) .$$

▶ **Theorem 2** (Chernoff Bound for Weakly Dependent Variables, e.g. [15]). *Let $X_1, X_2, \ldots, X_n$ be 0/1 random variables for which there is a $p \in [0,1]$ such that for all $k \in [n]$ the inequality*

$$\mathbb{P}\left[X_k = 1 \,|\, X_1, X_2, \ldots, X_{k-1}\right] \leq p$$

holds (i.e., the probability of $X_k = 1$ conditioned on any possible outcome of $X_1, \ldots, X_{k-1}$ is at most p). Let further $\mu \geq p \cdot n$. Then, for every $\delta > 0$:

$$\mathbb{P}\left[\sum_{i=1}^{n} X_i \geq (1+\delta)\mu\right] \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu} .$$

We will say that an event occurs with *high probability in variable x*, if the the probability of the event occurring is at least $1 - x^{-C}$, for some $C \geq 1$. If we do not mention x explicitly, then the high probability statement is in n, the number of vertices of the input graph.

We say that an algorithm is a C-approximation algorithm for MBM if it computes a matching M of size at least $C \cdot \mu(G) - o(\mu(G))$.

3 Finding a Large Set of Disjoint 3-augmenting Paths

We now present an algorithm that, given a maximal matching M in a bipartite graph $G = (A, B, E)$, finds a set of disjoint 3-augmenting paths $\mathcal{P}$ by running the Greedy matching algorithm on a random subgraph of G. The set $\mathcal{P}$ is such that, when augmenting M along the paths $\mathcal{P}$, a matching of size at least $(2-\sqrt{2})\mu(G) - o(\mu(G)) \approx 0.5857\mu(G) - o(\mu(G))$ is obtained.

Algorithm 1 Finding a large set of 3-augmenting paths.

Input: Bipartite graph $G = (A, B, E)$, maximal matching M, parameter $0 < p < 1$

1. Sample each edge $e \in M$ with probability p; let M' be the resulting sample
2. $M_L \leftarrow \textsc{Greedy}(G[A(M') \cup \overline{B(M)}])$; $M_R \leftarrow \textsc{Greedy}(G[\overline{A(M)} \cup B(M')])$
3. $\mathcal{P} \leftarrow \{\text{paths } b'a, ab, ba' \mid b'a \in M_L, ab \in M, ba' \in M_R\}$
4. **return** $\mathcal{P}$

Our algorithm is illustrated in Algorithm 1. For the sake of a clear presentation, the algorithm employs two invocations of Greedy on two disjoint subgraphs. This is equivalent to invoking Greedy only once on their union. Our algorithm is parametrized by a sampling probability p. To obtain the claimed bound stated above, we will later optimize p.

To obtain a better understanding of our algorithm, we first discuss structural properties that help us locate 3-augmenting paths in G with respect to the matching M.

Let M^* be a maximum matching in G and let ϵ be such that $|M| = (\frac{1}{2} + \epsilon)|M^*|$. Observe first that $M \oplus M^*$ contains a collection of $(\frac{1}{2} - \epsilon)|M^*|$ disjoint augmenting paths. Further, observe that the endpoints of each augmenting path are a free vertex in A (i.e., a vertex in $\overline{A(M)}$) and a free vertex in B. Hence, the subgraphs $G_L := G[A(M) \cup \overline{B(M)}]$ and $G_R := G[\overline{A(M)} \cup B(M)]$ each contain a matching of size $(\frac{1}{2} - \epsilon)|M^*|$. We summarize this in Observation 3:

▶ **Observation 3.** *Let ϵ be such that $|M| = (\frac{1}{2} + \epsilon)\mu(G)$. Then:*

$$\min\{\mu(G_L), \mu(G_R)\} \geq (\frac{1}{2} - \epsilon)\mu(G) \ .$$

Suppose now that ϵ is small. Further, suppose that we could compute maximum matchings M_L^* in G_L and M_R^* in G_R. Then for almost every edge $e \in M$ there are edges $e_l \in M_L$ and $e_r \in M_R$ such that $e_l e e_r$ forms a 3-augmenting path. We will call e_l a left wing for edge e and e_r a right wing for edge e.

Our augmentation method should of course not be based on computing maximum matchings themselves. We therefore proceed differently. First, observe that if we computed maximal matchings, i.e., $\frac{1}{2}$-approximations, in G_L and G_R, then we may not find any 3-augmenting path at all, since it may happen that we find left wings for half of the edges of M, and right wings for the other half. Our strategy therefore is as follows: We first sample a subset of edges $M' \subseteq M$, where each edge of M is included in M' with probability p, and we attempt to augment only the edges in M' by computing Greedy matchings in the subgraphs $G'_L := G[A(M') \cup \overline{B(M)}]$ and $G'_R := G[\overline{A(M)} \cup B(M')]$. Konrad et al. [26] proved that, in expectation, the resulting matchings are essentially $\frac{1}{1+p} \geq \frac{1}{2}$-approximations, albeit for a slightly different notion of approximation, which is nevertheless suitable for our purposes:

▶ **Theorem 4** (Konrad et al. [26])**.** *Let $G = (U, V, E)$ be a bipartite graph, and let $U' \subseteq U$ be such that every vertex $u \in U$ is included in U' with probability p ($p \in [0, 1]$). Then, for any arbitrary but fixed order in which* Greedy *processes the edges, the following holds:*

$$\mathbb{E}_{U'}|\textsc{Greedy}(G[U' \cup V])| \geq \frac{p}{1+p}\mu(G) \ .$$

Hence, if ϵ is close to 0, and p is substantially smaller than 1, then it follows from the previous theorem that a large fraction of the vertices $A(M')$ will be matched by Greedy in G'_L, and a large fraction of the vertices of $B(M')$ will be matched by Greedy in G'_R. This

in turn implies that a substantial amount of edges of M' both have left and a right wings and are thus included in 3-augmenting paths.

Before we make this intuition formal, we point out one shortcoming of applying the previous theorem by Konrad et al. directly. They prove that the resulting matching is large only *in expectation*, which in turn would imply that our result only holds in expectation. We therefore first strengthen their result and prove that a similar version holds with high probability. To this end, we first prove a technical lemma that is employed in the proof of our strengthened theorem.

▶ **Lemma 5.** *Let $G = (U, V, E)$ be a bipartite graph and let $u \in U, v \in V$ be arbitrary vertices. Let $U' \subseteq U$ be such that every vertex $u \in U$ is included in U' with probability p. Then, for any arbitrary but fixed order in which* Greedy *processes the edges, the following holds:*

$$0 \leq \mathbb{E}_{U'}|\textsc{Greedy}(G[U' \cup V])| - \mathbb{E}_{U'}|\textsc{Greedy}(G[(U' \cup V) \setminus \{u, v\}])| \leq 2 \ .$$

Proof. First, observe that

$$\mathbb{E}_{U'}|\textsc{Greedy}(G[U' \cup V])| - \mathbb{E}_{U'}|\textsc{Greedy}(G[(U' \cup V) \setminus \{u, v\}])| = \mathbb{E}_{U'}\left(|\textsc{Greedy}(G[U' \cup V])| - |\textsc{Greedy}(G[(U' \cup V) \setminus \{u, v\}])|\right) \ .$$

We will prove next that $0 \leq \textsc{Greedy}(G[U' \cup V]) - \textsc{Greedy}(G[U' \cup V - \{u, v\}]) \leq 2$ holds for any $U' \subseteq U$, which then proves the lemma. We will in fact argue the stronger statement that for any graph $G = (V, E)$ and any vertex $v \in V$, the inequality $0 \leq \textsc{Greedy}(G) - \textsc{Greedy}(G \setminus \{v\}) \leq 1$ holds. The result then follows by applying this statement twice.

Consider thus an arbitrary graph $G = (V, E)$ and a vertex $v \in V$. First observe that if $\textsc{Greedy}(G)$ leaves v unmatched, then $\textsc{Greedy}(G) = \textsc{Greedy}(G \setminus \{v\})$. If $\textsc{Greedy}(G)$ matches v, then it is not hard to see that $\textsc{Greedy}(G) \oplus \textsc{Greedy}(G \setminus \{v\})$ consists of one alternating path whose one endpoint is v. This further implies that $\textsc{Greedy}(G \setminus \{v\}) \leq \textsc{Greedy}(G) \leq \textsc{Greedy}(G \setminus \{v\}) + 1$, which completes the proof. ◀

We now give our strengthened version of Theorem 4.

▶ **Theorem 6.** *Let $G = (U, V, E)$ be a bipartite graph, and let $U' \subseteq U$ be such that every vertex $u \in U$ is included in U' with probability p ($p \in [0, 1]$). Then, for any arbitrary but fixed order in which* Greedy *processes the edges, the following holds with probability at least $1 - (\mu(G))^{-12}$:*

$$|\textsc{Greedy}(G[U' \cup V])| \geq \frac{p}{1+p}\mu(G) - o(\mu(G)) \ .$$

Proof. Let $X := |\textsc{Greedy}(G[U' \cup V])|$. By Theorem 4 we have $\mathbb{E}X \geq \frac{p}{1+p}\mu(G)$.

For $1 \leq i \leq n$, let Z_i be the ith edge selected by Greedy, and let $Z_i = \perp$ if $i > X$. Let Y_i be the Doob martingale induced by the first i choices of the algorithm, i.e.,

$$Y_i := \mathbb{E}_{Z_{i+1}, Z_{i+2}, \ldots, Z_n}(X \mid Z_1, \ldots, Z_i) \ .$$

Observe that the expectation in the previous expression is in itself a random variable, since the expectation is only taken over $Z_{i+1}, \ldots, Z_n$, while $Z_1, \ldots, Z_i$ are random variables. It is not hard to check that the sequence $(Y_i)_i$ always forms a martingale, independently of the underlying sequence Z_i. Observe next that $Y_0 = \mathbb{E}X$ and $Y_n = X$. We thus need to show that $|Y_n - Y_0|$ is small with high probability. To this end, we will apply Azuma's inequality, which requires bounding the differences $|Y_{i+1} - Y_i|$, for every i, first.

First, observe that $|Y_{i+1} - Y_i| = 0$ for every $i \geq X$. Next, we claim that $|Y_{i+1} - Y_i| \leq 1$, for every $i < X$. Indeed, observe that Y_i is the expected size of the computed matching conditioned on the first i choices of the algorithm. We can thus rewrite Y_i as:

$$Y_i = i + \mathbb{E}_{U'}|\textsc{Greedy}(H_i)| \ ,$$

where $H_i := G[(U' \cup V) \setminus \cup_{j \leq i} V(Z_j)]$ is the residual graph obtained when removing the vertices incident to the first i selected edges. We thus obtain:

$$\begin{aligned} Y_{i+1} - Y_i &= 1 + \mathbb{E}_{U'}|\textsc{Greedy}(H_{i+1})| - \mathbb{E}_{U'}|\textsc{Greedy}(H_i)| \\ &= 1 + \mathbb{E}_{U'}|\textsc{Greedy}(H_i \setminus V(Z_{i+1}))| - \mathbb{E}_{U'}|\textsc{Greedy}(H_i)| \in \{-1, 0, 1\} \ , \end{aligned}$$

where we applied Lemma 5.

Next, since $X \leq \mu(G[U' \cup V]) \leq \mu(G)$, we have $|Y_{i+1} - Y_i| \leq 1$ for every $i \leq \mu(G)$, and $|Y_{i+1} - Y_i| = 0$ for every $i > \mu(G)$. Applying Azuma's Inequality (Theorem 1), we obtain:

$$\mathbb{P}\left[|Y_n - Y_0| \geq 5\sqrt{\mu(G)\ln(\mu(G))}\right] \leq \mu(G)^{-12} \ . \qquad \blacktriangleleft$$

Equipped with Theorem 6, we now show that our algorithm finds many disjoint 3-augmenting paths, provided that M is close to a $\frac{1}{2}$-approximation.

▶ **Lemma 7.** *Consider Algorithm 1 and suppose that $|M| = (\frac{1}{2} + \epsilon)\mu(G)$. Then, with probability at least $1 - \mu(G)^{-10}$,*

$$|\mathcal{P}| \geq \mu(G)p\left(\frac{1-2\epsilon}{1+p} - \frac{1}{2} - \epsilon\right) - o(\mu(G)) \ .$$

Proof. First, by an application of a Chernoff bound, we obtain $|M'| = p|M| \pm O(\sqrt{|M|\ln(|M|)})$, with probability at least $1 - |M|^{-C}$, for an arbitrarily large constant C. Next, by Theorem 6 and Observation 3, with probability at least $1 - 2(\mu(G))^{-12}$, we have $|M_L| \geq \frac{p}{1+p}(\frac{1}{2} - \epsilon)\mu(G) - o(\mu(G))$ and $|M_R| \geq \frac{p}{1+p}(\frac{1}{2} - \epsilon)\mu(G) - o(\mu(G))$. Observe that at most $|M'| - |M_L|$ edges of M' do not have a left wing, and at most $|M'| - |M_R|$ edges of M' do not have a right wing. Hence, at least $|M'| - (|M'| - |M_L|) - (|M'| - |M_R|) = |M_L| + |M_R| - |M'|$ edges have both left and right wings and therefore form 3 augmenting paths. We thus obtain:

$$\begin{aligned} |\mathcal{P}| &\geq |M_L| + |M_R| - |M'| \\ &\geq 2 \cdot \frac{p}{1+p}(\frac{1}{2} - \epsilon)\mu(G) - o(\mu(G)) - p|M| - O(\sqrt{|M|\ln(|M|)}) \\ &\geq 2 \cdot \frac{p}{1+p}(\frac{1}{2} - \epsilon)\mu(G) - p(\frac{1}{2} + \epsilon)\mu(G) - o(\mu(G)) \\ &= \mu(G)p\left(\frac{1-2\epsilon}{1+p} - \frac{1}{2} - \epsilon\right) - o(\mu(G)) \ . \end{aligned}$$

By the union bound, the error is bounded by $|M|^{-C} + 2(\mu(G))^{-12} \leq (\mu(G))^{-10}$. $\blacktriangleleft$

We are now ready to prove our main theorem:

▶ **Theorem 8.** *Let M be a maximal matching. Then, setting $p = \sqrt{2} - 1$ in Algorithm 1 guarantees that M augmented by $\mathcal{P}$ gives a matching of size at least $(2 - \sqrt{2})\mu(G) - o(\mu(G)) \approx (\frac{1}{2} + 0.0857)\mu(G) - o(\mu(G))$ with high probability in $\mu(G)$.*

Proof. Observe that the final matching is of size $|M| + |\mathcal{P}|$. Let ϵ be such that $|M| = (\frac{1}{2} + \epsilon)\mu(G)$. By Lemma 7, we have

$$|M| + |\mathcal{P}| \quad \geq \quad (\frac{1}{2} + \epsilon)\mu(G) + \mu(G)p\left(\frac{1-2\epsilon}{1+p} - \frac{1}{2} - \epsilon\right) - o(\mu(G)) \ . \tag{1}$$

It can be seen that for any value of p, the right side of Inequality 1 is minimized for $\epsilon = 0$. On the other hand, for any value of ϵ, the value $p(\epsilon) = \sqrt{\frac{1-2\epsilon}{\frac{1}{2}+\epsilon}} - 1$ maximizes Inequality 1. Using $\epsilon = 0$ and $p(0) = \sqrt{2} - 1$ in Inequality 1 gives $|M| + |\mathcal{P}| \geq (2 - \sqrt{2})\mu(G) - o(\mu(G))$. ◀

Multiple augmentation rounds with decreasing values of p allow further improvements. For example, a second round with $p = \sqrt{\frac{2-\sqrt{2}}{\sqrt{2}-1}} - 1 \approx 0.1892$ guarantees that the resulting matching is of size at least $0.6067\mu(G) - o(\mu(G))$. As we will discuss in the next section, this can give a 3-pass streaming algorithm for MBM with approximation factor 0.6067, which slightly improves the 3-pass 0.605-approximation algorithm by Esfandiari et al. [14].

4 Adversarial Order Streams

Our method for finding augmenting paths given in Section 3 can directly be implemented in the streaming model. In the first pass, we compute a maximal matching M. If the current edge is added to M, then with probability p we add the edge to M' as well. In the second pass, we run Greedy on the subgraphs G'_L and G'_R and as soon as a 3-augmenting path is completed, we augment M. This can be done with constant update times.

Since we would like our streaming algorithm to succeed with high probability in n, the number of vertices, we need to address the fact that our method as stated in Theorem 8 only succeeds with high probability in $\mu(G)$, the size of a maximum matching in G. If $\mu(G)$ is of size at least, say, $\Omega(n^{\frac{1}{4}})$, our method can also gives a high probability result with respect to n. To deal with the case $\mu(G) = o(n^{\frac{1}{4}})$ we run the 1-pass algorithm of Chitnis et al. [7] in parallel to our algorithm, which computes a subset of edges $E' \subseteq E$ of size $O(n^{\frac{1}{2}})$ that contains a maximum matching provided that $\mu(G) = O(n^{\frac{1}{4}})$. Observe that after the first pass, we know in which of the two cases we are. We then run the Hopcroft-Karp maximum matching algorithm [20] in time $O(\sqrt{n} \cdot \sqrt{n}) = O(n)$ on the set of collected edges. To obtain a streaming algorithm with constant update time, we amortize the previous computation during the processing of the second pass, which is possible under the natural assumption that $m = \Omega(n)$. This gives the following theorem:

▶ **Theorem 9.** *There is a two-pass streaming algorithm for* MBM *with approximation factor* $2 - \sqrt{2} \approx \frac{1}{2} + 0.0857$ *that succeeds with high probability (in* n*). Using one additional pass, a* 0.6067*-approximation algorithm can be obtained.*

5 1-pass Random Order Streaming Algorithm

In this section, we assume that $\mu(G) = \Omega(n^{\frac{1}{4}})$. To deal with the case $\mu(G) = o(n^{\frac{1}{4}})$ we run the algorithm of Chitnis et al. [7] as outlined in Section 4 in parallel and compute and output a maximum matching after processing the stream. We also assume that the input graph has at least $C_1 \cdot n \log^{C_2} n$ edges, for suitably large constants C_1, C_2. If this is not the case then we could simply store all edges within the semi-streaming space constraint and compute and output a maximum matching.

Algorithm 2 One-pass random order matching algorithm.

Input: Bipartite graph $G = (A, B, E)$ with m edges, parameter $0 < p < 1$

Let $\pi = \pi[1], \pi[2], \ldots \pi[m]$ be the edges of G in uniform random order

1. $M \leftarrow$ Greedy$(\pi[1, \frac{m}{\log n}])$
2. Let $M' \subseteq M$ be such that every edge of M is included in M' with probability p
3. **while** processing $\pi(\frac{m}{\log n}, m]$ **do in parallel:**
   a. Compute set E_M of edges $ab \in \pi(\frac{m}{\log n}, m]$ with $a, b \notin V(M)$; if $|E_M| \geq C \cdot n \log^2 n$, for some appropriate large constant C, then **abort**
   b. $M_L \leftarrow$ Greedy(G_L^r), where G_L^r is the subgraph of G induced by all edges $\pi(\frac{m}{\log n}, m]$ between $A(M')$ and $\overline{B(M)}$
   c. $M_R \leftarrow$ Greedy(G_R^r), where G_R^r is the subgraph of G induced by all edges $\pi(\frac{m}{\log n}, m]$ between $\overline{A(M)}$ and $B(M')$
4. $\mathcal{P} \leftarrow \{\text{paths } b'a, ab, ba' \mid b'a \in M_L, ab \in M, ba' \in M_R\}$
5. **if** $|\mathcal{P}| \geq \mu(G[E_M])$ **then return** M augmented by $\mathcal{P}$
   else return $M \cup opt(G[E_M])$

Our 1-pass random order streaming algorithm combines our method for finding augmenting paths with a *residual sparsity* property of the random order Greedy matching algorithm:

▶ **Theorem 10** (Residual Sparsity of Greedy)**.** *Suppose that* Greedy *processes the edges E of a graph $G = (V, E)$ with $m = |E|$ in uniform random order. Let M_i be the matching produced by* Greedy *after having processed the ith edge. Then:*

$$\Delta(G[V \setminus V(M_i)]) = O(\frac{m \log n}{i})$$

with probability $1 - n^{-12}$ (over the uniform random ordering of the edges).

This theorem is implied by a similar theorem concerning the random order Greedy algorithm for independent sets as given in [25]. Observe that the Greedy algorithm for matchings on a randomly ordered sequence of the edges of a graph G can be seen as the Greedy algorithm for independent sets on a randomly ordered sequence of the vertices of the line graph $L(G)$.

Our one-pass random order algorithm is parametrized by a probability p, and is illustrated in Algorithm 2. In this listing, we write $\pi = \pi[1], \pi[2], \ldots, \pi[m]$ to be a uniform random ordering of the edges E. For $a < b$ we also write $\pi[a, b]$ to denote edges $\pi[a], \pi[a+1], \ldots, \pi[b]$, and $\pi(a, b]$ to denote edges $\pi[a+1], \pi[a+2], \ldots, \pi[b]$.

We run Greedy on the first $\frac{m}{\log n}$ edges to compute a matching M. Theorem 10 implies that the maximum degree in the residual graph $H := G[V \setminus V(M)]$ is $O(\log^2 n)$. This allows us to collect the entire residual graph (i.e., set E_M) within the semi-streaming space bound, since it has $O(n \log^2 n)$ edges with high probability. We abort if $|E_M|$ becomes too large.

In the next stage, we proceed as in our two-pass algorithm: We sample a subset of edges $M' \subseteq M$ and we try to find 3-augmenting paths for M' by computing matchings M_L and M_R in the subgraphs G_L^r and G_R^r. Ideally we would like to search for left and right wings in the subgraphs $G_L := G[A(M) \cup \overline{B(M)}]$ and $G_R := G[\overline{A(M)} \cup B(M)]$. Since however the first $\frac{1}{\log n}$ fraction of edges in the stream has already been processed, we can only search for augmenting paths in G_L^r and G_R^r. Concentration bounds however allow us to prove that not many important edges have arrived among the first $\frac{1}{\log n}$ fraction of edges (Lemma 14).

Our analysis is build on the following important observation. Suppose first that the matching M is small, i.e., $|M| = \alpha|M^*|$, for a small value of α. Then we will argue in the next lemma that a maximum matching in the residual graph is large:

▶ **Lemma 11.** *Let α be such that $|M| = \alpha|M^*|$, and let $H := G[E_M]$ $(= G[V \setminus V(M)])$ be the residual graph. Then:*

$$\mu(H) \geq (1 - 2\alpha)|M^*| \ .$$

Proof. Let M^* be a maximum matching in G. Let $M_1^* \subseteq M^*$ be those edges of M^* that share at least one endpoint with an edge in M, and let $M_2^* = M^* \setminus M_1^*$. Then $|M_1^*| \leq 2|M|$, since each edge of M can only be incident to at most two edges of M^*. Observe further that $M_2^* \subseteq E_M$. Hence: $\mu(H) \geq |M_2^*| = |M^*| - |M_1^*| \geq |M^*| - 2|M| = (1 - 2\alpha)|M^*|$. ◀

By combining M with a maximum matching in H we obtain the following corollary:

▶ **Corollary 12.** *Algorithm 2 finds a matching of size at least $(1-\alpha)|M^*|$ with high probability.*

The previous corollary shows that either the matching $M \cup opt(H)$ is large (if α is small), or the matching M itself is already reasonably large (if α is large). This is an important property since we next attempt to augment M, which necessitates that M is already close to a $\frac{1}{2}$-approximation. For this to succeed, we need to show that $\mu(G_L^r)$ and $\mu(G_R^r)$ are large. To this end, let δ be such that $|M| + \mu(G[E_M]) = (\frac{1}{2} + \delta)\mu(G)$. We will first bound $\mu(G_L)$ and $\mu(G_R)$ and then prove a similar bound for $\mu(G_L^r)$ and $\mu(G_R^r)$.

▶ **Lemma 13.** *Suppose that $|M| + \mu(G[E_M]) = (\frac{1}{2} + \delta)\mu(G)$. Then:*

$$\min\{\mu(G_L), \mu(G_R)\} \geq (\frac{1}{2} - \delta)\mu(G) \ .$$

Proof. Let M^* be a maximum matching in G and let M_H^* be an arbitrary maximum matching in $H(= G[E_M])$. First, it is not hard to see that $M \cup M_H^*$ is a maximal matching. Next, consider the set of edges $M^* \oplus (M \cup M_H^*)$. Since $|M| + |M_H^*| = (\frac{1}{2} + \delta)\mu(G)$, the set $M^* \oplus (M \cup opt(H))$ contains $(\frac{1}{2} - \delta)\mu(G)$ augmenting paths.

Observe that none of these augmenting paths only contain edges of M^* and M_H^*, since this would imply that M_H^* is not maximum in H. Consider now one such augmenting path P and remove all edges of M_H^* from P. Then P contains at least one augmenting path that only contains edges from M and M^*. Applying this argument to all augmenting paths, this proves that there are matchings in G_L and G_R of sizes $(\frac{1}{2} - \delta)\mu(G)$. ◀

▶ **Lemma 14.** *Suppose that $|M| + \mu(G[E_M]) = (\frac{1}{2} + \delta)\mu(G)$. Then, with high probability,*

$$\min\{\mu(G_L^r), \mu(G_R^r)\} \geq (1 - \frac{4}{\log n}) \cdot (\frac{1}{2} - \delta)\mu(G) \ .$$

Proof. We only give the argument for G_L^r, the argument for G_R^r is identical. Let $M_L^* = opt(G_L)$. We will show that most edges of M_L^* are included in $\pi(\frac{m}{\log n}, m]$ with high probability.

By Lemma 13, we have $|M_L^*| \geq (\frac{1}{2} - \delta)\mu(G)$. Let e_i be the i-th edge of M_L^*, let t_i be its position in the stream, and let Y_i be the indicator variable of the event "$t_i \leq \frac{m}{\log n}$". Our aim is to bound the probabilities $\mathbb{P}[Y_i = 1 \,|\, Y_1, \ldots, Y_{i-1}]$ and then apply the Chernoff bound stated in Theorem 2.

In the following, all our arguments are conditioned on the event "$|E(G[V \setminus V(M)])| = O(n \log^2 n)$" (without explicitly mentioning it), which we denote by E_1. This implies that

the algorithm does not abort in Line 3a. By the residual sparsity property as stated in Theorem 10, E_1 occurs with probability at least $1 - n^{-12}$.

We will argue now that

$$\mathbb{P}\left[\pi[\frac{m}{\log n}+1] \cup M \text{ is not a matching} \wedge \pi[\frac{m}{\log n}+1] \notin M_L^* \,|\, Y_1, \ldots, Y_{i-1}\right] \geq 1 - \frac{1}{\log^5 n}. \tag{2}$$

Since E_1 happens, observe that the second part of the stream consists of $m(1-\frac{1}{\log n}) - O(n\log^2 n)$ edges that cannot be added to matching M, at most $n/2$ edges of M_L^* (depending on the outcome of variables $Y_1, \ldots, Y_{i-1}$), and at most $O(n\log^2 n)$ edges that could extend M. Further, the arrival order of the edges $\pi(\frac{m}{\log n}, m]$ in the second part of the stream is uniform random, since the computed matching M is not affected by their order. Hence,

$$\mathbb{P}\left[\pi[\frac{m}{\log n}+1] \cup M \text{ is not a matching } \wedge \pi[\frac{m}{\log n}+1] \notin M_L^* \,|\, Y_1, \ldots, Y_{i-1}\right] \geq \frac{m(1-\frac{1}{\log n}) - O(n\log^2 n)}{m(1-\frac{1}{\log n})} \geq 1 - \frac{1}{\log^5 n},$$

using the assumption that the graph has at least $C \cdot n\log^{10} n$ edges, for a large enough C.

The key part of our argument is as follows: Let Π be the set of permutations that fulfill the event in Inequality 2. Given Π, we generate a set of permutations Π' with $\Pi' \supseteq \Pi$, which thus implies that the respective event is more likely to happen than the event in Inequality 2. Let $\pi \in \Pi$ be any permutation. Consider edge e_i and let j_i be such that $\pi[j_i] \in M$ is the edge incident to e_i. Since $e_i \in M_L^*$, we know that $t_i > j_i$. Construct now new permutations such that e_i is removed from its position t_i and is inserted at every position $\{t_i+1, t_i+2, \ldots, m\}$ and add the resulting permutations to Π'. Observe that for any permutation π' created this way, the exact same matching M is computed, which uses the fact that $\pi[\frac{m}{\log n}+1]$ cannot be added to M, which is important if e_i is inserted at a position larger than $\frac{m}{\log n}+1$. Observe further that the conditionings Y_j stay the same, which uses the fact that $\pi[\frac{m}{\log n}+1] \notin M_L^*$. Observe that Π' and Π are not identical, since we do not necessarily have that $\pi'[\frac{m}{\log n}+1] \cup M$ is not a matching for $\pi' \in \Pi'$. By construction, at least a $(1-\frac{1}{\log n})$-fraction of the permutations in Π' imply $Y_i = 0$. We thus obtain:

$$\begin{aligned}
&\mathbb{P}\left[Y_i = 0 \,|\, Y_1, \ldots, Y_{i-1}\right] \geq \\
&(1-\frac{1}{\log n}) \cdot \mathbb{P}\left[\pi[\frac{m}{\log n}+1] \cup M \text{ is not a matching} \wedge \pi[\frac{m}{\log n}+1] \notin M_L^* \,|\, Y_1, \ldots, Y_{i-1}\right] \\
&\quad \geq (1-\frac{1}{\log n})(1-\frac{1}{\log^5 n}) \geq 1 - \frac{2}{\log n}\ .
\end{aligned}$$

We now use the Chernoff bound for dependent variables stated in Theorem 2. Using $k = (\frac{1}{2}-\delta)\mu(G)$, we obtain (using $\mu = 2k/\log n$, and $\delta = 1$ in Theorem 2):

$$\mathbb{P}\left[\sum_{i=1}^{k} Y_i \geq 2\frac{2k}{\log n}\right] \leq \left(\frac{e}{4}\right)^{\frac{2k}{\log n}} \leq n^{-10}\ ,$$

using the assumption $\mu(G) = \Omega(n^{\frac{1}{4}})$. The result follows. ◀

In the remaining analysis, with the help of the previous lemma we bound the number of augmenting paths found in Lemma 15. We then conclude with our main theorem, where we show that one of the two computed matchings returned by the algorithm is necessarily large.

▶ **Lemma 15.** *Let $p = \Omega(1)$, suppose that $|M| + \mu(H) = (\frac{1}{2} + \delta)\mu(G)$, and let $|M| = \alpha\mu(G)$. Then, with high probability,*

$$|\mathcal{P}| \geq p\mu(G)\left(\frac{1-2\delta}{1+p} - \alpha\right) - o(\mu(G)) \ .$$

Proof. We follow the structure of the proof of Lemma 7. By an application of a Chernoff bound, we obtain $|M'| = p|M| \pm O(\sqrt{|M|\ln(|M|)})$, with probability at least $1 - |M|^{-C}$, for an arbitrarily large constant C. Next, by Theorem 6 and Lemma 14, with high probability in $\mu(G)$ we have

$$\min\{|M_L|, |M_R|\} \geq \frac{p}{1+p}(\frac{1}{2} - \delta)\mu(G) - o(\mu(G)) \ .$$

Since we assumed that $\mu(G) = \Omega(n^{\frac{1}{4}})$, this event also holds with high probability in n. As argued in the proof of Lemma 7, the quantity $|M_L| + |M_R| - |M'|$ bounds the number of 3-augmenting paths found, which then completes the proof:

$$\begin{aligned} |\mathcal{P}| &\geq |M_L| + |M_R| - |M'| \geq \frac{2p}{1+p}(\frac{1}{2} - \delta)\mu(G) - p|M| - o(\mu(G)) \\ &= p\mu(G)\left(\frac{2}{1+p}(\frac{1}{2} - \delta) - \alpha\right) - o(\mu(G)) = p\mu(G)\left(\frac{1-2\delta}{1+p} - \alpha\right) - o(\mu(G)) \ . \end{aligned}$$ ◀

▶ **Theorem 16.** *Setting $p = \sqrt{2} - 1$ in Algorithm 2 gives a one-pass random order semi-streaming algorithm for* MBM *with approximation ratio $\frac{1}{2} + \frac{2\sqrt{2}-3}{4\sqrt{2}-10} \geq 0.5390$ that succeeds with high probability.*

Proof. Suppose that $|M| = \alpha\mu(G)$ and $|M| + \mu(H) = (\frac{1}{2} + \delta)\mu(G)$. By Lemma 11, we have $\mu(H) \geq (1 - 2\alpha)\mu(G)$. Hence, $(1 - \alpha)\mu(G) \leq (\frac{1}{2} + \delta)\mu(G)$, which in turn implies $\alpha \geq \frac{1}{2} - \delta$. Plugging this into the bound given in Lemma 15, we obtain (ignoring the $o(\mu(G))$ term):

$$\begin{aligned} |M| + |\mathcal{P}| &\geq \alpha\mu(G) + p\mu(G)\left(\frac{1-2\delta}{1+p} - \alpha\right) = \mu(G)\left(\alpha(1-p) + p\left(\frac{1-2\delta}{1+p}\right)\right) \\ &\geq \mu(G)\left((\frac{1}{2} - \delta)(1-p) + p\left(\frac{1-2\delta}{1+p}\right)\right) \ . \end{aligned}$$

The quantity $|M| + \max\{|\mathcal{P}|, \mu(H)\}$, i.e., the size of the resulting matching, is minimizes if $|\mathcal{P}| = \mu(H)$. Hence, setting the right side of the previous inequality equal to $(\frac{1}{2} + \delta)\mu(G)$, we obtain $\delta = \frac{p(p-1)}{2p^2-6p-4}$, which is maximized for $p = \sqrt{2} - 1$ (observe that this is the same value as in the proof of Theorem 8). In this case, we obtain $\delta = \frac{2\sqrt{2}-3}{4\sqrt{2}-10} \approx 0.03950$, which completes the proof. ◀

6 Conclusion

In this paper, we gave a new method for finding a set of disjoint 3-augmenting paths that allows the augmentation of a maximal matching such that the resulting matching is of size at least $\sqrt{2} - 2$ times the size of a maximum matching. Our method is simple and only requires running the Greedy matching algorithm on a random subgraph. We applied this method in the data streaming setting and improved over the state-of-the-art one-pass random order algorithm and the state-of-the-art two- and three-pass adversarial order algorithms.

How large a matching can we compute in a single pass in the random order setting? All relevant known lower bounds for matchings [18, 22, 19] are highly sensitive to the arrival order of the edges and do not translate to the random order setting. Can we compute a

2/3-approximation in a single pass in the random order semi-streaming setting? In the adversarial order setting, it is known how to obtain a $2/3 - \delta$ approximation in $O(\frac{1}{\delta})$ passes. How many passes are required to obtain a 2/3-approximation?

References

1 Kook Jin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. Correlation clustering in data streams. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2237–2246, 2015. URL: http://jmlr.org/proceedings/papers/v37/ahn15.html.

2 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *Inf. Comput.*, 222:59–79, 2013. doi:10.1016/j.ic.2012.10.006.

3 Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. *CoRR*, abs/1711.03076, 2017. arXiv:1711.03076.

4 Sepelir Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 1345–1364, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=2884435.2884528.

5 Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tohoku Math. J. (2)*, 19(3):357–367, 1967. doi:10.2748/tmj/1178243286.

6 Marc Bury and Chris Schwiegelshohn. Sublinear estimation of weighted matchings in dynamic data streams. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 263–274, 2015. doi:10.1007/978-3-662-48350-3_23.

7 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 1326–1344, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=2884435.2884527.

8 Graham Cormode, Hossein Jowhari, Morteza Monemizadeh, and S. Muthukrishnan. The Sparse Awakens: Streaming Algorithms for Matching Size Estimation in Sparse Graphs. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2017.29.

9 Michael Crouch and Daniel M. Stubbs. Improved Streaming Algorithms for Weighted Matching, via Unweighted Matching. In Klaus Jansen, José D. P. Rolim, Nikhil R. Devanur, and Cristopher Moore, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*, volume 28 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 96–104, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.96.

10 Michael S. Crouch, Andrew McGregor, and Daniel Stubbs. Dynamic graphs in the sliding-window model. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms – ESA 2013*, pages 337–348, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

11 Jack Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, pages 449–467, 1965.

12 Sebastian Eggert, Lasse Kliemann, Peter Munstermann, and Anand Srivastav. Bipartite matching in the semi-streaming model. *Algorithmica*, 63(1):490–508, Jun 2012. doi:10.1007/s00453-011-9556-8.

13 Hossein Esfandiari, Mohammad T. Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof Onak. Streaming algorithms for estimating the matching size in planar graphs and beyond. In *Proceedings of the Twenty-sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, pages 1217–1233, Philadelphia, PA, USA, 2015. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=2722129.2722210.

14 Hossein Esfandiari, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. Finding large matchings in semi-streaming. In *IEEE International Conference on Data Mining Workshops, ICDM Workshops 2016, December 12-15, 2016, Barcelona, Spain.*, pages 608–614, 2016. doi:10.1109/ICDMW.2016.0092.

15 Alexander Fanghänel, Thomas Kesselheim, and Berthold Vöcking. Improved algorithms for latency minimization in wireless networks. *Theor. Comput. Sci.*, 412(24):2657–2667, 2011. doi:10.1016/j.tcs.2010.05.004.

16 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2):207–216, 2005. doi:10.1016/j.tcs.2005.09.013.

17 Mohsen Ghaffari, Themis Gouleakis, Slobodan Mitrovic, and Ronitt Rubinfeld. Improved massively parallel computation algorithms for mis, matching, and vertex cover. *CoRR*, abs/1802.08237, 2018. arXiv:1802.08237.

18 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 468–485, 2012. URL: http://portal.acm.org/citation.cfm?id=2095157&CFID=63838676&CFTOKEN=79617016.

19 Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. *Algorithmica*, 76(3):654–683, 2016. doi:10.1007/s00453-016-0138-7.

20 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi:10.1137/0202019.

21 Sagar Kale and Sumedh Tirodkar. Maximum matching in two, three, and a few more passes over graph streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, pages 15:1–15:21, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.15.

22 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the Twenty-fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 1679–1697, Philadelphia, PA, USA, 2013. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=2627817.2627938.

23 Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 734–751, Philadelphia, PA, USA, 2014. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=2634074.2634129.

24 Christian Konrad. Maximum matching in turnstile streams. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015*, pages 840–852, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

25 Christian Konrad. MIS in the congested clique model in $O(\log\log\Delta)$ rounds. *CoRR*, abs/1802.07647, 2018. arXiv:1802.07647.

26 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pages 231–242, 2012. doi:10.1007/978-3-642-32512-0_20.

27 Christian Konrad and Adi Rosén. Approximating semi-matchings in streaming and in two-party communication. *ACM Trans. Algorithms*, 12(3):32:1–32:21, 2016. doi:10.1145/2898960.

28 Colin McDiarmid. On the method of bounded differences. In *Surveys in Combinatorics 1989*. Cambridge University Press, Cambridge, 1989.

29 Andrew McGregor. Finding graph matchings in data streams. In Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 170–181, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

30 Andrew McGregor and Sofya Vorotnikova. A Simple, Space-Efficient, Streaming Algorithm for Matchings in Low Arboricity Graphs. In Raimund Seidel, editor, *1st Symposium on Simplicity in Algorithms (SOSA 2018)*, volume 61 of *OpenAccess Series in Informatics (OASIcs)*, pages 14:1–14:4, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.SOSA.2018.14.

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

32 Xiaoming Sun and David P. Woodruff. Tight Bounds for Graph Problems in Insertion Streams. In Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*, volume 40 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 435–448, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.435.

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

Reconfiguration of Graph Minors

Benjamin Moore
Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Canada
brmoore@uwaterloo.ca

Naomi Nishimura
David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada
nishi@uwaterloo.ca

Vijay Subramanya
David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada
v7subram@uwaterloo.ca

Abstract

Under the reconfiguration framework, we consider the various ways that a target graph H is a *minor* of a host graph G, where a subgraph of G can be transformed into H by means of *edge contraction* (replacement of both endpoints of an edge by a new vertex adjacent to any vertex adjacent to either endpoint). Equivalently, an *H-model* of G is a labeling of the vertices of G with the vertices of H, where the contraction of all edges between identically-labeled vertices results in a graph containing representations of all edges in H.

We explore the properties of G and H that result in a connected *reconfiguration graph*, in which nodes represent H-models and two nodes are adjacent if their corresponding H-models differ by the label of a single vertex of G. Various operations on G or H are shown to preserve connectivity. In addition, we demonstrate properties of graphs G that result in connectivity for the target graphs K_2, K_3, and K_4, including a full characterization of graphs G that result in connectivity for K_2-models, as well as the relationship between connectivity of G and other H-models.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases reconfiguration, graph minors, graph algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.75

Related Version A full version of the paper is available at `https://arxiv.org/pdf/1804.09240.pdf`.

Funding Supported by the Natural Sciences and Engineering Research Council of Canada

Acknowledgements We wish to thank Tesshu Hanaka, Venkatesh Raman, Krishna Vaidyanathan, and Marcin Wrochna for helpful discussions.

1 Introduction

Graph minors have been studied extensively as a means for categorizing graphs and exploiting their properties. A graph H is a *minor* of a graph G if H can be formed from a subgraph of G by a series of edge contractions, where the *contraction* of the edge uv results in the replacement of both u and v by a new vertex w that is adjacent to any vertex that was adjacent to u or v (or both). Much of the research in the area has focused on classes of graphs that are closed under the taking of minors, and on exploiting properties of graphs known not to contain certain graphs as minors. For example, it is known that for every minor

closed class, that class is characterized by a finite set of forbidden minors [12]. Additionally, it has been shown that for any fixed graph H, every H-minor-free graph of treewidth w has an $\Omega(w) \times \Omega(w)$ grid as a minor [2].

In our work, we instead focus on the solution space of H-models of a graph G using the reconfiguration framework [6, 11, 14], where an H-model is a mapping that labels the vertices of G with the vertices of H. A *reconfiguration graph* for an instance of a problem consists of a node for each possible feasible solution and an edge between any two nodes representing solutions that are adjacent. The definition of adjacency may be presented as a *reconfiguration step* used to transform a solution into a neighbouring solution. Structural properties of the reconfiguration graph, including its diameter and whether it is connected, are of interest both in their own right and as keys to solving algorithmic problems, such as determining whether there is a path (or *reconfiguration sequence*) in the graph between two given vertices and, if so, finding the shortest such path.

The reconfiguration framework has been used to explore the relationships among solutions to instances of many graph problems, such as Independent Set, Shortest Path, and k-Colouring. Practical uses of such work include determining whether and how it is possible to make incremental changes to form one solution from another, ensuring that at each step along the way, the intermediate configuration is also a solution. The study of minor reconfiguration is a natural extension of work that determines conditions under which reconfiguration graphs are guaranteed to be connected, as well as recent work on mappings between graphs, in which each node in the reconfiguration graph represents a subgraph in a specified class [5, 10]. In addition, our results widen the scope of work in which configurations are represented as labelings, which previously had been limited to problems entailing moving labels from a source to a target configuration using the minimum number of exchanges of labels on adjacent vertices (detailed in a survey of reconfiguration [11]), and the use of labeled edges in the reconfiguring of triangulations [8].

In this paper, we consider how the connectivity of the reconfiguration graph depends on the choices of the host G and target H. We consider an instance of Minor Reconfiguration to consist of a *host graph* G and *target graph* H such that H is a minor of G. Each node in the reconfiguration graph consists of a labeling of the vertices of G with the vertices of H (or, more simply, integers in $\{1, \ldots, |V(H)|\}$) such that the contraction of each edge with identically-labeled endpoints results in a graph that, upon deletion of zero or more edges, yields H. We consider two H-models to be adjacent if they differ by a single label.

We begin by establishing properties of k-connected graphs and minors in Section 2, based on which we form a toolkit of techniques used in reconfiguration (Section 3). We consider various properties of G and H that determine whether or not the reconfiguration graph is connected. For a target graph H, we define host(H) to be the set of host graphs G such that the reconfiguration graph for G and H is connected (Definition 8). We then focus on characterizing host(K_2) (Section 4), host(K_3) (Section 5), and host(K_4) (Section 6). Finally, in Section 7, we summarize the results and present directions for future work.

2 Preliminaries

We define key terms used in the description of graphs; for common terms not defined in this paper, the reader is referred to a resource on graph theory [3]. We will frequently focus on subsets of the vertices; for a subgraph $V \subseteq V(G)$, the *induced subgraph* $G[V]$ is the subgraph with vertex set V and edge set $\{uv \in V(G) \mid u, v \in V\}$. As shorthand, for G a graph and S a set of vertices, we use $G \setminus S$ to denote $G[V(G) \setminus S]$. In order to avoid confusion with the vertices in graphs G and H, we refer to the *nodes* of a reconfiguration graph.

2.1 Properties of k-connected graphs

We focus on various ways of connecting vertices in the graph. A *cut set* S of G is a set of vertices such that $G \setminus S$ consists of at least two components; the member of a cut set of size one is also called a *cut vertex*. A *bridge* is an edge whose deletion disconnects the graph. A graph is k-connected if there is no cut set of size k. Equivalently, in a k-connected graph there exist k vertex-disjoint paths between any pair of vertices in the graph. At times we will focus on how highly connected a specific vertex might be. A *universal vertex* is adjacent to all other vertices in the graph. In a *complete graph* on j vertices, denoted K_j, all vertices are universal vertices.

To characterize the behaviour of various host and target graphs, we make use of characterizations of graphs in terms of a base graph class and a series of operations. *Adding an edge* consists of adding an edge between two vertices in $V(G)$. To *split* a vertex v is to first delete v from G, and then add two vertices v_1 and v_2 to G such that $v_1v_2 \in E(G)$, each neighbour of v in G is a neighbour of exactly one of v_1 or v_2, and $\deg(v_i) \geq 3$ for $i \in \{1, 2\}$.

We make use of Tutte's characterization of 3-connected graphs and Ding and Qin's characterization of a subset of the 4-connected graphs, given below. The base case for Tutte's characterization is a wheel, defined as follows.

▶ **Definition 1.** A *k-wheel* W_k is a graph on $k + 1$ vertices, the *rim vertices* $r_1, \ldots, r_k$ and the *hub vertex* h, where there is a cycle induced on the rim vertices and an edge between h and each rim vertex.

▶ **Theorem 2.** *[13] A graph is 3-connected if and only if it is obtained from a wheel by repeatedly adding edges and splitting vertices.*

To state Ding and Qin's result, we need a few additional definitions. The *line graph* $L(G)$ of a graph G has a vertex corresponding to each edge of G and two vertices are adjacent if their corresponding edges share an endpoint in G. A graph is *cubic* if each vertex has degree three. Furthermore, a cubic graph with at least six vertices is *internally 4-connected* if its line graph is 4-connected. One of the base classes for their characterization is a square of a cycle, as defined below.

▶ **Definition 3.** The *square of a cycle* C_k^2 is formed from the cycle C_k by adding an edge between any pair of vertices joined by a path of length two.

Finally, we say a sequence of 4-connected graphs $G_1, \ldots, G_n$ form a (G_1, G_n)*-chain* if for all $i \in \{1, \ldots, n-1\}$, there exists an edge e such that G_{i+1} is formed from G_i by removal of the edge e. Theorem 4 is a generalization of a well-known theorem of Martinov [9].

▶ **Theorem 4.** *[4] Let $\mathcal{C} = \{C_k^2 : k \geq 5\}$ and $\mathcal{L} = \{H : H$ is the line graph of an internally 4-connected cubic graph$\}$. Let G be a 4-connected graph not in $\mathcal{C} \cup \mathcal{L}$. Then if G is planar, there is a (G, C_6^2)-chain. Otherwise, there is a (G, K_5)-chain.*

2.2 Branch sets, H-models, host(H), and block trees

For the purposes of reconfiguration, we make use of an equivalent definition of a minor as a mapping of each vertex of host graph G to a vertex of target graph H. For convenience, we sometimes represent the vertices of H as integer labels.

▶ **Definition 5.** For graphs G and H and mapping $f : V(G) \to V(H)$, we refer to $f(v)$ as the *label* of v and define the *branch set* $G(f, i)$ to be the subgraph of G induced on the set of vertices with label i.

For ease, we will make use of $|G(f,i)|$ to denote $|V(G(f,i))|$. Given a mapping between $V(G)$ and $V(H)$, an edge of G is a *connecting edge* if its endpoints are members of two different branch sets, and we say that it *connects* those two branch sets. A mapping is equivalent to a minor when two additional properties hold, as indicated in Definition 6.

▶ **Definition 6.** For graphs G and H and mapping $f : V(G) \to V(H)$, we say that H is a *minor of* G and that f is an *H-model of G* if the following conditions hold:
1. for each $i \in V(H)$, each branch set $G(f,i)$ is nonempty and connected, and
2. for each edge $ij \in E(H)$, there exists an edge in $E(G)$ connecting $G(f,i)$ and $G(f,j)$.

▶ **Definition 7.** For any graphs G and H such that H is a minor of G, the *branch set reconfiguration graph* of G and H consists of a node for each H-model of G and, for each pair of H-models f and g, an edge between the nodes corresponding to f and g if and only if there exists a $v \in V(G)$ such that $f(v) \neq g(v)$ and for all $u \neq v$, $f(u) = g(u)$.

▶ **Definition 8.** For any graphs G and H such that H is a minor of G, $G \in \text{host}(H)$ if and only if the branch set reconfiguration graph of G and H is connected.

We will often find it convenient to view each branch set in terms of the tree structure of its 2-connected components. A *block* of a connected graph is either a maximal 2-connected subgraph or one of the endpoints of a bridge. The *block tree* of a connected graph consists of a node for each block B; there is an edge between the nodes corresponding to blocks B and B' if there exists a cut vertex v of G such that $V(B) \cap V(B') = \{v\}$. Given a graph G, an H-model f, and a label a, we use $T(G, f, a)$ to denote the block tree for $G(f,a)$. In addition, for a subgraph A of G, we use $T(G, f, a, A)$ to denote the subtree of $T(G, f, a)$ induced by the blocks containing vertices in $V(G(f,a)) \cap V(A)$. For convenience, we sometimes use "block of $G(f,a)$" to refer to a block of $T(G, f, a)$.

To make use of the tree structure, our algorithms typically process a block tree starting with blocks that are leaves of their block trees, or *leaf blocks*; a branch set that is 2-connected can be viewed as having a block tree consisting of a single leaf block. For ease of description, we will refer to the cut vertices of G that appear in multiple blocks as *joining vertices* and all other vertices as *interior vertices*.

2.3 Essential edges, crucial vertices, weak connections, and lynchpins

When considering how labelings can be reconfigured, we need to ensure that we retain the connecting edges as required in Definition 6. In doing so, we need to pay particular attention to vertices and edges whose relabeling will cause problems.

When there exists only a single edge that connects a pair of branch sets with labels a and b, $ab \in E(H)$, we call such an edge an *essential edge*, and denote it as $\text{ess}(a,b)$. If all the edges between branch sets with labels a and b have the same endpoint in a, we call that vertex an *essential vertex for b*; clearly every endpoint of an essential edge is an essential vertex, but not every essential vertex is the endpoint of an essential edge.

The presence of essential vertices will be important in determining when it is easy to relabel vertices. For any two labels, if the branch set with label a contains an essential vertex for b or if the branch set with label b contains an essential vertex for a, we will say that the branch sets with labels a and b are *weakly connected*, or form a *weak connection*.

Our results rely on the interplay between the presence of weak connections and the connectivity of a graph. For each weak connection, we identify a vertex as the *lynchpin* for the connection. When the branch sets with labels a and b are weakly connected by an essential edge, then either of the endpoints of the essential edge can be designated as the

lynchpin. Otherwise, the (single) essential vertex giving rise to the weak connection is the lynchpin for that connection. We will use lynchpins to form cut sets between non-lynchpins and other branch sets.

A vertex v with label a is a *crucial vertex* if it is an essential vertex for b and an essential vertex for c, for $b \neq c$, and a *non-crucial vertex* otherwise. If for some distinct labels a, b, and c, a vertex $v \in G(f, a)$ is essential for c and also has at least one neighbour in $G(f, b)$, then v is a *b-crucial vertex*. Clearly, a vertex in $G(f, a)$ that is essential for b and c is crucial, b-crucial, and c-crucial.

2.4 Properties of H-models of k-connected graphs

When G is k-connected, we are able to establish properties of connecting edges of branch sets, as shown in Lemma 9 and Lemma 10, as well as the structure of weak edges (Lemma 11). The results make use of the fact that in a k-connected graph there cannot be a cut set of size less than k separating any two vertices; cut sets are typically formed from the joining vertices of leaf blocks and lynchpins, and vertices separated by cut sets are typically non-lynchpins. Proofs of results marked with (*) have been omitted due to space limitations.

▶ **Lemma 9 (*).** *Given a k-connected graph G and an H-model f of G for some graph H, for any branch set $G(f, a)$, each leaf block has $k - 1$ interior vertices that are endpoints of connecting edges.*

▶ **Lemma 10.** *Given a k-connected graph G and an H-model f of G, where $|V(H)| = k$, suppose there exist branch sets $G(f, \ell)$ and $G(f, m)$ such that $\ell m \in E(H)$ and there are weak connections between $G(f, \ell)$ and each branch set other than $G(f, m)$ (where a weak connection between $G(f, \ell)$ and $G(f, m)$ is possible but not required). Then, the following hold:*

1. *Each leaf block in $G(f, \ell)$ must contain an interior vertex that is the endpoint of a connecting edge to $G(f, m)$.*
2. *If it is possible to designate lynchpins of the weak connections such that $G(f, \ell)$ contains a non-lynchpin, then each leaf block in $G(f, m)$ must contain an interior vertex that is the endpoint of a connecting edge to $G(f, \ell)$.*

Proof. To see why the first point holds, suppose instead that no such interior vertex existed in a leaf block of $G(f, \ell)$. Then, each path between an interior vertex u in the leaf block in $G(f, \ell)$ and any vertex v in $G(f, m)$ must pass through either the joining vertex of the leaf block or one of the lynchpins for the weak connections. However, u and v are thus separated by a cut set of size at most $k - 1$, contradicting the k-connectivity of G.

The argument for the second point is similar; we can show that the joining vertex of the leaf block in $G(f, m)$ and the lynchpins of the weak connections form a cut set of size at most $k - 1$ separating any interior vertex in the leaf block and the non-lynchpin in $G(f, \ell)$. ◀

▶ **Lemma 11 (*).** *Given a k-connected graph G and a K_k-model f of G such that there is a branch set B with weak connections to all other branch sets, it is not possible to designate lynchpins such that B contains at least one vertex x that is not a lynchpin for any of the weak connections, and at least one other branch set contains a vertex y that is not a lynchpin for any of the weak connections.*

3 Toolkit for reconfiguration of minors

In this section, we introduce techniques and properties that are exploited in the results found in the rest of the paper. In particular, we focus on the types of steps used in reconfiguration and the properties that need to be satisfied for each type of transformation. In Lemmas 12

and 13 we determine conditions under which a vertex can be relabeled in a single step. In the remainder of the section, we present results that can be used to handle more complex situations in which one or more of the conditions do not hold.

Lemma 12 delineates the conditions necessary for a vertex to be able to be relabeled from a to b in a single reconfiguration step: it cannot be the only vertex with label a, it cannot be a cut vertex in its branch set, it must be connected to a vertex with label b, and it is not incident with every edge between the branch sets for labels a and c, where $c \neq b$.

▶ **Lemma 12 (*).** *Given a graph G and an H-model f of G, a vertex v can be relabeled from a to b in a single reconfiguration step if and only if the following conditions hold:*

1. $|G(f, a)| > 1$;
2. *v is not a cut vertex in $G(f, a)$;*
3. *v has at least one neighbour in $G(f, b)$; and*
4. *v is not a b-crucial vertex.*

Because several of the conditions hold automatically for a universal vertex, the following lemma lists a smaller number of conditions:

▶ **Lemma 13 (*).** *Given a graph G and an H-model f of G, a vertex v can be relabeled from a to b in a single reconfiguration step if the following conditions hold:*

1. *$G(f, a)$ contains a universal vertex u such that $u \neq v$; and*
2. *v has at least one neighbour in $G(f, b)$.*

When neither Lemma 12 nor Lemma 13 applies, the relabeling of a vertex requires a series of reconfiguration steps. When the vertex to be relabeled is the only member of its branch set or a crucial vertex, we first need to fill its branch set with new vertices that can provide the necessary connecting edges to other branch sets. When the vertex to be relabeled is a cut vertex, we will need to siphon away vertices from its branch set so that it is no longer a cut vertex among the remaining vertices with its label.

When a branch set is a block tree, both filling and siphoning entail the relabeling of vertices in a branch set block by block, starting at the leaf blocks. If we are able to relabel all the interior vertices of a leaf block, we can simplify the block tree by removing the leaf block. We will show in Lemma 20 that such relabeling is possible as long as we can avoid certain bad situations involving *leaf-crucial models* and *leaf-ℓ-crucial models*, as outlined in Definitions 14, 15, and 16.

▶ **Definition 14.** Given a graph G and an H-model f of G, we say that a vertex v is a *leaf-crucial vertex* if v is a crucial vertex that is an interior vertex in a leaf block in its branch set. An H-model that contains a leaf-crucial vertex is a *leaf-crucial model*.

▶ **Definition 15.** Given a graph G and an H-model f of G, we say that a vertex v is a *leaf-ℓ-crucial vertex* if v is an ℓ-crucial vertex that is an interior vertex in a leaf block in its branch set. An H-model that contains a leaf-ℓ-crucial vertex is a *leaf-ℓ-crucial model*.

▶ **Definition 16.** Given a graph G, an H-model f of G, a label a, and a subgraph A of $G(f, a)$, we say that *f hits a leaf-crucial model on relabeling A* if any relabeling of f that changes only the vertices of A can be extended by relabeling only the vertices of A to reach a leaf-crucial model, and that *f hits a leaf-ℓ-crucial model on relabeling A* if any relabeling of f that changes only the vertices in A can be extended by relabeling only the vertices of A to reach a leaf-ℓ-crucial model.

▶ **Observation 17.** *Given a graph G, an H-model f of G, a label a, and a subgraph A of $G(f,a)$, if f does not hit a leaf-ℓ-crucial model on relabeling A, then for each model g reachable from f by relabeling only the vertices of A, no interior vertex in a leaf block of $T(G,g,a)$ is ℓ-crucial.*

In Lemma 18, we show that each time we relabel an interior vertex in a leaf block, if the leaf block still has interior vertices, one will be a neighbour of the relabeled vertex. We use the result in Lemma 19 to show that if we can avoid leaf-crucial models, then it is possible to relabel all the interior vertices in a leaf block (and hence remove it from the branch set). By repeatedly relabeling leaf blocks, an entire connected component can be siphoned away, as shown in Lemma 20.

▶ **Lemma 18.** *Given a graph G and an H-model f of G, suppose there exist labels a and b and a vertex v such that $|G(f,a)| \geq 2$, v is in a leaf block L of $T(G,f,a)$, and relabeling v to b (and no other vertices) results in another H-model g. Then v has a neighbour in $G(g,a)$, and if $|V(L) \setminus \{v\}| \geq 2$, then v has a neighbour u in $G(g,a)$ such that $u \in V(L)$ and u is an interior vertex in a leaf block of $T(G,g,a)$.*

Proof. We observe that since Lemma 12 holds for the relabeling of v from a to b, $|G(f,a)| \geq 2$, and since each branch set is connected, v must then have a neighbour in $G(g,a)$.

We now suppose that $|V(L) \setminus \{v\}| \geq 2$. At least one neighbour $u \in V(L)$ of v is in a leaf block L_g of $T(G,g,a)$, as v is not a cut vertex of $G(f,a)$ (by Lemma 12, condition 2) and L is a 2-connected leaf block of $T(G,f,a)$. If u is an interior vertex in $G(g,a)$, we are done. If instead u is a joining vertex and there exists no interior vertex in L_g that is a neighbour of v, then for each interior vertex $w \in V(L_g)$, u lies on every path from v to w in L, which contradicts the fact that L is 2-connected. Hence, v is adjacent to an interior vertex in $G(g,a)$. ◀

▶ **Lemma 19 (*).** *Given a graph G, an H-model f of G, a label a, and a leaf block L of $T(G,f,a)$, suppose that $|V(G(f,a)) \setminus V(L)| \geq 1$, L contains at least one interior vertex that is the endpoint of a connecting edge, and f does not hit a leaf-crucial model on relabeling L. Then we can reconfigure f to a model g such that $g(v) \neq f(v)$ for each $v \in V(L)$ that is an interior vertex of $G(f,a)$ and $g(u) = f(u)$ for all other vertices u.*

▶ **Lemma 20.** *Given a 2-connected graph G and an H-model f of G, suppose there exist $ab \in E(H)$, a cut vertex x of $G(f,a)$, and a connected component C of $G(f,a) \setminus \{x\}$ that contains at least one vertex with a neighbour in $G(f,b)$ such that f does not hit a leaf-crucial model or a leaf-b-crucial model on relabeling C. Then we can reconfigure f to a model g such that $g(v) \neq f(v)$ for each $v \in V(C)$, $g(u) = f(u)$ for all $u \notin V(C)$, and x has a neighbour in $G(g,b)$.*

Proof. We use B to denote the block of $T(G,f,a,C)$ containing x, and view $T(G,f,a,C)$ as rooted at B. We observe that any leaf block of $T(G,f,a,C)$ is also a leaf block of $T(G,f,a)$.

To reconfigure f to g, we work up the tree $T(G,f,a,C)$ from leaf blocks up to B, at each step relabeling all the vertices in the current block with labels different from a. Specifically, if a leaf block does not have an interior vertex with a neighbour labeled b, then in Case 1 below, we can relabel the vertices in the block; such a relabeling removes the block from the branch set for label a. If instead a leaf block does have an interior vertex with a neighbour labeled b, then in Case 2 below, we can relabel the block with b. Such a relabeling not only removes the block from the branch set for label a, but also ensures that the joining vertex of the block has a neighbour with label b. Repeated applications of the two cases suffice

to ensure that we eventually reach a point in the process at which B is a leaf block and contains an interior vertex with a neighbour labeled b; using Case 2, we can then satisfy the statement of the lemma by ensuring that every vertex in B except x receives label b.

Case 1: A leaf block L of $T(G, f, a, C)$ does not contain an interior vertex with a neighbour in $G(f, b)$.

Since G is 2-connected, by Lemma 9, L has at least one interior vertex that is an endpoint of a connecting edge. Because f does not hit a leaf-crucial model and $|V(G(f, a)) \setminus V(L)| \geq 1$, by Lemma 19, we can relabel the interior vertices of L.

Case 2: A leaf block L of $T(G, f, a, C)$ contains an interior vertex v with a neighbour in $G(f, b)$.

We first observe that we can relabel v to b: since f does not hit a leaf-b-crucial model on relabeling C, v is not b-crucial (Observation 17), and hence all the conditions of Lemma 12 hold. We can then repeat the same argument on the resulting model h, as follows. For L_h the leaf block of $T(G, h, a, C)$ such that $V(L_h) = V(L) \setminus \{v\}$, if one exists, by Lemma 18, L_h contains an interior vertex u that is a neighbour of v. We can then use the fact that f does not hit a leaf-b-crucial model on relabeling C to again apply Observation 17 to conclude that u is not b-crucial and that Lemma 12 is satisfied for the labeling of u to b. Further repetitions of the argument result in the relabeling of all interior vertices of L to b. ◀

4 Characterizing host(K_2)

Theorem 21 fully characterizes host(K_2); as a consequence, we can use membership in host(K_2) as an alternate definition of 2-connectivity. The reconfiguration of a 2-connected graph G is achieved by defining a canonical model (one in which one vertex has one label and all other vertices have the other label) and then showing it is possible both to reconfigure any K_2-model to a canonical model and to reconfigure between canonical models. In contrast, when G is not 2-connected, the presence of a cut vertex prevents reconfiguration, as no ordering of relabeling steps can prevent a branch set from being disconnected.

▶ **Theorem 21 (*).** *$G \in$ host(K_2) if and only if G is 2-connected.*

5 Characterizing host(K_3)

To show that every 3-connected graph is in host(K_3), we make use of Tutte's characterization in Theorem 2. In order to prove Theorem 22, it suffices to show that wheels are in host(K_3) (Corollary 24) and that connectivity is preserved under the splitting of vertices (Lemma 27) and adding of edges (Lemma 30).

▶ **Theorem 22 (*).** *Every 3-connected graph is in host(K_3).*

The result for wheels (Corollary 24) follows from a result on a generalization of wheels by allowing multiple hub vertices, each of which is a universal vertex, and replacing each rim vertex by a connected graph. To obtain a more general result, we use $W(G_1, G_2, \ldots, G_m, n, \ell, m)$ to denote a generalized wheel, for each G_i a connected graph on n vertices, $V(G_i) = \{v_{(i,1)} \ldots v_{(i,n)}\}$, and ℓ and m both positive integers. The graph $W(G_1, G_2, \ldots, G_m, n, \ell, m)$ consists of ℓ *hub vertices*, $V_H = \{h_1, \ldots, h_\ell\}$, and mn *subgraph vertices*, $V_S = \{s_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq n\}$, where $s_{i,j}$ corresponds to $v_{(i,j)}$. The edge set consists of the *hub edges*,

$E_H = \{h_i h_j \mid 1 \leq i \leq \ell, 1 \leq j \leq \ell, i \neq j\}$, the *subgraph edges*, $E_s = \{s_{i,j}s_{i,k} \mid v_{(i,j)}v_{(i,k)} \in E(G_i), 1 \leq i \leq m\}$, the *rim edges*, $E_R = \{s_{(i,j)}s_{(k,j)} \mid k \equiv i+1 \bmod m, 1 \leq j \leq n\}$, and the *connecting edges*, $E_C = \{h_k s_{(i,j)} \mid 1 \leq k \leq \ell, 1 \leq i \leq m, 1 \leq j \leq n\}$. Observe that $W(G_1, G_2, \ldots, G_m, n, l, m)$ has a K_{l+2}-minor when $m \geq 3$.

▶ **Lemma 23 (*).** *For any graphs G_i, $W(G_1, G_2, \ldots, G_m, n, \ell, m)$ is in host$(K_{\ell+2})$ for any $m \geq 3$.*

▶ **Corollary 24.** *W_n is in host(K_3).*

We consider the splitting of vertices in two steps. In Lemma 25, which applies more generally to K_k for any $k > 2$, we show that we can reconfigure between models in which the vertices resulting from the split have the same label. Then, in Lemma 27, which uses Lemma 26, we consider cases in which the vertices can have different labels.

▶ **Lemma 25.** *Let G be a 2-connected graph and G' be formed from G by splitting a vertex v into vertices x and y. For any $k > 2$, let f and g be K_k-models of G and f' and g' be K_k models of G' such that $f(v) = f'(x) = f'(y)$, $g(v) = g'(x) = g'(y)$ and for all $u \in V(G) \setminus \{v\}$, $f(u) = f'(u)$ and $g(u) = g'(u)$. If f and g are reconfigurable, then f' and g' are reconfigurable.*

Proof. Since f and g are reconfigurable, there is a reconfiguration sequence $\sigma = f = f_1, \ldots, f_\ell = g$ for some value of ℓ. Using σ, we wish to form a reconfiguration sequence σ' from f' to g' in the reconfiguration graph for G'. In forming the sequence, we observe that if there is a prefix τ of σ such that v is not relabeled in any of the steps, then we can form a prefix τ' of σ' by executing the same sequence of steps.

We now consider the first relabeling of v in σ, say from f_j to f_{j+1}; we wish to show that in σ', we can relabel both x and y in the same way. Without loss of generality, we assume that $f_j(v) = f'_j(x) = f'_j(y) = a$ and that $f_{j+1}(v) = b$.

We can use the fact that Lemma 12 holds for the labeling of v by b to establish useful properties of x and y. Because v is not in a branch set of size one (condition 1), x and y are not the only two vertices in $G'(f'_j, a)$. Since v is a not a cut vertex of its branch set (condition 2), x and y together cannot form a cut set. As v has a neighbour with label b (condition 3), either x or y (or both) must have a neighbour with label b under f'_j. Finally, because v is not a b-crucial vertex (condition 4), there must exist some vertex other than x or y in $G'(f'_j, a)$ that has a neighbour with label b.

Without loss of generality, we assume that x has a neighbour with label b, and consider two cases, depending on whether or not x is a joining vertex in the block tree of $G'(f'_j, a)$.

Case 1: x is a not a joining vertex of the block tree of $G'(f'_j, a)$

By our observations above, all conditions of Lemma 12 hold for the relabeling of x to b. As a consequence of the relabeling, y now has a neighbour with label b. Because v was not a cut vertex in $G(f_j, a)$, the removal of both x and y cannot disconnect $G'(f'_j, a)$, and hence condition 2 holds for y. As the remaining conditions of Lemma 12 were established in the argument above, we can now relabel y to b, as needed.

Case 2: x is a joining vertex in the block tree of $G'(f'_j, a)$

We first show that the component C of $G'(f'_j, a)$ containing y consists solely of the vertex y. If instead there existed another vertex in C, then the removal of both x and y would separate the vertex from the other components of $G'(f'_j, a) \setminus \{x\}$, and consequently v would be a cut vertex in $G(f_j, a)$, which contradicts Lemma 12 for the relabeling of v to b.

By the definition of the split operation, $\deg(y) \geq 3$; because x is y's only neighbour with label a, y must have at least one neighbour with label b or c, for some $c \notin \{a, b\}$. If y has a neighbour with label b, we can use Case 1 to complete the relabeling of y and then x. Otherwise, we will show that we can first relabel y to c, relabel x to b, and finally relabel y to b. In each case, we show that we can use Lemma 12.

To see that we can relabel y to c, it suffices to observe that y is not c-crucial, as x has a neighbour in b, and if y were essential for some $d \notin \{b, c\}$, then v would be essential for d in f_j, and hence b-crucial in f_j, which is a contradiction. Now, since x is now no longer a cut vertex, we can relabel x to b. Finally, since y now has a neighbour with label b (that is, x), we can relabel y with b, as needed. ◀

▶ **Lemma 26.** *Given a 3-connected graph G and a K_3-model f of G, suppose there exists a cut vertex x of $G(f, a)$ with a neighbour in $G(f, b)$ such that x is not essential for c. Then there exists a component D of $G(f, a) \setminus \{x\}$ such that we can reconfigure f to a model g in which $g(v) = f(v)$ for each $v \in V(D)$, $g(x) = b$, and $g(u) \neq f(u)$ for all other vertices of $G(f, a)$.*

Proof. We form model g by first siphoning all components of $G(f, a) \setminus \{x\}$ except D out of $G(f, a)$ and then by relabeling x to b.

Because x is not essential for c, there must exist at least one component of $G(f, a) \setminus \{x\}$ that contains a vertex with a neighbour in $G(f, c)$; we choose D to be one such component.

By Lemma 9, each other component C must have a neighbour in either $G(f, b)$ or $G(f, c)$. Since x has a neighbour in $G(f, b)$ and D has a neighbour in $G(f, c)$, f does not hit a leaf-crucial model, a leaf-b-crucial model, or a leaf-c-crucial model on relabeling C, and hence we can use Lemma 20 to relabel all vertices of C with either b or c.

After the vertices of every component except D have been relabeled, all the conditions for Lemma 12 now hold for the relabeling of x to b: the branch set for label a has at least one vertex other than x, x is no longer a cut vertex, x has a neighbour labeled b, and a vertex in D has a neighbour labeled c. ◀

▶ **Lemma 27.** *Suppose G is a 3-connected graph such that $G \in host(K_3)$ and G' is a graph formed from G by splitting a vertex v into vertices x and y. Then G' is in $host(K_3)$.*

Proof. We show that for any source and target K_3-models of G', we can find a reconfiguration sequence from the source to the target. We know from Lemma 25 that we can reconfigure between any two K_3-models in which x and y have the same labels. Here, we show that we can reconfigure any K_3-model to a K_3-model in which x and y have the same labels. This suffices to demonstrate the existence of a reconfiguration sequence between the source and target K_3-models, as we reconfigure from the source K_3-model to a K_3-model in which x and y have the same labels, then to another K_3-model in which x and y have the same labels, and finally to the target K_3-model.

Without loss of generality, we assume that the labels are a, b, and c, and that in the starting K_3-model, $f'(x) = a$ and $f'(y) = b$. If Lemma 12 holds for either relabeling x to b or relabeling y to a, then we can accomplish the reconfiguration in a single step. Similarly, the reconfiguration can be accomplished using Lemma 26 if either x or y is a cut vertex that is not essential for c.

Thus, it suffices to consider the cases in which either condition 1 or 4 of Lemma 12 must be violated for both relabeling x to b and relabeling y to a. In any case, both x and y will be essential for c. By Lemma 11, it is not possible for both x and y to be essential for c unless $|G'(f', a)| = |G'(f', b)| = 1$. Thus, it suffices to consider the case $|G'(f', a)| = |G'(f', b)| = 1$.

Due to the 3-connectivity of G, x and y will each have at least two neighbours in $G'(f', c)$. We will show that one of y's neighbours w can be relabeled b, after which y can be relabeled a.

If Lemma 12 does not apply for the relabeling of w to b, then because w is not b-crucial, the only possible condition of Lemma 12 that can be violated is condition 2. Suppose that every neighbour of y in $G'(f', c)$ is a cut vertex. Since by Lemma 9, each leaf block in $G'(f', c)$ has two interior vertices that are endpoints of connecting edges, each of these edges must connect to x. By 3-connectivity, there must be three vertex-disjoint paths to y from an interior vertex in a leaf block in $G(f', c)$. As only one can pass through x and only one can pass through the joining vertex of the leaf block, there can only be two paths at most, forming a contradiction. Because all conditions of Lemma 12 must hold, we can relabel w to b.

To see that we can now relabel y to a, we observe that since $G'(f', c)$ was connected, z has a neighbour with label c. This implies that y is not essential for c, as needed to satisfy all conditions of Lemma 12. ◀

Finally, in Lemma 30, we show that for G' the graph formed by adding an edge xy to a 3-connected graph $G \in \text{host}(K_3)$, G' is also in $\text{host}(K_3)$. We achieve the result by showing that we can handle situations in which xy plays a role not played by any other edge, either as an essential edge or as a bridge within a branch set. The proof relies on the following results:

▶ **Lemma 28 (*).** *Given a 2-connected graph G and a K_3-model f of G, any branch set containing at least two vertices has no crucial vertex.*

▶ **Lemma 29.** *Given a 3-connected graph G and a K_3-model f of G, suppose that $x \in G(f, a)$, $y \in G(f, b)$, and xy is an essential edge. Then we can reconfigure f to a K_3-model g such that $g(v) \neq f(v)$ for each $v \in G(f, c)$, $c \notin \{a, b\}$, $g(u) = f(u)$ for all other vertices u, and xy is not an essential edge in g.*

Proof. We consider two cases, depending on whether or not $G(f, c)$ is 2-connected.

Case 1: $G(f, c)$ is not 2-connected.

By Lemma 9, each leaf block L in $G(f, c)$ has at least two interior vertices that are endpoints of connecting edges. Due to 3-connectivity, we can further show that each leaf block in $G(f, c)$ has edges to both $G(f, a)$ and $G(f, b)$, as otherwise the joining vertex of the leaf block and either x or y would form a cut set of size two separating internal vertices of the leaf block at one of the branch sets.

The fact that all other leaf blocks connect to both $G(f, a)$ and $G(f, b)$ ensure that f does not hit a leaf-crucial model on relabeling L. We can then use Lemma 19 to relabel all interior vertices of L. Due to the connectivity of L and the fact that it contained neighbours in both $G(f, a)$ and $G(f, b)$, it follows that xy is not an essential edge in the resulting model g.

Case 2: $G(f, c)$ is 2-connected.

We first use 3-connectivity to show that $|G(f, c)| > 1$ and that $G(f, c)$ contains vertices u and v such that u has a neighbour in $G(f, a)$ and v has a neighbour in $G(f, b)$. If $|G(f, c)| = 1$, then the vertex in $G(f, c)$ and either x or y form a cut set of size two separating the branch sets $G(f, a)$ and $G(f, b)$. Similarly, if $G(f, c)$ contained only a single endpoint of a connecting edge, then the endpoint and either x or y would also form a cut set of size two.

We can choose u and v such that P is a (u, v)-path in $G(f, c)$ such that no vertex in P other than u or v has a neighbour in $G(f, a)$ or $G(f, b)$. We let $u = v_1, \ldots, v_t = v$ be the

vertices of P with edges v_iv_{i+1}, $i \in \{1, \ldots, t\}$. Since G is 3-connected and xy is an essential edge, $G(f, c)$ must contain vertices $w \notin \{u, v\}$ and $z \notin \{u, v\}$ such that w has an edge to $G(f, a)$ and z has an edge to $G(f, b)$.

We will attempt to relabel all of P to label a. As $G(f, c)$ is 2-connected, we can relabel v_1 to a. If the resulting branch set is 2-connected, then we attempt relabel v_2,v_3, $\ldots$, v_t in that order until we relabel the entire path. If this succeeds, then we are done. Otherwise, at some step relabeling along the path we obtain a K_3 model g such that $G(g, c)$ is not 2-connected. Now we apply Case 1 to g to complete the claim. ◀

▶ **Lemma 30.** *Suppose G is a 3-connected graph such that $G \in host(K_3)$ and G' is formed from G adding an edge xy. Then $G' \in host(K_3)$.*

Proof. We first observe that any K_3-model of G is also K_3-model of G'. Consequently, to show that we can reconfigure between any K_3-models of G', it suffices to show that we can reconfigure between any K_3-model of G' and a K_3-model of G, as the fact that $G \in \text{host}(K_3)$ ensures that we can reconfigure between any two K_3-models of G.

There are only two cases in which a K_3-model f of G' is not a K_3-model of G, namely cases in which the role xy plays in the K_3-model is not played by any other edge. In both cases we can assume that $G' \neq G$, and consequently that $|V(G')| > 3$.

Case 1: xy is the essential edge connecting $G'(f, f(x))$ and $G'(f, f(y))$

Without loss of generality, we assume $f(x) = a$ and $f(y) = b$. By Lemma 29, we can reconfigure f to a K_3-model f' by relabeling only vertices in $G(f, c)$ such that xy is not an essential edge in f', which means f' is also a K_3-model of G.

Case 2: $f(x) = f(y)$ and xy is a bridge in $G'(f, f(x))$

We show that we can reconfigure to a model in which x and y have different labels so that xy is a connecting edge. Depending on whether xy is then an essential edge, we have either completed the reconfiguration or we have reduced the situation to Case 1.

Without loss of generality, we assume that $f(x) = f(y) = a$, and observe that the removal of xy separates $G'(f, a)$ into two components C_1 (containing x) and C_2 (containing y), each of which contains at least one leaf block.

By Lemma 9, each of the leaf blocks has two vertices with neighbours in $G'(f, b)$ or $G'(f, c)$. We will show that we can reconfigure to a K_3-model in which either C_1 or C_2 has no vertex with label a, so that xy is no longer a bridge.

Case 2a: One component has edges to both $G'(f, b)$ and $G'(f, c)$.

Without loss of generality, let C_1 have connecting edges to both $G'(f, b)$ and $G'(f, c)$. Then f does not hit a leaf-b-crucial model or a leaf-c-crucial model on relabeling C_2 because there are necessary connecting edges from C_1. Also, f does not hit a leaf-crucial model on relabeling C_2 because there are at least two vertices labeled a in each model in the reconfiguration sequence, and so by Lemma 28, there never exists a crucial vertex labeled a. Now by Lemma 20, since x is a cut vertex of $G'(f, a)$, we can relabel all the vertices of C_2, which ensures xy is a connecting edge.

Case 2b: One component has two edges to $G'(f, b)$ and one component has two edges to $G'(f, c)$.

Without loss of generality, let C_1 have connecting edges to $G'(f, b)$ and C_2 have connecting edges to $G'(f, c)$. Then f does not hit a leaf-c-crucial model on relabeling C_2 because C_1 has edges to $G'(f, b)$. Also, it follows from Lemma 28 that f does not hit a leaf-crucial model on relabeling C_2 because there are at least two vertices labeled a in each model in the reconfiguration sequence. Hence, by Lemma 20, since x is a cut vertex of $G'(f, a)$, we can relabel all the vertices of C_2, which ensures xy is a connecting edge. ◀

6 Characterizing host(K_4)

In order to use Ding and Qin's characterization in Theorem 4, we show that $C_6^2 \in \text{host}(K_4)$ (Lemma 32) and $K_5 \in \text{host}(K_4)$ (a special case of Lemma 33) and present analogues of Lemmas 27 and 30 (Lemmas 37 and 38), showing that host(K_4) is closed under the splitting of vertices or adding of edges for 4-connected graphs. The four results are sufficient to prove the following theorem:

▶ **Theorem 31 (*).** *Every 4-connected graph is in host(K_4), provided it is not in $\mathcal{L}$, where $\mathcal{L} = \{H : H$ is the line graph of an internally 4-connected cubic graph$\}$.*

The results establishing the base cases of the characterization are relatively straightforward. Lemma 32 makes use of various properties of the structure of C_6^2, most notably the fact that for each vertex v, there is a unique vertex $s(v)$ such that there is no edge between v and $s(v)$. As an immediate consequence, any four vertices form a 4-cycle, which permits the use of Theorem 21 for reconfiguration of part of the graph. Lemma 33, a generalization of $K_5 \in \text{host}(K_4)$, follows easily from the high connectivity of cliques.

▶ **Lemma 32 (*).** *C_6^2 is in host(K_4).*

▶ **Lemma 33 (*).** *For any $m > \ell$, $K_m \in$ host(K_ℓ).*

As essential edges can result from either the splitting of vertices or the adding of edges, Lemma 36 plays a crucial role in the proofs of both Lemma 37 and Lemma 38. The proof of Lemma 36 makes extensive use of Lemmas 11, 34, and 35 in covering all possible cases of weak connections among branch sets, where branch sets of size one are handled separately.

▶ **Lemma 34 (*).** *Given a 4-connected graph G and a K_4-model f of G such that for labels a, b, c, d there exist weak connections between branch sets with labels a and b, b and c, c and d, and d and a, then it is not possible to designate lynchpins such that the branch set with label a contains at loeast one vertex x that is not a lynchpin and the branch set with label d contains at least one vertex y that is not a lynchpin.*

▶ **Lemma 35.** *Given a 3-connected graph G and an K_4-model f of G, suppose xy is an essential edge and there exists an essential edge e from $G(f, f(y))$ to $G(f, f(z))$, $z \neq x$, such that y is not the endpoint of e. Then it is possible to reconfigure f to a model g in which $g(x) = g(y) = f(x)$, and for all $a \neq f(y)$, for $v \in G(f, a)$, $g(v) = f(v)$.*

Proof. Without loss of generality, we let $f(x) = a$, $f(y) = b$, and $f(z) = c$, so that there is an essential edge ess(b, c). We consider two cases, depending on whether or not y is a cut vertex.

When y is not a cut vertex, we verify that all conditions of Lemma 12 hold for relabeling y to a: condition 1 follows as $G(f, b)$ contains at least y and an endpoint v of ess(b, c), condition 2 follows by assumption, condition 3 follows from the existence of xy, and condition 4 follows

from the observation that if y is an a-crucial vertex, then it must be essential for d, which implies $\{y, z\}$ is a 2-cut in G, contradicting the fact that G is 3-connected.

If instead y is a cut vertex, by removing y we can break $T(G, f, b)$ into components such that one of the components C contains the endpoint of $\text{ess}(b, c)$. By Lemma 9, each leaf block of C must contain at least two vertices with neighbours in other branch sets, and hence C must contain an edge with a neighbour in $G(f, d)$. As C has all necessary connecting edges, we can apply Lemma 20 to siphon away the vertices in every other component $C' \neq C$. Consequently, y will no longer be a cut vertex, we can then relabel y to a, as needed. ◀

▶ **Lemma 36 (*).** *Suppose G is a 4-connected graph such that $G \in host(K_4)$, $|V(G)| \geq 5$, and xy is an essential edge under the K_4-model f. Then it is possible to reconfigure G to a K_4-model in which x and y have the same label.*

Lemma 37 follows the structure of the proof of Lemma 27, relying on Lemma 25 for the reconfiguring between K_4-models in which x and y have the same label and on Lemma 36 for the case in which xy is an essential edge. The two possible cases for Lemma 38 are xy being an essential edge (handled by Lemma 36) and xy being a bridge in a branch set.

▶ **Lemma 37 (*).** *Suppose G is a 4-connected graph such that $G \in host(K_4)$ and G' is a 4-connected graph formed from G by splitting a vertex v into vertices x and y. Then G' is in $host(K_4)$.*

▶ **Lemma 38 (*).** *Suppose G is a 4-connected graph such that $G \in host(K_4)$ and G' is formed from G adding an edge xy. Then $G' \in host(K_4)$.*

7 Conclusions and open questions

We have developed a toolkit for the reconfiguration of minors, and specific results for H-models of small cliques H. Our results imply an alternate definition of 2-connectivity, whereby a graph is 2-connected if and only if it is in host(K_2). Furthermore, we have shown that every 3-connected graph is in host(K_3) and that every 4-connected graph is in host(K_4), provided that it is not in $\mathcal{L}$, where $\mathcal{L} = \{H : H$ is the line graph of an internally 4-connected cubic graph$\}$.

It remains to be shown whether similar results can be obtained for larger cliques, or for other graphs H. As our results rely on characterizations of k-connected graphs, further work is likely to depend on further progress on such results.

As there are alternate ways of defining adjacency relations, further work is needed to determine which definitions are equivalent and for those that are not, what results can be obtained. In our work, we can view each label as a token; based on this viewpoint, the adjacency relation we have considered can be viewed as *Token relabeling (TR)*, changing the label of one vertex in G. Two other possibilities worthy of consideration are *Token sliding (TS)*, swapping the labels of two adjacent vertices in G, and *Token jumping (TJ)*, swapping the labels of any two vertices in G. Both TS [6] and TJ [7] are well-studied for other types of reconfiguration problems, many of which have unlabeled or distinctly labeled tokens. The use of TS instead of TR is instrumental in handling degree-one vertices in G, which otherwise can rarely be relabeled.

Moreover, it is worth considering an alternate formulation in which solutions are considered to be adjacent if one can be formed from another by reassigning labels to vertices according to some permutation on the labels.

Future directions for research include considering other ways of assessing the reconfiguration graph, such as determining its diameter or, in cases in which the reconfiguration graph

is connected, to form algorithms that determine whether there is a path between an input pair of solutions. It remains open how to characterize isolated vertices in the reconfiguration graph, known as *frozen configurations* [1].

Throughout the paper, we required every vertex of G to be a member of a branch set in an H-model. If instead we considered a subgraph of G, a solution might entail the labeling of a subset of the vertices of G. We observe that when the number of labels is equal to the number of vertices in H, the problem is reduced subgraph isomorphism [5]. Alternative mappings can be considered as well, such as topological embedding of one graph in another.

References

1. Richard C. Brewster, Jae-Baek Lee, Benjamin Moore, Jonathan A. Noel, and Mark Siggers. Graph homomorphism reconfiguration and frozen h-colourings. *CoRR*, arXiv:1712.00200, 2017.
2. Erik D. Demaine and MohammadTaghi Hajiaghayi. Graphs excluding a fixed minor have grids as large as treewidth, with combinatorial and algorithmic applications through bidimensionality. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 682–689. Society for Industrial and Applied Mathematics, 2005.
3. Reinhard Diestel. *Graph theory*. Springer-Verlag, Electronic Edition, 2005.
4. Guoli Ding and Chengfu Qin. Generating 4-connected graphs, 2015. URL: `https://www.math.lsu.edu/~ding/chain4.pdf`.
5. Tesshu Hanaka, Takehiro Ito, Haruka Mizuta, Benjamin Moore, Naomi Nishimura, Vijay Subramanya, Akira Suzuki, and Krishna Vaidyanathan. Reconfiguring spanning and induced subgraphs. In *Proceedings of the* 24^{th} *International Computing and Combinatorics Conference*, 2018.
6. Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12-14):1054–1065, 2011.
7. Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012.
8. Anna Lubiw, Zuzana Masárová, and Uli Wagner. Proof of the orbit conjecture for flipping edge-labelled triangulations. In *Proceedings of the* 33^{rd} *International Symposium on Computational Geometry*, 2017.
9. Nicola Martinov. Uncontractable 4-connected graphs. *Journal of Graph Theory*, 6(3):343–344, 1982.
10. Moritz Mühlenthaler. Degree-contrained subgraph reconfiguration is in P. In *40th International Symposium on Mathematical Foundations of Computer Science*, pages 505–516, 2015.
11. N. Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018.
12. Neil Robertson and P.D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. Special Issue Dedicated to Professor W.T. Tutte.
13. W.T. Tutte. A theory of 3-connected graphs. *Indagationes Mathematicae (Proceedings)*, 64:441–455, 1961.
14. Jan van den Heuvel. The complexity of change. *Surveys in Combinatorics 2013*, 409:127–160, 2013.

A Feferman-Vaught Decomposition Theorem for Weighted MSO Logic

Manfred Droste
Institute of Computer Science, Leipzig University, 04109 Leipzig, Germany
droste@informatik.uni-leipzig.de

Erik Paul
Institute of Computer Science, Leipzig University, 04109 Leipzig, Germany
epaul@informatik.uni-leipzig.de

Abstract

We prove a weighted Feferman-Vaught decomposition theorem for disjoint unions and products of finite structures. The classical Feferman-Vaught Theorem describes how the evaluation of a first order sentence in a generalized product of relational structures can be reduced to the evaluation of sentences in the contributing structures and the index structure. The logic we employ for our weighted extension is based on the weighted MSO logic introduced by Droste and Gastin to obtain a Büchi-type result for weighted automata. We show that for disjoint unions and products of structures, the evaluation of formulas from two respective fragments of the logic can be reduced to the evaluation of formulas in the contributing structures. We also prove that the respective restrictions are necessary. Surprisingly, for the case of disjoint unions, the fragment is the same as the one used in the Büchi-type result of weighted automata. In fact, even the formulas used to show that the respective restrictions are necessary are the same in both cases. However, here proving that they do not allow for a Feferman-Vaught-like decomposition is more complex and employs Ramsey's Theorem. We also show how translation schemes can be applied to go beyond disjoint unions and products.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases Quantitative Logic, Quantitative Model Theory, Feferman-Vaught Theorem, Translation Scheme, Transduction

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.76

Funding This work was supported by Deutsche Forschungsgemeinschaft (DFG), Graduiertenkolleg 1763 (QuantLA).

Acknowledgements We are thankful to Vitaly Perevoshchikov for a discussion of the results of this paper.

1 Introduction

The Feferman-Vaught Theorem [6] is one of the fundamental theorems in model theory. The theorem describes how the computation of the truth value of a first order sentence in a generalized product of relational structures can be reduced to the computation of truth values of first order sentences in the contributing structures and the evaluation of a monadic second order sentence in the index structure. The theorem itself has a long-standing history. It builds upon work of Mostowski [17], and was later shown to hold true for monadic second order logic (MSO logic) as well [5, 8, 9, 12, 21]. For a survey and more background information, see [13].

In this paper, we show that under appropriate assumptions, the Feferman-Vaught Theorem also holds true for a weighted MSO logic with arbitrary commutative semirings as weight structure. The logic we employ is based on the weighted logic by Droste and Gastin [3]. In this logic, formulas can take values which convey a quantitative meaning. The logic's connectives and quantifiers hence also adopt quantitative roles. The disjunction becomes a sum, the conjunction a product. The existential quantifier, instead of only checking whether some element with a certain property exists, now takes the truth value of this property for every element in the universe and sums over these values. Under appropriate assumptions, the result of this summation can for instance be the exact number of elements that satisfy the given property. One example of a property which can be expressed using this logic is the number of cliques of a given size in an undirected graph. In [3], the authors prove a Büchi-like result for a specific fragment of the MSO logic, showing that for finite and infinite words, this fragment is expressively equivalent to semiring-weighted automata [20]. The study of a weighted Feferman-Vaught Theorem for disjoint unions, employing the same logic as we do, was initiated by Ravve et al. in [19], where the authors also point out several algorithmic uses and possible applications of a weighted Feferman-Vaught Theorem.

The classical Feferman-Vaught Theorem considers finite and infinite structures without any need for distinction between them. This results from the fact that, in the Boolean setting, infinite joins and meets are well-defined. In particular, existential and universal quantification, which are essentially joins and meets ranging over the whole universe of a structure, are well-defined for finite and infinite structures alike. However, for arbitrary semirings, infinite sums and products are usually not defined. For lack of space, here we consider only finite structures and finite disjoint unions and products of these structures. We note that an extension to infinite structures is possible by employing *bicomplete* semirings. Bicomplete semirings are equipped with infinite sum and product operations that naturally extend their respective finite operations. Our main results are the following.

- We provide a Feferman-Vaught Theorem for disjoint unions of structures with our weighted MSO logic, where the first order product quantifier is restricted to quantify only over formulas which do not contain any sum or product quantifier themselves. Surprisingly, this restriction and the resulting fragment are the same as the one working for the Büchi-like result of [3].
- We show that no similar theorem can hold for disjoint unions if the first order product quantifier is not restricted. The formulas we employ for this in fact also occurred in [3] and [4] as examples of weighted formulas whose semantics could not be described by weighted automata. While in these papers, it was elementary to show that the formulas given define weighted languages not recognizable by weighted automata, here proving that they do not allow for a Feferman-Vaught-like decomposition is more complex and employs a weak version of Ramsey's Theorem [18].
- We show that a Feferman-Vaught Theorem also holds for products of structures for the product-quantifier-free first order fragment of our logic.
- We show that no similar theorem can hold for products if we include the first order product quantifier.
- We show that our theorems are also true for more general disjoint unions and products defined by *translation schemes* [13, 22, 2].

With respect to our proofs, here we just note that in comparison to the universal quantifier of the Boolean setting, the product quantifier requires a separate and new consideration. While universal quantification can simply be expressed using negation and existential quantification, it is in general not possible to express multiplication by addition.

Translation schemes are a model theoretic tool to "translate" structures over one logical signature into structures over another signature in a well behaved fashion, namely in an MSO-defined fashion. They can be applied, for example, to translate between *texts* and *trees* [11], and between *nested words*, *alternating texts*, and *hedges* [16]. These particular translations were employed in [15, 14, 16] to prove that weighted automata over texts, hedges, and nested words are expressively equivalent to weighted logics over these structures. Translation schemes are a rather natural concept and therefore they have been frequently rediscovered and named differently [13, 22, 2]. Our notion of a translation scheme is mostly due to [13].

Related work. A concept related to weighted logics is that of many-valued logics. In both models the evaluation of a formula on a structure produces a quantitative piece of information. In many approaches to many-valued logics, values are taken in the interval $[0, 1]$, cf. [10, 7]. In contrast to this, weights in weighted logics are taken from a semiring and may occur as atomic formulas which enables the modeling of quantitative properties.

2 Preliminaries

Let $\mathbb{N} = \{1, 2, \ldots\}$ and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. A *signature* σ is a pair $(\mathrm{Rel}_\sigma, \mathrm{ar}_\sigma)$ where Rel_σ is a set of relation symbols and $\mathrm{ar}_\sigma \colon \mathrm{Rel}_\sigma \to \mathbb{N}$ the arity function. A *σ-structure* $\mathfrak{A}$ is a pair $(\mathcal{U}_\mathfrak{A}, \mathcal{I}_\mathfrak{A})$ where $\mathcal{U}_\mathfrak{A}$ is a set, called the *universe of* $\mathfrak{A}$, and $\mathcal{I}_\mathfrak{A}$ is an *interpretation*, which maps every $R \in \mathrm{Rel}_\sigma$ to a set $R^\mathfrak{A} \subseteq \mathcal{U}_\mathfrak{A}^{\mathrm{ar}_\sigma(R)}$. A structure is called *finite* if its universe is a finite set. By $\mathrm{Str}(\sigma)$ we denote the class of all σ-structures.

For two σ-structures $\mathfrak{A} = (A, \mathcal{I}_\mathfrak{A})$ and $\mathfrak{B} = (B, \mathcal{I}_\mathfrak{B})$, we define the *product* $\mathfrak{A} \times \mathfrak{B} \in \mathrm{Str}(\sigma)$ *of* $\mathfrak{A}$ *and* $\mathfrak{B}$ and the *disjoint union* $\mathfrak{A} \sqcup \mathfrak{B} \in \mathrm{Str}(\sigma)$ *of* $\mathfrak{A}$ *and* $\mathfrak{B}$ as follows. For the product we let $\mathfrak{A} \times \mathfrak{B} = (A \times B, \mathcal{I}_{\mathfrak{A}\times\mathfrak{B}})$ with $R^{\mathfrak{A}\times\mathfrak{B}} = \{((a_1, b_1), \ldots, (a_k, b_k)) \mid (a_1, \ldots, a_k) \in R^\mathfrak{A} \text{ and } (b_1, \ldots, b_k) \in R^\mathfrak{B}\}$. For the disjoint union, let $A \sqcup B$ be the disjoint union (i.e., the set theoretic coproduct) of A and B with inclusions ι_A and ι_B. Then $\mathfrak{A} \sqcup \mathfrak{B} = (A \sqcup B, \mathcal{I}_{\mathfrak{A}\sqcup\mathfrak{B}})$ with $R^{\mathfrak{A}\sqcup\mathfrak{B}} = \{(\iota_A(a_1), \ldots, \iota_A(a_k)) \mid (a_1, \ldots, a_k) \in R^\mathfrak{A}\} \cup \{(\iota_B(b_1), \ldots, \iota_A(b_k)) \mid (b_1, \ldots, b_k) \in R^\mathfrak{B}\}$. Throughout the paper, we identify $a \in A$ with $\iota_A(a) \in A \sqcup B$ and $b \in B$ with $\iota_B(b) \in A \sqcup B$.

A *commutative semiring* is a tuple $(S, +, \cdot, \mathbb{0}, \mathbb{1})$, abbreviated by S, with operations sum $+$ and product $\cdot$ and constants $\mathbb{0}$ and $\mathbb{1}$ such that $(S, +, \mathbb{0})$ and $(S, \cdot, \mathbb{1})$ are commutative monoids, multiplication distributes over addition, and $s \cdot \mathbb{0} = \mathbb{0}$ for every $s \in S$.

The following definitions are due to [3] in the form of [1]. We provide a countable set $\mathcal{V}$ of first and second order variables, where lower case letters like x and y denote first order variables and capital letters like X and Y denote second order variables. We define monadic second order formulas β over σ and weighted monadic second order formulas φ over σ and S through

$$\beta ::= \mathtt{false} \mid R(x_1, \ldots, x_n) \mid x \in X \mid \neg\beta \mid \beta \vee \beta \mid \exists x.\beta \mid \exists X.\beta$$
$$\varphi ::= \beta \mid s \mid \varphi \oplus \varphi \mid \varphi \otimes \varphi \mid \bigoplus x.\varphi \mid \bigotimes x.\varphi \mid \bigoplus X.\varphi \mid \bigotimes X.\varphi,$$

with $R \in \mathrm{Rel}_\sigma$, $n = \mathrm{ar}_\sigma(R)$, $x, x_1, \ldots, x_n \in \mathcal{V}$ first order variables, $X \in \mathcal{V}$ a second order variable and $s \in S$. We also allow the usual abbreviations $\wedge$, $\forall$, $\to$, $\longleftrightarrow$ and $\mathtt{true}$. By $\mathrm{MSO}(\sigma)$ and $\mathrm{wMSO}(\sigma, S)$ we denote the sets of all monadic second order formulas over σ and all weighted monadic second order formulas over σ and S, respectively. The sets of first order formulas $\mathrm{FO}(\sigma)$ and weighted first order formulas $\mathrm{wFO}(\sigma, S)$ are defined as the sets of all formulas from $\mathrm{MSO}(\sigma)$ and $\mathrm{wMSO}(\sigma, S)$, respectively, which do not contain any subformulas of the form $x \in X$, $\exists X.\beta$, $\bigoplus X.\varphi$ and $\bigotimes X.\varphi$.

The notion of *free variables* is defined as usual, i.e., the operators $\exists, \forall, \bigoplus$ and $\bigotimes$ bind variables. We let $\mathrm{Free}(\varphi)$ be the set of all free variables of φ. A formula φ with $\mathrm{Free}(\varphi) = \emptyset$ is called a *sentence*. For a vector $\bar{\varphi} = (\varphi_1, \dots, \varphi_n) \in \mathrm{wMSO}(\sigma, S)^n$, we define the set of free variables of $\bar{\varphi}$ as $\mathrm{Free}(\bar{\varphi}) = \bigcup_{i=1}^{n} \mathrm{Free}(\varphi_i)$.

We now define the semantics of MSO and wMSO. Let σ be a signature, $\mathfrak{A} = (A, \mathcal{I}_{\mathfrak{A}})$ a σ-structure and $\mathcal{V}$ a set of first and second order variables. A $(\mathcal{V}, \mathfrak{A})$-assignment ρ is a partial function $\rho\colon \mathcal{V} \nrightarrow A \cup \mathcal{P}(A)$ such that, whenever $x \in \mathcal{V}$ is a first order variable and $\rho(x)$ is defined, we have $\rho(x) \in A$, and whenever $X \in \mathcal{V}$ is a second order variable and $\rho(X)$ is defined, we have $\rho(X) \subseteq A$. The reason we consider partial functions is that in our Feferman-Vaught theorems for the disjoint union of structures we want to be able to restrict the range of a variable assignment to a subset of the universe. For a first order variable, this restriction may cause the variable to become undefined. Let $\mathrm{dom}(\rho)$ be the domain of ρ. For a first order variable $x \in \mathcal{V}$ and an element $a \in A$, the *update* $\rho[x \to a]$ is defined through $\mathrm{dom}(\rho[x \to a]) = \mathrm{dom}(\rho) \cup \{x\}$, $\rho[x \to a](\mathcal{X}) = \rho(\mathcal{X})$ for all $\mathcal{X} \in \mathcal{V} \setminus \{x\}$ and $\rho[x \to a](x) = a$. For a second order variable $X \in \mathcal{V}$ and a set $I \subseteq A$, the update $\rho[X \to I]$ is defined in a similar fashion. By $\mathfrak{A}_{\mathcal{V}}$ we denote the set of all $(\mathcal{V}, \mathfrak{A})$-assignments.

For $\rho \in \mathfrak{A}_{\mathcal{V}}$ and a formula $\beta \in \mathrm{MSO}(\sigma)$ the relation "$(\mathfrak{A}, \rho)$ satisfies β", denoted by $(\mathfrak{A}, \rho) \models \beta$, is defined as usual, with the minor addition that $(\mathfrak{A}, \rho)$ can satisfy $x \in X$ and $R(x_1, \dots, x_n)$ only if all of the occurring variables are in $\mathrm{dom}(\rho)$. In the following, for all sums and products to be well-defined, we assume that A is finite. For a formula $\varphi \in \mathrm{wMSO}(\sigma, S)$ and a structure $\mathfrak{A} \in \mathrm{Str}(\sigma)$, the *(weighted) semantics* of φ is a mapping $[\![\varphi]\!](\mathfrak{A}, \cdot)\colon \mathfrak{A}_{\mathcal{V}} \to S$ inductively defined as

$$
\begin{aligned}
[\![\beta]\!](\mathfrak{A}, \rho) &= \begin{cases} \mathbb{1} & \text{if } (\mathfrak{A}, \rho) \models \beta \\ \mathbb{0} & \text{otherwise} \end{cases} \\
[\![s]\!](\mathfrak{A}, \rho) &= s \\
[\![\varphi_1 \oplus \varphi_2]\!](\mathfrak{A}, \rho) &= [\![\varphi_1]\!](\mathfrak{A}, \rho) + [\![\varphi_2]\!](\mathfrak{A}, \rho) \\
[\![\varphi_1 \otimes \varphi_2]\!](\mathfrak{A}, \rho) &= [\![\varphi_1]\!](\mathfrak{A}, \rho) \cdot [\![\varphi_2]\!](\mathfrak{A}, \rho)
\end{aligned}
\qquad
\begin{aligned}
[\![\textstyle\bigoplus x.\varphi]\!](\mathfrak{A}, \rho) &= \textstyle\sum_{a \in A} [\![\varphi]\!](\mathfrak{A}, \rho[x \to a]) \\
[\![\textstyle\bigotimes x.\varphi]\!](\mathfrak{A}, \rho) &= \textstyle\prod_{a \in A} [\![\varphi]\!](\mathfrak{A}, \rho[x \to a]) \\
[\![\textstyle\bigoplus X.\varphi]\!](\mathfrak{A}, \rho) &= \textstyle\sum_{I \subseteq A} [\![\varphi]\!](\mathfrak{A}, \rho[X \to I]) \\
[\![\textstyle\bigotimes X.\varphi]\!](\mathfrak{A}, \rho) &= \textstyle\prod_{I \subseteq A} [\![\varphi]\!](\mathfrak{A}, \rho[X \to I]).
\end{aligned}
$$

We will usually identify a pair $(\mathfrak{A}, \emptyset)$ with $\mathfrak{A}$. For a vector of formulas $\bar{\varphi} \in \mathrm{wMSO}(\sigma, S)^n$, we define $[\![\bar{\varphi}]\!](\mathfrak{A}, \rho) = ([\![\varphi_1]\!](\mathfrak{A}, \rho), \dots, [\![\varphi_n]\!](\mathfrak{A}, \rho)) \in S^n$.

We give some examples of how weighted formulas can be interpreted. For more examples, see also [19].

▶ **Example 1.** If $S = \mathbb{B}$ is the Boolean semiring, we obtain the classical Boolean logic.

▶ **Example 2.** Assume that $S = (\mathbb{Q}, +, \cdot, 0, 1)$ is the field of rational numbers and that σ is the signature of an (undirected) graph, i.e., $\mathrm{Rel}_\sigma = \{\mathrm{edge}\}$ with edge binary. Then for every fixed $n \in \mathbb{N}$, we can count the number of n-cliques of a graph with no loops $\mathfrak{G} \in \mathrm{Str}(\sigma)$ using the formula $\varphi = \frac{1}{n!} \otimes \bigoplus x_1 \dots \bigoplus x_n . \bigwedge_{i \neq j} (\mathrm{edge}(x_i, x_j) \vee \mathrm{edge}(x_j, x_i))$.

▶ **Example 3.** We consider the *minimum cut* of directed acyclic graphs. For this, we interpret these graphs as *flow networks* in the following way. Every vertex which does not have a predecessor is considered a *source*, every vertex without successors is considered a *drain*, and every edge is assumed to have a capacity of 1. Let $G = (V, E)$ be a directed acyclic graph where V is the set of vertices and $E \subseteq V \times V$ the set of edges. A cut (S, D) of G is a partition of V, i.e., $S \cup D = V$ and $S \cap D = \emptyset$, such that all sources of G are in S, and all drains of G are in D. The *minimum cut* of G is the smallest number $|E \cap (S \times D)|$ such that (S, D) is a cut of G.

We can express the minimum cut of directed acyclic graphs by a weighted formula as follows. We let σ be the signature from the previous example and this time interpret it as the signature of a directed graph. For our semiring, we choose the tropical semiring $\mathrm{Trop} = (\mathbb{R}_{\geq 0} \cup \{\infty\}, \min, +, \infty, 0)$. Then using the abbreviation

$$\mathrm{cut}(X,Y) = \forall x.\Big((x \in X \leftrightarrow \neg(x \in Y)) \wedge (\exists y.\mathrm{edge}(y,x) \vee x \in X) \wedge (\exists y.\mathrm{edge}(x,y) \vee x \in Y)\Big)$$

we can express the minimum cut of a directed acyclic graph $\mathfrak{G} \in \mathrm{Str}(\sigma)$ using the formula

$$\varphi = \bigoplus X.\bigoplus Y.\Big(\mathrm{cut}(X,Y) \otimes \bigotimes x.\bigotimes y.(1 \oplus \neg(x \in X \wedge y \in Y \wedge \mathrm{edge}(x,y)))\Big).$$

For $\varphi \in \mathrm{wMSO}(\sigma, S)$ and a first order variable x which does not appear in φ as a bound variable, we define φ^{-x} as the formula obtained from φ by replacing all atomic subformulas containing x, i.e., all subformulas of the form $x \in X$ and $R(\ldots, x, \ldots)$ for $R \in \mathrm{Rel}_\sigma$, by `false`. It is easy to show by induction that for all σ-structures $\mathfrak{A} = (A, \mathcal{I}_\mathfrak{A})$ and $(\mathcal{V}, \mathfrak{A})$-assignments ρ with $x \notin \mathrm{dom}(\rho)$ we have $[\![\varphi]\!](\mathfrak{A}, \rho) = [\![\varphi^{-x}]\!](\mathfrak{A}, \rho)$. As in the sequel we will deal with disjoint unions and products of structures, we need to define the restrictions of a variable assignment to the contributing structures of the disjoint union or product. Fix two structures $\mathfrak{A}, \mathfrak{B} \in \mathrm{Str}(\sigma)$ with universes A and B. For a $(\mathcal{V}, \mathfrak{A} \sqcup \mathfrak{B})$-assignment ρ, we define the restriction $\rho|_\mathfrak{A} \colon \mathcal{V} \rightharpoonup A$ as

$$\rho|_\mathfrak{A}(\mathcal{X}) = \begin{cases} \rho(\mathcal{X}) \cap A & \text{if } \mathcal{X} \text{ is a second order variable} \\ \rho(\mathcal{X}) & \text{if } \mathcal{X} \text{ is a first order variable and } \rho(\mathcal{X}) \in A \\ \text{undefined} & \text{if } \mathcal{X} \text{ is a first order variable and } \rho(\mathcal{X}) \notin A. \end{cases}$$

The restriction $\rho|_\mathfrak{B}$ is defined similarly.

For a $(\mathcal{V}, \mathfrak{A} \times \mathfrak{B})$-assignment ρ, we define the restrictions $\rho|_\mathfrak{A}$ and $\rho|_\mathfrak{B}$ by projection on the corresponding entries. That is, we let π_A be the projection on the first and π_B be the projection on the second entry of $A \times B$ and let $\rho|_\mathfrak{A} = \pi_A \circ \rho$ and $\rho|_\mathfrak{B} = \pi_B \circ \rho$.

The *union* of two assignments ρ and ς with $\mathrm{dom}(\rho) \cap \mathrm{dom}(\varsigma) = \emptyset$, denoted by $\rho \cup \varsigma$, is defined by $\mathrm{dom}(\rho \cup \varsigma) = \mathrm{dom}(\rho) \cup \mathrm{dom}(\varsigma)$, $(\rho \cup \varsigma)(\mathcal{X}) = \rho(\mathcal{X})$ for $\mathcal{X} \in \mathrm{dom}(\rho)$ and $(\rho \cup \varsigma)(\mathcal{X}) = \varsigma(\mathcal{X})$ for $\mathcal{X} \in \mathrm{dom}(\varsigma)$.

Fix two disjoint sets of variables $(x_i)_{i \in \mathbb{N}}$ and $(y_i)_{i \in \mathbb{N}}$. For $n \in \mathbb{N}$ we define the set of *expressions* $\mathrm{Exp}_n(S)$ over a semiring S by the grammar

$$E ::= x_i \mid y_i \mid E \oplus E \mid E \otimes E,$$

where $i \in \{1, \ldots, n\}$. The *(weighted) semantics* of an expression $E \in \mathrm{Exp}_n(S)$ is a mapping $\langle\!\langle E \rangle\!\rangle \colon S^n \times S^n \to S$ defined for $\bar{s}, \bar{t} \in S^n$ inductively by

$$\langle\!\langle x_i \rangle\!\rangle(\bar{s}, \bar{t}) = s_i \qquad \langle\!\langle E_1 \oplus E_2 \rangle\!\rangle(\bar{s}, \bar{t}) = \langle\!\langle E_1 \rangle\!\rangle(\bar{s}, \bar{t}) + \langle\!\langle E_2 \rangle\!\rangle(\bar{s}, \bar{t})$$
$$\langle\!\langle y_i \rangle\!\rangle(\bar{s}, \bar{t}) = t_i \qquad \langle\!\langle E_1 \otimes E_2 \rangle\!\rangle(\bar{s}, \bar{t}) = \langle\!\langle E_1 \rangle\!\rangle(\bar{s}, \bar{t}) \cdot \langle\!\langle E_2 \rangle\!\rangle(\bar{s}, \bar{t}).$$

For expressions over the Boolean semiring $\mathbb{B} = (\{\mathtt{false}, \mathtt{true}\}, \vee, \wedge, \mathtt{false}, \mathtt{true})$ we will usually write $\vee$ instead of $\oplus$ and $\wedge$ instead of $\otimes$.

▶ **Construction 4.** We call an expression $E \in \mathrm{Exp}_n(S)$ a *pure product* if

$$E = \mathrm{x}_1 \otimes \ldots \otimes \mathrm{x}_l \otimes \mathrm{y}_1 \otimes \ldots \otimes \mathrm{y}_m$$

with $\mathrm{x}_i \in \{x_1, \ldots, x_n\}$ for $i \in \{1, \ldots, l\}$ and $\mathrm{y}_j \in \{y_1, \ldots, y_n\}$ for $j \in \{1, \ldots, m\}$. We define a substitution procedure as follows. Let $\bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^{2} \in \mathrm{wMSO}(\sigma, S)^n$ be given. Let $i \in \{1, \ldots, l\}$ and assume $\mathrm{x}_i = x_k$ for some k, then we define $\xi_i = \varphi_k^{\mathbb{1}}$. Likewise, for $j \in \{1, \ldots, m\}$ and $\mathrm{y}_j = y_k$, we define $\theta_j = \varphi_k^{2}$. We let $\xi = \xi_1 \otimes \ldots \otimes \xi_l$ and $\theta = \theta_1 \otimes \ldots \otimes \theta_m$. Then for $\mathfrak{A}, \mathfrak{B} \in \mathrm{Str}(\sigma)$, every $(\mathcal{V}, \mathfrak{A})$-assignment ρ and every $(\mathcal{V}, \mathfrak{B})$-assignment ς we have

$$\langle\!\langle E \rangle\!\rangle([\![\bar{\varphi}^{\mathbb{1}}]\!](\mathfrak{A}, \rho), [\![\bar{\varphi}^{2}]\!](\mathfrak{B}, \varsigma)) = [\![\xi]\!](\mathfrak{A}, \rho) \cdot [\![\theta]\!](\mathfrak{B}, \varsigma).$$

We define $\mathrm{PRD}^{\mathbb{1}}(E, \bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^{2}) = \xi$ and $\mathrm{PRD}^{2}(E, \bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^{2}) = \theta$.

Pure products $B \in \mathrm{Exp}_n(\mathbb{B})$ are also called *pure conjunctions*. For a pure conjunction $B \in \mathrm{Exp}_n(\mathbb{B})$, formulas $\bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^{2} \in \mathrm{MSO}(\sigma)$ and ξ_i, θ_j as above, we define the $\mathrm{MSO}(\sigma)$-formulas $\mathrm{CON}^{\mathbb{1}}(B, \bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^{2}) = \xi = \xi_1 \wedge \ldots \wedge \xi_l$ and $\mathrm{CON}^{2}(B, \bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^{2}) = \theta = \theta_1 \wedge \ldots \wedge \theta_m$. We then have

$$\langle\!\langle B \rangle\!\rangle([\![\bar{\varphi}^{\mathbb{1}}]\!](\mathfrak{A}, \rho), [\![\bar{\varphi}^{2}]\!](\mathfrak{B}, \varsigma)) = \mathtt{true} \text{ iff } (\mathfrak{A}, \rho) \models \xi \text{ and } (\mathfrak{B}, \varsigma) \models \theta. \qquad \lozenge$$

We say that an expression $E \in \mathrm{Exp}_n(S)$ is in *normal form* if $E = E_1 \oplus \ldots \oplus E_m$ for some $m \geq 1$ and pure products E_i. By applying the laws of distributivity of the semiring S, every expression $E \in \mathrm{Exp}_n(S)$ can be transformed into normal form. More precisely, we have the following lemma.

▶ **Lemma 5.** *For every $E \in \mathrm{Exp}_n(S)$ there exists an expression $E' \in \mathrm{Exp}_n(S)$ in normal form with the same semantics as E.*

3 The classical Feferman-Vaught Theorem

For convenience, we recall the Feferman-Vaught Theorem for disjoint unions and products of two structures. Let σ be a signature.

▶ **Theorem 6** ([6])**.** *Let $\mathcal{V}$ be a set of first and second order variables and $\beta \in \mathrm{MSO}(\sigma)$ with variables from $\mathcal{V}$. Then there exist $n \geq 1$, vectors of formulas $\bar{\beta}^{\mathbb{1}}, \bar{\beta}^{2} \in \mathrm{MSO}(\sigma)^n$ and an expression $B_\beta \in \mathrm{Exp}_n(\mathbb{B})$ such that $\mathrm{Free}(\bar{\beta}^{\mathbb{1}}) \cup \mathrm{Free}(\bar{\beta}^{2}) \subseteq \mathrm{Free}(\beta)$ and for all structures $\mathfrak{A}, \mathfrak{B} \in \mathrm{Str}(\sigma)$ and all $(\mathcal{V}, \mathfrak{A} \sqcup \mathfrak{B})$-assignments ρ:*

$$(\mathfrak{A} \sqcup \mathfrak{B}, \rho) \models \beta \text{ iff } \langle\!\langle B_\beta \rangle\!\rangle([\![\bar{\beta}^{\mathbb{1}}]\!](\mathfrak{A}, \rho|_{\mathfrak{A}}), [\![\bar{\beta}^{2}]\!](\mathfrak{B}, \rho|_{\mathfrak{B}})) = \mathtt{true}.$$

▶ **Theorem 7** ([6])**.** *Let $\mathcal{V}$ be a set of first and second order variables and $\beta \in \mathrm{FO}(\sigma)$ with variables from $\mathcal{V}$. Then there exist $n \geq 1$, vectors of formulas $\bar{\beta}^{\mathbb{1}}, \bar{\beta}^{2} \in \mathrm{FO}(\sigma)^n$ and an expression $B_\beta \in \mathrm{Exp}_n(\mathbb{B})$ such that $\mathrm{Free}(\bar{\beta}^{\mathbb{1}}) \cup \mathrm{Free}(\bar{\beta}^{2}) \subseteq \mathrm{Free}(\beta)$ and for all structures $\mathfrak{A}, \mathfrak{B} \in \mathrm{Str}(\sigma)$ and all $(\mathcal{V}, \mathfrak{A} \times \mathfrak{B})$-assignments ρ:*

$$(\mathfrak{A} \times \mathfrak{B}, \rho) \models \beta \text{ iff } \langle\!\langle B_\beta \rangle\!\rangle([\![\bar{\beta}^{\mathbb{1}}]\!](\mathfrak{A}, \rho|_{\mathfrak{A}}), [\![\bar{\beta}^{2}]\!](\mathfrak{B}, \rho|_{\mathfrak{B}})) = \mathtt{true}.$$

4 Translation schemes

Theorems 6 and 7 consider disjoint unions and products only. So far, there is no interaction between the two constituting structures. *Translation schemes* allow us to create such interactions in an MSO-defined manner. More precisely, translation schemes "translate" structures over one signature into structures over another signature. Applying this to disjoint unions and products, we can extend Theorems 6 and 7 to more complex constructs. The usefulness of such extensions by translation schemes was discussed in [13], which we follow here.

Let σ and τ be two signatures, $\mathcal{Z} = \{z, z_1, z_2, \ldots\}$ be a set of distinguished first order variables and $\mathcal{W}$ be a set of first and second order variables with $\mathcal{W} \cap \mathcal{Z} = \emptyset$. A *$\sigma$-$\tau$-translation scheme Φ over $\mathcal{W}$ and $\mathcal{Z}$* is a pair $(\phi_\mathcal{U}, (\phi_T)_{T\in\mathrm{Rel}_\tau})$ where $\phi_\mathcal{U}, \phi_T \in \mathrm{MSO}(\sigma)$, $\phi_\mathcal{U}$ has variables from $\mathcal{W} \cup \{z\}$ and ϕ_T has variables from $\mathcal{W} \cup \{z_1, \ldots, z_{\mathrm{ar}_\tau(T)}\}$. The variables from $\mathcal{Z}$ may not be used for quantification, i.e., all variables from $\mathcal{Z}$ must be free. We set $\mathrm{Free}(\Phi) = \mathrm{Free}(\phi_\mathcal{U}) \cup \bigcup_{T\in\mathrm{Rel}_\tau} \mathrm{Free}(\phi_T)$. The formulas $\phi_\mathcal{U}$ and $(\phi_T)_{T\in\mathrm{Rel}_\tau}$ depend on $\mathcal{Z}$ in the following way. For a first order variable x not occurring in $\phi_\mathcal{U}$, the formula $\phi_\mathcal{U}(x)$ is obtained from $\phi_\mathcal{U}$ by replacing all occurrences of z by x. Similarly, for $T \in \mathrm{Rel}_\tau$ and first order variables $x_1, \ldots, x_{\mathrm{ar}_\tau(T)}$ not occurring in ϕ_T, the formula $\phi_T(x_1, \ldots, x_{\mathrm{ar}_\tau(T)})$ is obtained from ϕ_T by replacing all occurrences of z_i by x_i for $i \in \{1, \ldots, \mathrm{ar}_\tau(T)\}$.

For a σ-structure $\mathfrak{A} = (A, \mathcal{I}_\mathfrak{A})$ and a $(\mathcal{W}, \mathfrak{A})$-assignment ς, we define the *Φ-induced τ-structure of $\mathfrak{A}$ and ς*, denoted by $\Phi^\star(\mathfrak{A}, \varsigma)$, as a τ-structure with universe $\mathcal{U}_\mathfrak{C}$ and interpretation $\mathcal{I}_\mathfrak{C}$ as follows.

$$\mathcal{U}_\mathfrak{C} = \{a \in A \mid (\mathfrak{A}, \varsigma[z \to a]) \models \phi_\mathcal{U}\} \qquad \mathcal{I}_\mathfrak{C}(T) = \{\bar{c} \in \mathcal{U}_\mathfrak{C}^{\mathrm{ar}_\tau(T)} \mid (\mathfrak{A}, \varsigma[\bar{z} \to \bar{c}]) \models \phi_T\}$$

▶ **Example 8.** A translation scheme can be used to cut a subtree from a given tree at a specified node in the tree. For this let $\sigma = \tau = (\{\mathrm{edge}\}, \mathrm{edge} \mapsto 2)$ be the signature of a directed graph. For a σ-structure $\mathfrak{G} = (V, \mathrm{edge} \mapsto E)$ let E' be the transitive closure of the relation $E \subseteq V \times V$. We say that $\mathfrak{G}$ is a directed rooted tree with root $r \in V$ if (1) E' is irreflexive, (2) $(r, v) \in E'$ for all $v \in V \setminus \{r\}$ and (3) for all $v \in V \setminus \{r\}$ there is exactly one $v' \in V$ with $(v', v) \in E$. We define the following abbreviation which describes the reflexive transitive closure of E.

$$(x \leq y) = \forall X\, (x \in X \wedge (\forall z.(\exists z'.z' \in X \wedge \mathrm{edge}(z', z)) \to z \in X)) \to y \in X$$

We define a σ-σ-translation scheme $\Phi = (\phi_\mathcal{U}, \phi_\mathrm{edge})$ through $\phi_\mathcal{U} = (x \leq z)$ and $\phi_\mathrm{edge} = \mathrm{edge}(z_1, z_2)$. Then with $\mathfrak{G}$ as above and $v \in V$, the structure $\mathfrak{C} = \Phi^*(\mathfrak{G}, x \mapsto v)$ is the subtree of $\mathfrak{G}$ at the node v, i.e.,

$$\mathcal{U}_\mathfrak{C} = \{v\} \cup \{v' \in V \mid (v, v') \in E'\} \qquad \mathcal{I}_\mathfrak{C} = E \cap (\mathcal{U}_\mathfrak{C} \times \mathcal{U}_\mathfrak{C}).$$

We have the following fundamental property of translation schemes [13].

▶ **Lemma 9** ([13]). *Let $\Phi = (\phi_\mathcal{U}, (\phi_T)_{T\in\mathrm{Rel}_\tau})$ be a σ-τ-translation scheme over $\mathcal{W}$ and $\mathcal{Z}$, $\mathcal{V}$ be a set of first and second order variables such that $\mathcal{V}$, $\mathcal{W}$, and $\mathcal{Z}$ are pairwise disjoint, and $\beta \in \mathrm{MSO}(\tau)$ with variables from $\mathcal{V}$. Then there exists a formula $\alpha \in \mathrm{MSO}(\sigma)$ such that $\mathrm{Free}(\alpha) \subseteq \mathrm{Free}(\beta) \cup \mathrm{Free}(\Phi)$ and for all structures $\mathfrak{A} \in \mathrm{Str}(\sigma)$, all $(\mathcal{W}, \mathfrak{A})$-assignments ς and all $(\mathcal{V}, \Phi^\star(\mathfrak{A}, \varsigma))$-assignments ρ:*

$$(\Phi^\star(\mathfrak{A}, \varsigma), \rho) \models \beta \text{ iff } (\mathfrak{A}, \varsigma \cup \rho) \models \alpha.$$

Together with Theorems 6 and 7, this gives us the following Feferman-Vaught decomposition theorems for disjoint unions and products with translations schemes.

▶ **Theorem 10** ([13]). *Let $\Phi = (\phi_\mathcal{U}, (\phi_T)_{T\in\mathrm{Rel}_\tau})$ be a σ-τ-translation scheme over $\mathcal{W}$ and $\mathcal{Z}$, $\mathcal{V}$ be a set of first and second order variables such that $\mathcal{V}$, $\mathcal{W}$, and $\mathcal{Z}$ are pairwise disjoint, and $\beta \in \mathrm{MSO}(\tau)$ with variables from $\mathcal{V}$. Then there exist $n \geq 1$, vectors of formulas $\bar{\beta}^{\mathbb{1}}, \bar{\beta}^2 \in \mathrm{MSO}(\sigma)^n$ and an expression $B_\beta \in \mathrm{Exp}_n(\mathbb{B})$ such that $\mathrm{Free}(\bar{\beta}^{\mathbb{1}}) \cup \mathrm{Free}(\bar{\beta}^2) \subseteq \mathrm{Free}(\beta) \cup \mathrm{Free}(\Phi)$ and for all structures $\mathfrak{A}, \mathfrak{B} \in \mathrm{Str}(\sigma)$, all $(\mathcal{W}, \mathfrak{A} \sqcup \mathfrak{B})$-assignments ς and all $(\mathcal{V}, \Phi^\star(\mathfrak{A} \sqcup \mathfrak{B}, \varsigma))$-assignments ρ:*

$$(\Phi^\star(\mathfrak{A} \sqcup \mathfrak{B}, \varsigma), \rho) \models \beta \text{ iff } \langle\!\langle B_\beta \rangle\!\rangle([\![\bar{\beta}^{\mathbb{1}}]\!](\mathfrak{A}, (\varsigma \cup \rho)|_\mathfrak{A}), [\![\bar{\beta}^2]\!](\mathfrak{B}, (\varsigma \cup \rho)|_\mathfrak{B})) = \mathtt{true}.$$

▶ **Theorem 11** ([13]). *Let $\Phi = (\phi_{\mathcal{U}}, (\phi_T)_{T\in \mathrm{Rel}_\tau})$ be a σ-τ-translation scheme over $\mathcal{W}$ and $\mathcal{Z}$, $\mathcal{V}$ be a set of first and second order variables such that $\mathcal{V}$, $\mathcal{W}$, and $\mathcal{Z}$ are pairwise disjoint, and $\beta \in \mathrm{FO}(\tau)$ with variables from $\mathcal{V}$. Then there exist $n \geq 1$, vectors of formulas $\bar{\beta}^{\mathbb{1}}, \bar{\beta}^{2} \in \mathrm{FO}(\sigma)^n$ and an expression $B_\beta \in \mathrm{Exp}_n(\mathbb{B})$ such that $\mathrm{Free}(\bar{\beta}^{\mathbb{1}}) \cup \mathrm{Free}(\bar{\beta}^{2}) \subseteq \mathrm{Free}(\beta) \cup \mathrm{Free}(\Phi)$ and for all structures $\mathfrak{A}, \mathfrak{B} \in \mathrm{Str}(\sigma)$, all $(\mathcal{W}, \mathfrak{A} \sqcup \mathfrak{B})$-assignments ς and all $(\mathcal{V}, \Phi^\star(\mathfrak{A} \times \mathfrak{B}, \varsigma))$-assignments ρ:*

$$(\Phi^\star(\mathfrak{A} \times \mathfrak{B}, \varsigma), \rho) \models \beta \text{ iff } \langle\!\langle B_\beta \rangle\!\rangle([\![\bar{\beta}^{\mathbb{1}}]\!](\mathfrak{A}, (\varsigma \cup \rho)|_{\mathfrak{A}}), [\![\bar{\beta}^{2}]\!](\mathfrak{B}, (\varsigma \cup \rho)|_{\mathfrak{B}})) = \mathtt{true}.$$

We give a short example to illustrate Theorem 10.

▶ **Example 12.** We consider the signature σ of a labeled graph, i.e., $\mathrm{Rel}_\sigma = \{\mathrm{edge}, \mathrm{lab}_a, \mathrm{lab}_b\}$ where edge has arity 2 and $\mathrm{lab}_a, \mathrm{lab}_b$ both have arity 1. Given two directed rooted labeled trees $\mathfrak{G}_1, \mathfrak{G}_2$ in this signature (see Example 8), we can use a translation scheme to add edges between all leaves of $\mathfrak{G}_1$ and the root of $\mathfrak{G}_2$ in $\mathfrak{G}_1 \sqcup \mathfrak{G}_2$. For this scenario we have to distinguish between the vertices from the first and the second graph, so the use of an intermediate signature is necessary. We define the signature σ' to be σ extended by the relation symbols G_1 and G_2 of arity 1. Then for $i \in \{1, 2\}$ we define a σ-σ'-translation scheme $\Phi_i = (\phi_{\mathcal{U}}, \phi'_{\mathrm{edge}}, \phi_{\mathrm{lab}_a}, \phi_{\mathrm{lab}_b}, \phi^i_{G_1}, \phi^i_{G_2})$ by

$$\phi_{\mathcal{U}} = \mathtt{true} \qquad \phi_{\mathrm{lab}_a} = \mathrm{lab}_a(z_1) \qquad \phi^i_{G_j} = \begin{cases} \mathtt{true} & \text{if } i = j \\ \mathtt{false} & \text{otherwise.} \end{cases}$$
$$\phi'_{\mathrm{edge}} = \mathrm{edge}(z_1, z_2) \qquad \phi_{\mathrm{lab}_b} = \mathrm{lab}_b(z_1)$$

With the abbreviations $\mathrm{root}(x) = \neg\exists y.\mathrm{edge}(y, x)$ and $\mathrm{leaf}(x) = \neg\exists y.\mathrm{edge}(x, y)$ we then define the σ'-σ-translation scheme $\Phi = (\phi_{\mathcal{U}}, \phi_{\mathrm{edge}}, \phi_{\mathrm{lab}_a}, \phi_{\mathrm{lab}_b})$ through

$$\phi_{\mathrm{edge}} = \mathrm{edge}(z_1, z_2) \vee (G_1(z_1) \wedge G_2(z_2) \wedge \mathrm{leaf}(z_1) \wedge \mathrm{root}(z_2)).$$

Then $\mathfrak{G} = \Phi^*(\Phi_1^*(\mathfrak{G}_1) \sqcup \Phi_2^*(\mathfrak{G}_2))$ is exactly $\mathfrak{G}_1 \sqcup \mathfrak{G}_2$ with the leaves of $\mathfrak{G}_1$ connected to the root of $\mathfrak{G}_2$. We now consider the formula

$$\beta = \exists x.\exists y.(\mathrm{edge}(x, y) \wedge \mathrm{lab}_a(x) \wedge \mathrm{lab}_b(y))$$

which asks whether there is some edge between an a-labeled and a b-labeled vertex. We can apply Lemma 9 and Theorem 10 to obtain the following decomposition of β. Let

$$\bar{\beta}^{\mathbb{1}} = (\beta, \exists x.\mathrm{lab}_a(x) \wedge \mathrm{leaf}(x)) \qquad B_\beta = x_1 \vee y_1 \vee (x_2 \wedge y_2).$$
$$\bar{\beta}^{2} = (\beta, \exists y.\mathrm{lab}_b(y) \wedge \mathrm{root}(y))$$

Then we have $\mathfrak{G} \models \beta$ iff $\langle\!\langle B_\beta \rangle\!\rangle([\![\bar{\beta}^{\mathbb{1}}]\!](\mathfrak{G}_1), [\![\bar{\beta}^{2}]\!](\mathfrak{G}_2)) = \mathtt{true}$.

5 Weighted Feferman-Vaught Decomposition Theorems

Our goal is to prove weighted versions of Theorems 10 and 11. That is, we would like to replace FO by wFO and MSO by wMSO in those theorems. This, however, is not possible as we will see in Sections 5.2 and 5.3. For disjoint unions, we have to restrict the use of the first order product quantifier and entirely remove the second order product quantifier in wMSO. For products, it is not possible to include the first order product quantifier at all.

5.1 Formulation of the theorems

Let σ be a signature and S a commutative semiring. We define two fragments of our logic and formulate our weighted versions of Theorems 10 and 11 for these fragments.

▶ **Definition 13** (Product-free weighted first order logic)**.** We define the *product-free* first order fragment $\mathrm{wFO}^{\bigotimes\text{-free}}(\sigma, S)$ of our logic as the set of all formulas from $\mathrm{wFO}(\sigma, S)$ which do not contain any first order product quantifier. Using this fragment, we will formulate a weighted Feferman-Vaught decomposition theorem for products of structures.

▶ **Definition 14** (Product-restricted weighted monadic second order logic)**.** In order to define the *product-restricted* fragment of our weighted monadic second order logic, we first define the fragment of so-called *almost-Boolean* formulas through the grammar

$$\psi ::= \beta \mid s \mid \psi \oplus \psi \mid \psi \otimes \psi.$$

This fragment, which we denote by $\mathrm{wMSO}^{\text{a-bool}}(\sigma, S)$, already appeared in [3] in the form of *recognizable step functions*. To obtain the main theorem of [3], the product quantifier was restricted to quantify only over recognizable step functions. We employ the same restriction and define the product-restricted fragment of our logic through the grammar

$$\varphi ::= \beta \mid s \mid \varphi \oplus \varphi \mid \varphi \otimes \varphi \mid \bigoplus x.\varphi \mid \bigotimes x.\psi \mid \bigoplus X.\varphi,$$

where $\beta \in \mathrm{MSO}(\sigma)$ is a monadic second order formula, $s \in S$, x is a first order variable, X is a second order variable and $\psi \in \mathrm{wMSO}^{\text{a-bool}}(\sigma, S)$ is an almost-Boolean formula. By $\mathrm{wMSO}^{\bigotimes\text{-res}}(\sigma, S)$ we denote the set of all such formulas. The set $\mathrm{wMSO}^{\bigotimes\text{-res}}(\sigma, S)$ contains all formulas from $\mathrm{wMSO}(\sigma, S)$ which do not contain any second order quantifier and where for every subformula of the form $\bigotimes x.\psi$ we have that ψ is an almost-Boolean formula. Our weighted Feferman-Vaught decomposition theorem for disjoint unions of structures will be formulated for this fragment. In [3] it was shown that for finite and infinite words, this fragment is expressively equivalent to weighted finite automata.

We note that the restrictions we impose on the product quantifier are necessary as we will show in Sections 5.2 and 5.3. We formulate the weighted versions of Theorems 10 and 11 as follows.[1] Let τ, $\mathcal{W}$ and $\mathcal{Z}$ be as in Section 4.

▶ **Theorem 15.** *Let S be a commutative semiring. Let $\Phi = (\phi_{\mathcal{U}}, (\phi_T)_{T \in \mathrm{Rel}_\tau})$ be a σ-τ-translation scheme over $\mathcal{W}$ and $\mathcal{Z}$, $\mathcal{V}$ be a set of first and second order variables such that $\mathcal{V}$, $\mathcal{W}$, and $\mathcal{Z}$ are pairwise disjoint, and $\varphi \in \mathrm{wMSO}^{\bigotimes\text{-res}}(\tau, S)$ with variables from $\mathcal{V}$. Then there exist $n \geq 1$, vectors of formulas $\bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^2 \in \mathrm{wMSO}^{\bigotimes\text{-res}}(\sigma, S)^n$ with $\mathrm{Free}(\bar{\varphi}^{\mathbb{1}}) \cup \mathrm{Free}(\bar{\varphi}^2) \subseteq \mathrm{Free}(\varphi) \cup \mathrm{Free}(\Phi)$ and an expression $E_\varphi \in \mathrm{Exp}_n(S)$ such that the following holds. For all finite structures $\mathfrak{A}, \mathfrak{B} \in \mathrm{Str}(\sigma)$, all $(\mathcal{W}, \mathfrak{A} \sqcup \mathfrak{B})$-assignments ς and all $(\mathcal{V}, \Phi^\star(\mathfrak{A} \sqcup \mathfrak{B}, \varsigma))$-assignments ρ we have*

$$[\![\varphi]\!](\Phi^\star(\mathfrak{A} \sqcup \mathfrak{B}, \varsigma), \rho) = \langle\!\langle E_\varphi \rangle\!\rangle([\![\bar{\varphi}^{\mathbb{1}}]\!](\mathfrak{A}, (\varsigma \cup \rho)|_{\mathfrak{A}}), [\![\bar{\varphi}^2]\!](\mathfrak{B}, (\varsigma \cup \rho)|_{\mathfrak{B}})).$$

[1] In [19] a weighted version of Theorem 10 similar to ours is stated (without proof) to hold without any restriction on the first order product quantifier. However, in Subsection 5.2 we show that a restriction on the product quantifier is necessary.

▶ **Theorem 16.** *Let S be a commutative semiring. Let $\Phi = (\phi_{\mathcal{U}}, (\phi_T)_{T\in \mathrm{Rel}_\tau})$ be a σ-τ-translation scheme over $\mathcal{W}$ and $\mathcal{Z}$, $\mathcal{V}$ be a set of first and second order variables such that $\mathcal{V}$, $\mathcal{W}$, and $\mathcal{Z}$ are pairwise disjoint, and $\varphi \in \mathrm{wFO}^{\bigotimes\text{-free}}(\tau, S)$ with variables from $\mathcal{V}$. Then there exist $n \geq 1$, vectors of formulas $\bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^{2} \in \mathrm{wFO}^{\bigotimes\text{-free}}(\sigma, S)^n$ with $\mathrm{Free}(\bar{\varphi}^{\mathbb{1}}) \cup \mathrm{Free}(\bar{\varphi}^{2}) \subseteq \mathrm{Free}(\varphi) \cup \mathrm{Free}(\Phi)$ and an expression $E_\varphi \in \mathrm{Exp}_n(S)$ such that the following holds. For all finite structures $\mathfrak{A}, \mathfrak{B} \in \mathrm{Str}(\sigma)$, all $(\mathcal{W}, \mathfrak{A}\times\mathfrak{B})$-assignments ς and all $(\mathcal{V}, \Phi^\star(\mathfrak{A}\times\mathfrak{B}, \varsigma))$-assignments ρ we have*

$$[\![\varphi]\!](\Phi^\star(\mathfrak{A}\times\mathfrak{B}, \varsigma), \rho) = \langle\!\langle E_\varphi \rangle\!\rangle([\![\bar{\varphi}^{\mathbb{1}}]\!](\mathfrak{A}, (\varsigma \cup \rho)|_{\mathfrak{A}}), [\![\bar{\varphi}^{2}]\!](\mathfrak{B}, (\varsigma \cup \rho)|_{\mathfrak{B}})).$$

The proofs of both theorems are deferred to Section 5.4. For formulas without free variables and a trivial translation scheme, i.e., $\phi_{\mathcal{U}} = \mathtt{true}$ and $\phi_T = T(z_1, \ldots, z_{\mathrm{ar}_\tau(T)})$ for all $T \in \mathrm{Rel}_\tau$, the theorems reduce to the following, simplified versions.

▶ **Theorem 17.** *Let S be a commutative semiring and $\varphi \in \mathrm{wMSO}^{\bigotimes\text{-res}}(\sigma, S)$ be a sentence. Then there exist $n \geq 1$, vectors of sentences $\bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^{2} \in \mathrm{wMSO}^{\bigotimes\text{-res}}(\sigma, S)^n$ and an expression $E_\varphi \in \mathrm{Exp}_n(S)$ such that the following holds. For all finite structures $\mathfrak{A}, \mathfrak{B} \in \mathrm{Str}(\sigma)$ we have*

$$[\![\varphi]\!](\mathfrak{A} \sqcup \mathfrak{B}) = \langle\!\langle E_\varphi \rangle\!\rangle([\![\bar{\varphi}^{\mathbb{1}}]\!](\mathfrak{A}), [\![\bar{\varphi}^{2}]\!](\mathfrak{B})).$$

▶ **Theorem 18.** *Let S be a commutative semiring and $\varphi \in \mathrm{wFO}^{\bigotimes\text{-free}}(\sigma, S)$ be a sentence. Then there exist $n \geq 1$, vectors of sentences $\bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^{2} \in \mathrm{wFO}^{\bigotimes\text{-free}}(\sigma, S)^n$ and an expression $E_\varphi \in \mathrm{Exp}_n(S)$ such that the following holds. For all finite structures $\mathfrak{A}, \mathfrak{B} \in \mathrm{Str}(\sigma)$ we have*

$$[\![\varphi]\!](\mathfrak{A} \times \mathfrak{B}) = \langle\!\langle E_\varphi \rangle\!\rangle([\![\bar{\varphi}^{\mathbb{1}}]\!](\mathfrak{A}), [\![\bar{\varphi}^{2}]\!](\mathfrak{B})).$$

▶ **Example 19.** To illustrate Theorem 17 we consider the semiring of natural numbers $\mathbb{N}_0 = (\mathbb{N}_0, +, \cdot, 0, 1)$ and the signature σ of a labeled graph, i.e., $\mathrm{Rel}_\sigma = \{\mathrm{edge}, \mathrm{lab}_a, \mathrm{lab}_b\}$ with edge binary and $\mathrm{lab}_a, \mathrm{lab}_b$ both unary. Consider the following formula which multiplies the number of vertices labeled a with the number of edges between two vertices labeled b.

$$\Big(\underbrace{\textstyle\bigoplus x.\mathrm{lab}_a(x)}_{=\varphi_a}\Big) \otimes \Big(\underbrace{\textstyle\bigoplus x.\bigoplus y.\mathrm{edge}(x,y) \wedge \mathrm{lab}_b(x) \wedge \mathrm{lab}_b(y)}_{=\varphi_b}\Big)$$

The formula can be decomposed as follows. Let $\bar{\varphi}^{\mathbb{1}} = \bar{\varphi}^{2} = (\varphi_a, \varphi_b)$ and $E_\varphi = (x_1 \oplus y_1) \otimes (x_2 \oplus y_2)$. Then for all σ-structures $\mathfrak{G}_1, \mathfrak{G}_2$ we have $[\![\varphi]\!](\mathfrak{G}_1 \sqcup \mathfrak{G}_2) = \langle\!\langle E_\varphi \rangle\!\rangle([\![\bar{\varphi}^{\mathbb{1}}]\!](\mathfrak{G}_1), [\![\bar{\varphi}^{2}]\!](\mathfrak{G}_2))$.

▶ **Example 20.** In [19], it is discussed how translation schemes can be applied for Feferman-Vaught-like decompositions of weighted properties. Theorems 15 and 16 show that this is possible for all properties which can be expressed by formulas in our weighted logic fragments.

5.2 Necessity of restricting the logic for disjoint unions

In this section, we show that the restrictions we impose on the product quantifiers are indeed necessary. For disjoint unions, we will prove that already Theorem 17 does not hold over the tropical semiring $\mathrm{Trop} = (\mathbb{R}_{\geq 0} \cup \{\infty\}, \min, +, \infty, 0)$ and over the arctic semiring $\mathrm{Arct} = (\mathbb{R}_{\geq 0} \cup \{-\infty\}, \max, +, -\infty, 0)$ for the formulas $\bigotimes x. \bigotimes y.1$ and $\bigotimes X.1$. Here, $\mathbb{R}_{\geq 0}$ denotes the set of non-negative real numbers. To prove this, we employ Ramsey's Theorem. Then we show that for the formula $\bigotimes x. \bigoplus y.1$, Theorem 17 does not hold over the semiring $\mathbb{N}_0 = (\mathbb{N}_0, +, \cdot, 0, 1)$. We note that these types of formulas also occurred in [3] and [4] as examples of weighted formulas whose semantics could not be described by weighted automata.

We will employ the following version of Ramsey's Theorem. For a set X, we denote by $\binom{X}{2}$ the set of all subsets of X of size 2.

▶ **Theorem 21** ([18]). *Let* $f\colon \left[\frac{\mathbb{N}}{2}\right] \to \{1,\dots,k\}$ *be a function. Then there exists an infinite subset* $E \subseteq \mathbb{N}$ *such that* $f|_{\left[\frac{E}{2}\right]} \equiv i$ *for some* $i \in \{1,\dots,k\}$.

▶ **Theorem 22.** *Let* $S \in \{\mathrm{Trop}, \mathrm{Arct}\}$, $\sigma = (\emptyset,\emptyset)$ *be the empty signature and for* $l \in \mathbb{N}$ *consider the* σ*-structures* $\mathfrak{S}_l = (\{1,\dots,l\},\emptyset)$. *Then for* $\varphi = \bigotimes x.\bigotimes y.1$ *there do not exist* $n \in \mathbb{N}$, $\bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^2 \in (\mathrm{wMSO}(\sigma,S))^n$ *and* $E_\varphi \in \mathrm{Exp}_n(S)$ *such that for all* $l,m \in \mathbb{N}$ *we have*

$$[\![\varphi]\!](\mathfrak{S}_l \sqcup \mathfrak{S}_m) = \langle\!\langle E_\varphi \rangle\!\rangle([\![\bar{\varphi}^{\mathbb{1}}]\!](\mathfrak{S}_l), [\![\bar{\varphi}^2]\!](\mathfrak{S}_m)). \tag{5.1}$$

Proof (Sketch). First, consider $S = \mathrm{Trop}$. For contradiction, suppose that $n, \bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^2$ and E_φ as above satisfying (5.1) exist. We may assume that $E_\varphi = E_1 \oplus \ldots \oplus E_k$ is in normal form with all E_i pure products. For $l \geq 1$ and $i \in \{1,\dots,k\}$ we let $a_{li} = [\![\mathrm{PRD}^{\mathbb{1}}(E_i, \bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^2)]\!](\mathfrak{S}_l)$ and $b_{li} = [\![\mathrm{PRD}^2(E_i, \bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^2)]\!](\mathfrak{S}_l)$. Then by assumption we have

$$(l+m)^2 = [\![\varphi]\!](\mathfrak{S}_l \sqcup \mathfrak{S}_m) = \min_{i=1}^{k}(a_{li} + b_{mi}). \tag{5.2}$$

Given $l \geq 1$ and $m \geq 1$, for at least one index $j \in \{1,\dots,k\}$ we have $(l+m)^2 = a_{lj} + b_{mj}$. We define j_{lm} as the smallest such index. Then we define a function $f\colon \left[\frac{\mathbb{N}}{2}\right] \to \{1,\dots,k\}$ by $f(\{l,m\}) = j_{lm}$ for $l < m$. Now take $E \subseteq \mathbb{N}$ according to Ramsey's Theorem. As E is infinite, there are $l,\lambda,m,\mu \in E$ with $l < \lambda < m < \mu$. With $j = j_{lm}$, we thus have $(l+m)^2 = a_{lj} + b_{mj}$, $(\lambda+m)^2 = a_{\lambda j} + b_{mj}$, $(l+\mu)^2 = a_{lj} + b_{\mu j}$, and $(\lambda+\mu)^2 = a_{\lambda j} + b_{\mu j}$. Using the first three of these equalities, an elementary calculation shows that we have $a_{\lambda j} + b_{\mu j} < (\lambda+\mu)^2$. This is clearly a contradiction to the fourth equality. Therefore, $n, \bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^2$ and E_φ as chosen cannot exist. To prove the theorem for the arctic semiring, it suffices to replace min by max in equation (5.2). ◀

With similar methods, we can show the following.

▶ **Theorem 23.** *Let* $S \in \{\mathrm{Trop}, \mathrm{Arct}\}$, $\sigma = (\emptyset,\emptyset)$ *be the empty signature and for* $l \in \mathbb{N}$ *consider the* σ*-structures* $\mathfrak{S}_l = (\{1,\dots,l\},\emptyset)$. *Then for* $\varphi = \bigotimes X.1$ *there do not exist* $n \in \mathbb{N}$, $\bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^2 \in (\mathrm{wMSO}(\sigma,S))^n$ *and* $E_\varphi \in \mathrm{Exp}_n(S)$ *such that for all* $l,m \in \mathbb{N}$ *we have*

$$[\![\varphi]\!](\mathfrak{S}_l \sqcup \mathfrak{S}_m) = \langle\!\langle E_\varphi \rangle\!\rangle([\![\bar{\varphi}^{\mathbb{1}}]\!](\mathfrak{S}_l), [\![\bar{\varphi}^2]\!](\mathfrak{S}_m)).$$

The nesting of a first order sum quantifier into the first order product quantifier also leads to formulas which do not allow for a Feferman-Vaught-like decomposition as the following theorem shows.

▶ **Theorem 24.** *Let* $S = (\mathbb{N}_0, +, \cdot, 0, 1)$, $\sigma = (\emptyset,\emptyset)$ *be the empty signature and for* $l \in \mathbb{N}$ *consider the* σ*-structures* $\mathfrak{S}_l = (\{1,\dots,l\},\emptyset)$. *Then for* $\varphi = \bigotimes x.\bigoplus y.1$ *there do not exist* $n \in \mathbb{N}$, $\bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^2 \in (\mathrm{wMSO}(\sigma,\mathbb{N}_0))^n$ *and* $E_\varphi \in \mathrm{Exp}_n(\mathbb{N}_0)$ *such that for all* $l,m \in \mathbb{N}$ *we have*

$$[\![\varphi]\!](\mathfrak{S}_l \sqcup \mathfrak{S}_m) = \langle\!\langle E_\varphi \rangle\!\rangle([\![\bar{\varphi}^{\mathbb{1}}]\!](\mathfrak{S}_l), [\![\bar{\varphi}^2]\!](\mathfrak{S}_m)). \tag{5.3}$$

Proof (Sketch). We proceed by contradiction and assume $n, \bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^2$ and E_φ as above satisfying (5.3) exist. We may assume that $E_\varphi = E_1 \oplus \ldots \oplus E_k$ is in normal form with all E_i pure products. For $l \geq 1$ and $i \in \{1,\dots,k\}$ we let $a_{li} = [\![\mathrm{PRD}^{\mathbb{1}}(E_i, \bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^2)]\!](\mathfrak{S}_l)$ and $b_{li} = [\![\mathrm{PRD}^2(E_i, \bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^2)]\!](\mathfrak{S}_l)$. Then by assumption we have

$$(l+m)^{(l+m)} = [\![\varphi]\!](\mathfrak{S}_l \sqcup \mathfrak{S}_m) = \textstyle\sum_{i=1}^{k}(a_{li} \cdot b_{mi}). \tag{5.4}$$

For every $j \in \{1, \dots, k\}$ we choose $L_j \geq 1$ such that $a_{L_j j} \neq 0$, or let $L_j = 0$ if for all $l \geq 1$ we have $a_{lj} = 0$. Assume $m \geq 1$ and $j \in \{1, \dots, k\}$ with $L_j \neq 0$, then $a_{L_j j} \geq 1$, hence

$$(L_j + m)^{(L_j+m)} = \textstyle\sum_{i=1}^{k} (a_{L_j i} \cdot b_{mi}) \geq (a_{L_j j} \cdot b_{mj}) \geq b_{mj}.$$

In particular, with $L = \max\{L_i \mid i \in \{1, \dots, k\}\}$, we have that for every $j \in \{1, \dots, k\}$ either (i) $b_{mj} \leq (L + m)^{(L+m)}$ for all $m \geq 1$ or (ii) $a_{lj} = 0$ for all $l \geq 1$. Note that from equation (5.4) it follows that $L = 0$ is impossible. In the same fashion, we can find $M \geq 1$ such that for every $l \geq 1$ and every $j \in \{1, \dots, k\}$ either (i) $a_{lj} \leq (l + M)^{(l+M)}$ for all $l \geq 1$ or (ii) $b_{mj} = 0$ for all $m \geq 1$.

Now consider (5.4) for $l = m$. If $j \in \{1, \dots, k\}$ such that either $a_{lj} = 0$ for all $l \geq 1$ or $b_{mj} = 0$ for all $m \geq 1$, then clearly also $(a_{lj} \cdot b_{lj}) = 0$ for all l. If j is not like this, we have

$$(a_{lj} \cdot b_{lj}) \leq (l + M)^{(l+M)} \cdot (L + l)^{(L+l)} \leq (l + C)^{2(l+C)}$$

for $C = \max\{L, M\}$. It follows that $(2l)^{2l} \leq k(l + C)^{2(l+C)}$ for every $l \geq 1$. Elementary calculus can be used to show that this is not true. ◀

5.3 Necessity of restricting the logic for products

The proof of Theorem 22 can also be used to show that no Feferman-Vaught-like theorem holds for products if the first order product quantifier is included in the weighted logic. More precisely, already Theorem 18 does not hold over the tropical and arctic semirings for the formula $\varphi = \bigotimes x.1$ even if $\bar{\varphi}^{\mathbb{1}}$ and $\bar{\varphi}^{2}$ are allowed to be from $\mathrm{wMSO}(\sigma, S)$.

▶ **Theorem 25.** *Let $S \in \{\mathrm{Trop}, \mathrm{Arct}\}$, $\sigma = (\emptyset, \emptyset)$ be the empty signature and for $l \in \mathbb{N}$ consider the σ-structures $\mathfrak{S}_l = (\{1, \dots, l\}, \emptyset)$. Then for $\varphi = \bigotimes x.1$ there do not exist $n \in \mathbb{N}$, $\bar{\varphi}^{\mathbb{1}}, \bar{\varphi}^{2} \in (\mathrm{wMSO}(\sigma, S))^n$ and $E_\varphi \in \mathrm{Exp}_n(S)$ such that for all $l, m \in \mathbb{N}$ we have*

$$[\![\varphi]\!](\mathfrak{S}_l \times \mathfrak{S}_m) = \langle\!\langle E_\varphi \rangle\!\rangle([\![\bar{\varphi}^{\mathbb{1}}]\!](\mathfrak{S}_l), [\![\bar{\varphi}^{2}]\!](\mathfrak{S}_m)).$$

5.4 Proofs of Theorems 15 and 16

We now come to the proof of Theorems 15 and 16. By the following result, we can reduce the proofs to the case where the translation scheme is the identity.

▶ **Lemma 26.** *Let $\Phi = (\phi_{\mathcal{U}}, (\phi_T)_{T \in \mathrm{Rel}_\tau})$ be a σ-τ-translation scheme over $\mathcal{W}$ and $\mathcal{Z}$, $\mathcal{V}$ be a set of first and second order variables such that $\mathcal{V}$, $\mathcal{W}$, and $\mathcal{Z}$ are pairwise disjoint, and $\varphi \in \mathrm{wMSO}(\tau, S)$ with variables from $\mathcal{V}$. Then there exists a formula $\psi \in \mathrm{wMSO}(\sigma, S)$ with $\mathrm{Free}(\psi) \subseteq \mathrm{Free}(\varphi) \cup \mathrm{Free}(\Phi)$ such that the following holds. For all finite structures $\mathfrak{A} \in \mathrm{Str}(\sigma)$, all $(\mathcal{W}, \mathfrak{A})$-assignments ς and all $(\mathcal{V}, \Phi^\star(\mathfrak{A}, \varsigma))$-assignments ρ we have*

$$[\![\varphi]\!](\Phi^\star(\mathfrak{A}, \varsigma), \rho) = [\![\psi]\!](\mathfrak{A}, \varsigma \cup \rho).$$

If φ is from $\mathrm{wMSO}^{\bigotimes\text{-res}}(\tau, S)$ or $\mathrm{wFO}^{\bigotimes\text{-free}}(\tau, S)$, then ψ can also be chosen as a formula from $\mathrm{wMSO}^{\bigotimes\text{-res}}(\sigma, S)$ or $\mathrm{wFO}^{\bigotimes\text{-free}}(\sigma, S)$, respectively.

Lemma 26 can be proved by induction on the structure of formulas.

Proof of Theorem 15 (Sketch). We proceed by induction. By Lemma 26 it suffices to prove the case $\tau = \sigma$ and $\Phi^\star(\mathfrak{A} \sqcup \mathfrak{B}, \varsigma) = \mathfrak{A} \sqcup \mathfrak{B}$.

- Assume $\varphi = \beta$ for some $\beta \in \mathrm{MSO}(\sigma)$. We apply Theorem 6 to the formula β and obtain $l \geq 1$, vectors of formulas $\bar{\beta}^{\mathbb{1}}, \bar{\beta}^{2} \in \mathrm{MSO}(\sigma)^l$ and an expression $B_\beta \in \mathrm{Exp}_l(\mathbb{B})$ such that

  $$(\mathfrak{A} \sqcup \mathfrak{B}, \rho) \models \beta \text{ iff } \langle\!\langle B_\beta \rangle\!\rangle([\![\bar{\beta}^{\mathbb{1}}]\!](\mathfrak{A}, \rho|_{\mathfrak{A}}), [\![\bar{\beta}^{2}]\!](\mathfrak{B}, \rho|_{\mathfrak{B}})) = \mathtt{true}.$$

  We may assume that $B_\beta = B_1 \vee \ldots \vee B_m$ is in normal form with all B_i pure conjunctions. We let $\gamma_i = \mathrm{CON}^{\mathbb{1}}(B_i, \bar{\beta}^{\mathbb{1}}, \bar{\beta}^{2})$ and $\delta_i = \mathrm{CON}^{2}(B_i, \bar{\beta}^{\mathbb{1}}, \bar{\beta}^{2})$ for $i \in \{1, \ldots, m\}$ (see Construction 4). We set $n = 2m$ and define

  $$\bar{\varphi}^{\mathbb{1}} = (\gamma_1, \ldots, \gamma_m, \neg\gamma_1, \ldots, \neg\gamma_m) \qquad \bar{\varphi}^{2} = (\delta_1, \ldots, \delta_m, \neg\delta_1, \ldots, \neg\delta_m).$$

  Intuitively, we would now define the expression E_φ as $x_1 \otimes y_1 \oplus \ldots \oplus x_m \otimes y_m$, but this expression is not necessarily evaluated to $\mathbb{1}$ in S if $\gamma_i \wedge \delta_i$ is true for more than one index i. Instead, we define expressions $E_k \in \mathrm{Exp}_n(S)$ for $k \in \{1, \ldots, m\}$ inductively by $E_1 = x_1 \otimes y_1$ and

  $$E_k = (E_{k-1} \otimes ((x_{k+m} \otimes y_k) \oplus y_{k+m})) \oplus (x_k \otimes y_k)$$

  and set $E_\varphi = E_m$. In a sense, E_k is evaluated to $\mathbb{1}$ if $\gamma_k \wedge \delta_k$ is true, and otherwise, if either γ_k or δ_k does not hold, it is evaluated to E_{k-1}.
- Assume $\varphi = s$ for some $s \in S$. We let $n = 1$, $\varphi^{\mathbb{1}}_1 = \varphi^{2}_1 = s$ and $E_\varphi = x_1$.
- For $\varphi = \zeta \oplus \eta$, we assume the theorem is true for ζ with $\bar{\zeta}^{\mathbb{1}}, \bar{\zeta}^{2} \in \mathrm{wMSO}^{\bigotimes\text{-res}}(\sigma, S)^l$ and $E_\zeta \in \mathrm{Exp}_l(S)$, and for η with $\bar{\eta}^{\mathbb{1}}, \bar{\eta}^{2} \in \mathrm{wMSO}^{\bigotimes\text{-res}}(\sigma, S)^m$ and $E_\eta \in \mathrm{Exp}_m(S)$. We set $\bar{\varphi}^{\mathbb{1}} = (\zeta^{\mathbb{1}}_1, \ldots, \zeta^{\mathbb{1}}_l, \eta^{\mathbb{1}}_1, \ldots, \eta^{\mathbb{1}}_m)$, $\bar{\varphi}^{2} = (\zeta^{2}_1, \ldots, \zeta^{2}_l, \eta^{2}_1, \ldots, \eta^{2}_m)$ and $E_\varphi = E_\zeta \oplus E'_\eta$, where E'_η is obtained from E_η by replacing every variable x_i by x_{i+l} and every variable y_i by y_{i+l}.
- For $\varphi = \zeta \otimes \eta$, the proof is the same as for the previous case, only that here we define $E_\varphi = E_\zeta \otimes E'_\eta$.
- For $\varphi = \bigoplus x.\zeta$, we assume the theorem is true for ζ with $\bar{\zeta}^{\mathbb{1}}, \bar{\zeta}^{2} \in \mathrm{wMSO}^{\bigotimes\text{-res}}(\sigma, S)^l$ and $E_\zeta \in \mathrm{Exp}_l(S)$. We may assume that $E_\zeta = E_1 \oplus \ldots \oplus E_m$ is in normal form with all E_i pure products and that x does no occur as a bound variable in any of the $\zeta^{\mathbb{1}}_i$ or ζ^{2}_i. We let $\xi_i = \mathrm{PRD}^{\mathbb{1}}(E_i, \bar{\zeta}^{\mathbb{1}}, \bar{\zeta}^{2})$ and $\theta_i = \mathrm{PRD}^{2}(E_i, \bar{\zeta}^{\mathbb{1}}, \bar{\zeta}^{2})$. We set $n = 2m$ and define

  $$\begin{aligned} \varphi^{\mathbb{1}} &= (\textstyle\bigoplus x.\xi_1, \ldots, \bigoplus x.\xi_m, \xi_1^{-x}, \ldots, \xi_m^{-x}) \\ \varphi^{2} &= (\textstyle\bigoplus x.\theta_1, \ldots, \bigoplus x.\theta_m, \theta_1^{-x}, \ldots, \theta_m^{-x}) \end{aligned} \qquad E_\varphi = \textstyle\bigoplus_{i=1}^{m}((x_i \otimes y_{m+i}) \oplus (x_{m+i} \otimes y_i)).$$

- For $\varphi = \bigoplus X.\zeta$, we assume that the theorem is true for ζ with $\bar{\zeta}^{\mathbb{1}}, \bar{\zeta}^{2} \in \mathrm{wMSO}^{\bigotimes\text{-res}}(\sigma, S)^l$ and $E_\zeta \in \mathrm{Exp}_l(S) = E_\zeta = E_1 \oplus \ldots \oplus E_m$ in normal form with all E_i pure products. We let $\xi_i = \mathrm{PRD}^{\mathbb{1}}(E_i, \bar{\zeta}^{\mathbb{1}}, \bar{\zeta}^{2})$ and $\theta_i = \mathrm{PRD}^{2}(E_i, \bar{\zeta}^{\mathbb{1}}, \bar{\zeta}^{2})$. We set $n = m$ and define

  $$\begin{aligned} \varphi^{\mathbb{1}}_j &= (\textstyle\bigoplus X.\xi_1, \ldots, \bigoplus X.\xi_m) \\ \varphi^{2}_j &= (\textstyle\bigoplus X.\theta_1, \ldots, \bigoplus X.\theta_m) \end{aligned} \qquad E_\varphi = \textstyle\bigoplus_{i=1}^{m}(x_i \otimes y_i).$$

- Assume $\varphi = \bigotimes x.\zeta$ with $\zeta \in \mathrm{wMSO}^{\text{a-bool}}(\sigma, S)$ almost boolean. Using the laws of distributivity in S and the fact that for two boolean formulas $\alpha, \beta \in \mathrm{MSO}(\sigma)$ we have $[\![\alpha]\!] \cdot [\![\beta]\!] \equiv [\![\alpha \otimes \beta]\!]$, we may assume that $\zeta = (s_1 \otimes \beta_1) \oplus \ldots \oplus (s_l \otimes \beta_l)$ for some $l \geq 1$, $s_i \in S$ and $\beta_i \in \mathrm{MSO}(\sigma)$. Applying a simple construction, we may even assume that $\beta_1, \ldots, \beta_l$ form a partition, i.e., that for all $(\mathcal{V}, \mathfrak{A} \sqcup \mathfrak{B})$-assignments ρ' there is exactly one $i \in \{1, \ldots, l\}$ with $(\mathfrak{A} \sqcup \mathfrak{B}, \rho') \models \beta_i$. Let $X_1, \ldots, X_l \in \mathcal{V}$ be second order variables not occurring in ζ. We define the abbreviation

  $$(x \in X_i) \rhd s_i = ((x \in X_i) \otimes s_i) \oplus \neg(x \in X_i).$$

One can check elementarily that

$$[\![\varphi]\!] \equiv [\![\bigoplus X_1 \ldots \bigoplus X_l. \left(\bigwedge_{i=1}^{l} \forall x.(x \in X_i \leftrightarrow \beta_i)\right) \otimes \bigotimes_{i=1}^{l} \bigotimes x.((x \in X_i) \rhd s_i)]\!].$$

Therefore, it suffices to show this case for formulas of the form

$$\varphi = \bigotimes x.((x \in X) \rhd s).$$

For this, we let $n = 1$ and define $\varphi^{\mathbb{1}} = \varphi^{\mathbb{2}} = (\bigotimes x.((x \in X) \rhd s))$ and $E_\varphi = x_1 \otimes y_1$. ◀

Proof of Theorem 16 (Sketch). Again we proceed by induction and assume that $\tau = \sigma$ and $\Phi^\star(\mathfrak{A} \times \mathfrak{B}, \varsigma) = \mathfrak{A} \times \mathfrak{B}$. The proofs for the cases $\varphi = \beta$, $\varphi = s$, $\varphi = \zeta \oplus \eta$ and $\varphi = \zeta \otimes \eta$ are identical to the ones used in the proof of Theorem 15 for the corresponding cases. For the case $\varphi = \bigoplus x.\zeta$ we proceed as for the case $\varphi = \bigoplus X.\zeta$ in the proof of Theorem 15. ◀

6 Conclusion

We have derived a weighted version of the Feferman-Vaught Theorem for disjoint unions and products of finite structures. We just mention here three possible extensions that were left out due to lack of space. First, Theorems 15 and 16 also hold for infinite structures if the commutative semiring S is bicomplete, i.e., if it is equipped with infinite sum and product operations that naturally extend its finite sum and product operations. Second, in the particular case that the semiring S is a De Morgan algebra, both theorems hold without any need for restrictions on the product quantifiers; that is, Theorem 15 holds for the full weighted MSO logic, and Theorem 16 holds for the full weighted FO logic. Third, both theorems may be extended to employ transductions as defined by Courcelle [2] in place of the present translation schemes.

References

1 Benedikt Bollig, Paul Gastin, and Benjamin Monmege. Weighted specifications over nested words. In Frank Pfenning, editor, *Proc. FoSSaCS*, volume 7794 of *LNCS*, pages 385–400. Springer, 2013.

2 Bruno Courcelle. Monadic second-order definable graph transductions: a survey. *Theor. Comput. Sci.*, 126(1):53–75, 1994.

3 Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theor. Comput. Sci.*, 380(1-2):69–86, 2007.

4 Manfred Droste and George Rahonis. Weighted automata and weighted logics with discounting. *Theor. Comput. Sci.*, 410(37):3481–3494, 2009. doi:10.1016/j.tcs.2009.03.029.

5 Andrzej Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fund. Math.*, 49(2):129–141, 1961.

6 Solomon Feferman and Robert L. Vaught. The first order properties of products of algebraic systems. *Fund. Math.*, 47:57–103, 1959.

7 Siegfried Gottwald. *A Treatise on Many-Valued Logics*, volume 9 of *Studies in Logic and Computation*. Research Studies Press, 2001.

8 Yuri Gurevich. Modest theory of short chains. I. *J. Symbolic Logic*, 44(4):481–490, 12 1979.

9 Yuri Gurevich. Chapter XIII: Monadic second-order theories. In Jon Barwise and Solomon Feferman, editors, *Model-Theoretic Logics*, volume 8 of *Perspect. Math. Logic*, pages 479–506. Springer, 1985.

10 Petr Hájek. *Metamathematics of Fuzzy Logic*, volume 4 of *Trends in Logic*. Kluwer, 1998.

11 Hendrik Jan Hoogeboom and Paulien ten Pas. Monadic second-order definable text languages. *Theory Comput. Syst.*, 30(4):335–354, 1997. doi:10.1007/s002240000055.

12 Hans Läuchli and John Leonard. On the elementary theory of linear order. *Fund. Math.*, 59(1):109–116, 1966.

13 Johann A. Makowsky. Algorithmic uses of the Feferman–Vaught theorem. *Ann. Pure Appl. Logic*, 126(1):159–213, 2004.

14 Christian Mathissen. Definable transductions and weighted logics for texts. In Tero Harju, Juhani Karhumäki, and Arto Lepistö, editors, *Proc. DLT*, pages 324–336. Springer, 2007.

15 Christian Mathissen. Weighted logics for nested words and algebraic formal power series. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Proc. ICALP*, volume 5126 of *LNCS*, pages 221–232. Springer, 2008.

16 Christian Mathissen. *Weighted Automata and Weighted Logics over Tree-like Structures*. PhD thesis, Leipzig University, Germany, 2009. URL: http://www.dr.hut-verlag.de/978-3-86853-180-0.html.

17 Andrzej Mostowski. On direct products of theories. *J. Symbolic Logic*, 17(1):1–31, 1952.

18 Frank P. Ramsey. On a problem of formal logic. *Proc. London Math. Soc.*, 30(1):264–286, 1930.

19 Elena V. Ravve, Zeev Volkovich, and Gerhard-Wilhelm Weber. Effective optimization with weighted automata on decomposable trees. *Optimization*, 63(1):109–127, 2014.

20 Marcel-Paul Schützenberger. On the definition of a family of automata. *Inform. Control*, 4(2–3):245–270, 1961.

21 Saharon Shelah. The monadic theory of order. *Ann. Math.*, 102(3):379–419, 1975.

22 Alan S. Stern, Jan Mycielski, and Pavel Pudlák. *A Lattice of Chapters of Mathematics: Interpretations between Theorems*, volume 426 of *Mem. Amer. Math. Soc.* American Math. Soc., 1990.

Maximum Area Axis-Aligned Square Packings

Hugo A. Akitaya
Tufts University, Medford, MA, USA
hugo.alves_akitaya@tufts.edu
https://orcid.org/0000-0002-6827-2200

Matthew D. Jones
Tufts University, Medford, MA, USA
matthew.jones@tufts.edu

David Stalfa
Northeastern University, Boston, MA, USA
stalfa@ccis.neu.edu
https://orcid.org/0000-0003-2101-8675

Csaba D. Tóth
California State University Northridge, Los Angeles, CA, USA
csaba.toth@csun.edu
https://orcid.org/0000-0002-8769-3190

Abstract

Given a point set $S = \{s_1, \ldots, s_n\}$ in the unit square $U = [0,1]^2$, an *anchored square packing* is a set of n interior-disjoint empty squares in U such that s_i is a corner of the ith square. The *reach* $R(S)$ of S is the set of points that may be covered by such a packing, that is, the union of all empty squares anchored at points in S.

It is shown that $\text{area}(R(S)) \geq \frac{1}{2}$ for every finite set $S \subset U$, and this bound is the best possible. The region $R(S)$ can be computed in $O(n \log n)$ time. Finally, we prove that finding a maximum area anchored square packing is NP-complete. This is the first hardness proof for a geometric packing problem where the size of geometric objects in the packing is unrestricted.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization, Theory of computation → Computational geometry

Keywords and phrases square packing, geometric optimization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.77

Related Version A full version of this paper is available at `https://arxiv.org/abs/1806.09562`.

Funding Research supported in part by the NSF awards CCF-1422311 and CCF-1423615. The first author was supported by the Science Without Borders program.

1 Introduction

Let $S = \{s_1, \ldots, s_n\}$ be a set of n points in the unit square $U = [0,1]^2$. We say that a square q is *empty* if no point in S lies in the interior of q, and q is *anchored* at a point s if one of its four corners is s. An *anchored square packing for* S is a set $Q = \{q_1, \ldots, q_n\}$ of interior-disjoint axis-aligned empty squares that lie in U such that q_i is anchored at s_i for $i = 1, \ldots, n$. A *lower-left* anchored square packing is an anchored square packing in which s_i is the lower-left corner of q_i, for $i = 1, \ldots, n$ [2]. No polynomial-time algorithm is known for

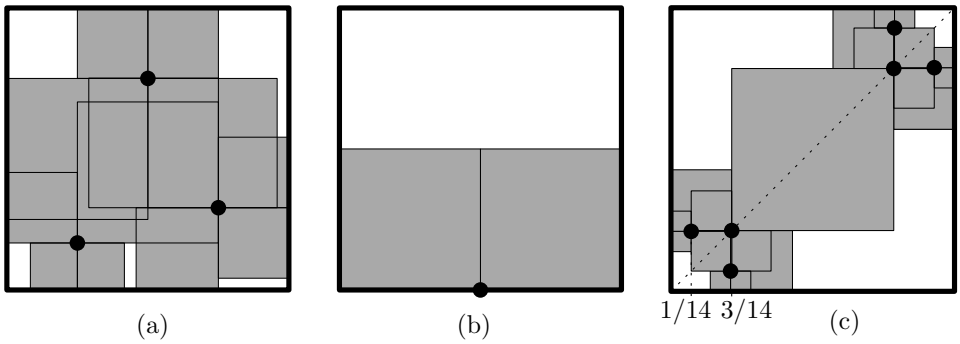


Figure 1 (a) The reach $R(S)$ for a set S of three points. (b) The area of $R(S)$ is $\frac{1}{2}$ for $S = \{(\frac{1}{2}, 0)\}$. (c) The reach $R(S)$ touches all four sides of U, and its area is $\frac{4}{7}$.

computing the maximum area of an anchored square packing for a given point set S; the problem admits a PTAS using a reduction to the maximum weight independent set problem (MWIS) [1]. The empty squares anchored at S do not always cover U entirely (Fig. 1(a)). For finding a maximum anchored square packing for S, it suffices to consider the subset of U that can be reached by anchored empty squares. Specifically, we define the *reach* of S, denoted $R(S)$, as the union of all axis-aligned empty squares contained in U and anchored at some point in S.

For computing the reach $R(S)$, we can take the union of all maximal empty squares anchored at the points in S, as follows. For $i = 1, \ldots, n$, let q_i^1 be the maximal axis-aligned empty square in U whose lower-left corner is s_i, and similarly define q_i^2, q_i^3, and q_i^4 where s_i is the upper-left, upper-right, and lower-right corner, respectively. We say that a point $s \in S$ *blocks* a square q_i^1 if s is incident to the top or right edge of q_i^1. Similarly, s blocks q_i^j if $j = 2$ (resp., 3, 4) and s is incident to the bottom or right edges of q_i^j (resp., bottom or left edges, or top or left edges of q_i^j). It is now clear that $R(S) = \bigcup_{i=1}^{n} \bigcup_{j=1}^{4} q_i^j$.

Summary of Results. We prove that for every finite set $S \subset U$, the area of $R(S)$ is at least $\frac{1}{2}$, and this bound is the best possible (Section 2). This settles in the affirmative a conjecture by Balas et al. [1]. We show how to compute $R(S)$ in $O(n \log n)$ time where $n = |S|$ (Section 3). We also show that finding the maximum area anchored square packing for a given point set S is NP-complete (Section 4). This is the first NP-hardness result for a geometric packing problem, where the size of the geometric objects in the packing is unrestricted. We conclude with related open problems (Section 5).

Motivation and Related Previous Work. Geometric packing and covering problems have a long and revered history, going back to Kepler's problem about the densest packing of congruent balls in Euclidean space. In a classical packing problem, we are given a container region C, and a set O of geometric objects, and we wish to find a maximum subset $O' \subseteq O$ such that congruent copies (or translates) of the objects in O' fit in C without overlap.

Anchored variants, where each geometric object needs to contain a given point (*anchor*) initially emerged in VLSI design, where the anchors represent the endpoints of wires. Allen Freedman [15] conjectured that for every finite set $S \subset [0, 1]^2$, which contains the origin (i.e., $\mathbf{0} \in S$), there is a lower-left anchored rectangle packing of area at least $\frac{1}{2}$. This lower bound would match an easy upper bound construction, where n points are equally distributed on the diagonal. The current best lower bound is 0.091 [6].

More recently, a broad family of anchored packing problems were proposed in the context of *map labelling*, where the anchors represent cities in a map, and axis-aligned rectangles represent labels [7, 8, 9, 10, 12, 13, 16]. Variants of the problem require the anchor to be at a corner, at a side, or anywhere in the rectangle, and the objective is to maximize the number of labels that can be packed in the map. Many of these problems are known to be NP-complete. However, in all previous reductions, the label boxes have a finite number of possible sizes [7, 12, 16] or bounded size [8].

In this paper, we consider the variant of Freedman's problem: We need to place an axis-aligned square at each anchor, and the sizes of the squares are not given in advance. Our objective is to maximize the total area of an anchored square packing. Balas et al. [1] showed that a greedy strategy finds an $\frac{5}{32}$-approximation, and a reduction to MWIS yields a PTAS that achieves an $(1-\varepsilon)$-approximation in time $n^{O(1/\varepsilon)}$. It is known that the number of maximum-area square anchored packings may be exponential in n [2].

2 The Minimum Area of the Reach

In this section, we prove $\text{area}(R(S)) \geq \frac{1}{2}$ for every set S of n points in $U = [0,1]^2$ (Theorem 12). Note that this bound is the best possible for all $n \in \mathbb{N}$. Indeed, if S is the one-element set $S = \{(\frac{1}{2}, 0)\}$, then $\text{area}(R(S)) = \frac{1}{2}$; see Fig. 1(b). By placing n points in an ε-neighborhood of $(\frac{1}{2}, 0)$ in U, we see that for every $\varepsilon > 0$ and every $n \in \mathbb{N}$, there exists a set S of n points in U such that $\text{area}(R(S)) < \frac{1}{2} + \varepsilon$. Note that in this construction all maximal anchored squares are disjoint from the top side of U. Under this constraint, the upper bound $\frac{1}{2}$ is always attained.

We call a point set S *trivial* if $R(S)$ is disjoint from one of the sides of U. The following lemma shows that $\text{area}(R(S)) \geq \frac{1}{2}$ for trivial instances.

▶ **Lemma 1.** *If $R(S)$ does not touch one of the sides of U, then* $\text{area}(R(S)) \geq \frac{1}{2}$.

Proof. Without loss of generality, $R(S)$ does not touch the top side of U. Let $s = (x, y)$ be a point in S with maximum y-coordinate. Consider the maximal empty squares whose lower-left and lower-right corners are at s. Since these squares do not touch the top side of U, and s has maximum y-coordinate, they touch the left and right side of U, respectively. Consequently, their combined area is $x^2 + (1-x)^2 \geq (\frac{1}{2})^2 + (\frac{1}{2})^2 = \frac{1}{2}$. Hence $\text{area}(R(S)) \geq \frac{1}{2}$, as claimed. ◀

▶ Remark. We do not know of any nontrivial point set S that attains the lower bound $\text{area}(R(S)) \geq \frac{1}{2}$. Our best lower bound construction for nontrivial instances yields $\frac{4}{7}$; see Fig. 1(c).

Outline. In the remainder of Section 2, we consider nontrivial instances $S \subset U$. A *gap* is a connected component of $U \setminus R(S)$, i.e., of the complement of the reach. Section 2.1 presents basic properties of $R(S)$ and its gaps, Section 2.2 classifies the possible gaps into five types, and Section 2.3 presents a charging scheme in which we define for every gap C a region $R_C \subset R(S)$ such that $\text{area}(C) \leq \text{area}(R_C)$, and the regions R_C are pairwise interior-disjoint. Summation over all gaps yields $\text{area}(U \setminus R(S)) \leq \sum_C \text{area}(R_C) \leq \text{area}(R(S))$, consequently $\text{area}(R(S)) \geq \frac{1}{2}\text{area}(U) = \frac{1}{2}$.

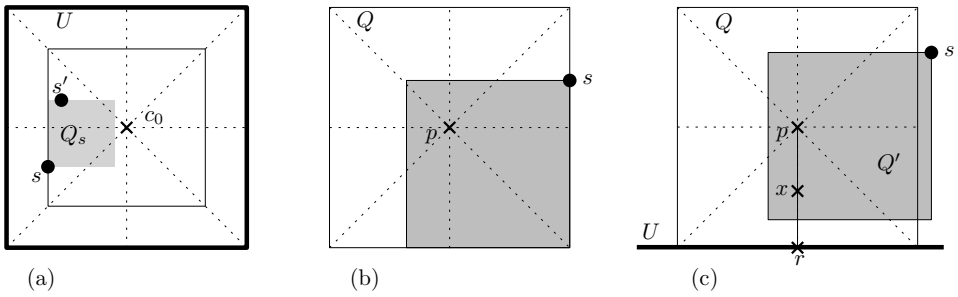


Figure 2 (a) A point $s \in S$ where the anchored square Q_s does not contain c_0. (b) If $s \in \partial Q$, $x(p) \leq x(s)$, and $y(p) \leq y(s)$, then the lower-left square anchored at s contains p. (c) ∂Q intersects the bottom side of U, and r lies below p.

2.1 Properties of the Reach and its Gaps

▶ **Lemma 2.** *For every finite set $S \subset U$, the reach $R(S)$ is connected.*

Proof. Let $S \subset U$ be a finite set, and let $c_0 = (\frac{1}{2}, \frac{1}{2})$ denote the center of U. We show that for each $s \in S$, there is an empty square Q_s anchored at s that contains c_0 **or** whose boundary contains an anchor $s' \in S$ such that $\|s' - c_0\|_\infty < \|s - c_0\|_\infty$ (i.e., s' is closer to c_0 in L_∞ norm than s). This implies that Q_s (hence $R(S)$) contains a line segment from s to c_0 or to s'. Consequently, $R(S)$ contains a polyline from every $s \in S$ to c_0. By the definition of $R(S)$, this further implies that $R(S)$ contains a polyline between any two points in $R(S)$.

It remains to prove the claim. Let $s \in S$. We may assume without loss of generality that $x(s) \leq y(s) \leq \frac{1}{2}$, hence $\|s - c_0\|_\infty = \frac{1}{2} - x(s)$. Let Q_s be the maximal empty square whose lower-left corner is s. Refer to Fig. 2(a). If $c_0 \in Q_s$, then our proof is complete. Otherwise, the side length of Q_s is $a_s < \frac{1}{2} - x(s)$, and there is a point s' in the right or the top side of Q_s. The anchor s' lies in the interior of the L_∞-ball of radius $\frac{1}{2} - x(s)$ centered at c_0, hence $\|s' - c_0\|_\infty < \|s - c_0\|_\infty$, as claimed. ◀

▶ **Lemma 3.** *For every point $p \in U \setminus R(S)$, there exists a point $r \in \partial U$ such that the line segment pr is horizontal or vertical; and $pr \subset U \setminus R(S)$.*

Proof. Let $p \in U \setminus R(S)$, and let Q be the maximal empty axis-aligned square centered at p. Refer to Fig. 2(b). The boundary of this square, ∂Q, intersects S or ∂U, otherwise Q would not be maximal.

First assume that ∂Q contains a point $s \in S$. Without loss of generality, we may assume that $x(p) \leq x(s)$ and $y(p) \leq y(s)$. Since Q is empty, the maximal anchored square with upper-right corner at s contains p, hence $p \in R(S)$, contradicting our assumption that $p \notin R(S)$.

We can now assume that ∂Q intersects ∂U. Without loss of generality, $\partial Q \cap \partial U$ lies in the bottom side of both Q and U. Let $r \in \partial Q \cap \partial U$ be a point vertically below p (see Fig. 2(c) for an example). Suppose that segment pr intersects $R(S)$. Then some point $x \in pr$ lies in a square Q' anchored at a point $s \in S$. Since Q is empty, the anchor s lies outside of Q, and so the side length of Q' is at least half of that of Q, i.e., the side length of Q' is at least $|pr|$. However, then $y(s) \geq |pr|$, and the square Q' contains the segment px, contradicting our assumption that $p \notin R(S)$. Therefore there is no such point $x \in pr$, and $pr \subset U \setminus R(S)$, as claimed. ◀

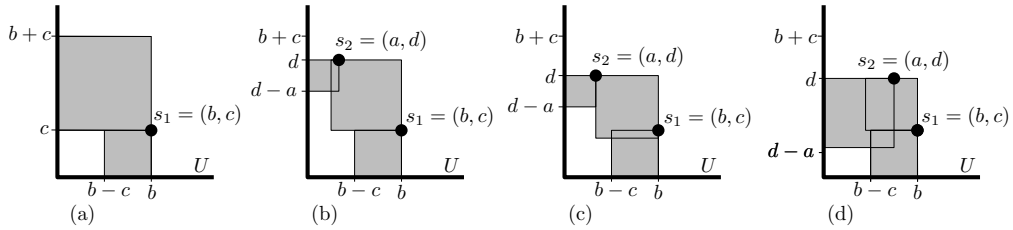


Figure 3 (a) A corner gap of type 1. (b–d) Corner gaps of type 2.

▶ **Corollary 4.** *The reach is simply connected.*

Proof. By Lemma 2, $R(S)$ is connected. Suppose that $R(S)$ is not simply connected. Then there is a gap $C \subset U \setminus R(S)$ such that $\partial C \subset R(S)$. Let $p \in \text{int}(C)$ be an arbitrary point in the interior of C. By Lemma 3, there is a point $r \in \partial U$ such that $pr \subset U \setminus R(S)$, which implies $r \in \partial C$, contradicting our assumption $\partial C \subset R(S)$. Therefore $R(S)$ is simply connected, as required. ◀

2.2 Classification of Gaps

In this section we classify the possible shapes of the gaps in $U \setminus R(S)$ for nontrivial instances. To simplify our analysis, we assume that $S \subset \text{int}(U)$ and no two points in S have the same x- or y-coordinates. This assumption is justified by the following lemma.

▶ **Lemma 5.** *If* $\text{area}(R(S)) \geq \frac{1}{2}$ *for every finite point set* $S \subset U$ *such that* $S \subset \text{int}(U)$ *and no two points in* S *have the same* x- *or* y*-coordinates, then* $\text{area}(R(S)) \geq \frac{1}{2}$ *for every finite point set* $S \subset U$*.*

Proof. Let $S \subset U$ be a finite point set that contains a point in ∂U or two points with the same x- or y-coordinate. Let ε_0 be minimum positive difference between x- and y-coordinates of points in S. For every $\varepsilon \in (0, \varepsilon_0/2)$, translate each point in S by a random vector of length at most ε into $\text{int}(U)$. The resulting point set S_ε lies in $\text{int}(U)$ and have distinct x- and y-coordinates with probability 1; the side length of each maximal anchored square may increase by at most 2ε, but could decrease substantially. Consequently, $\text{area}(R(S_\varepsilon)) \leq \text{area}(R(S)) + 4n\varepsilon$, hence $\lim_{\varepsilon \to 0} \text{area}(R(S_\varepsilon)) \leq \text{area}(R(S))$. ◀

We distinguish a *corner gap*, which is incident to a corner of U; and a *side gap*, which is adjacent to exactly one side of U. We show that every gap is bounded by ∂U and by squares anchored at up to three points in S. We define five types of gaps (two types of corner gaps and three types of side gaps). Each type is defined together with an empty rectangle $B \subset U$ and 1–3 anchors on the boundary of B. In each case, the gap is determined by the maximal empty squares that lie entirely in B and are anchored at points in $S \cap B$.

We describe each type modulo the symmetry group of U (i.e., the dyhedral group D_4). Specifically, we restrict ourselves to corner gaps incident to the lower-left corner of U, and side gaps adjacent to the bottom side of U. Reflection in the line $x = y$ (resp., $x = \frac{1}{2}$) maintains corner gaps incident to the origin (resp., side gaps along the bottom side of U); and we describe only one variant modulo reflection.

1. Let $0 < c < b < 1$. If $B = [0, b] \times [0, b + c]$ is empty and $s_1 = (b, c) \in S$, then the squares anchored at s_1 form a corner gap $[0, b - c] \times [0, c]$. See Fig. 3(a).

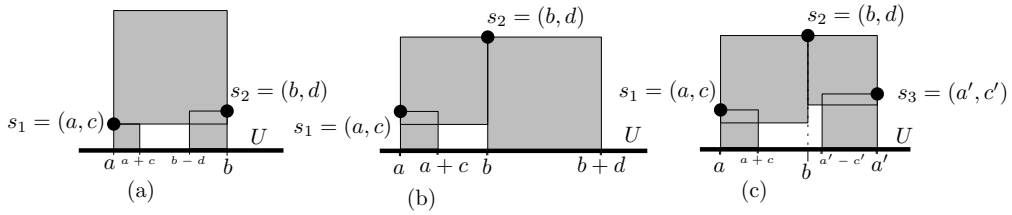


Figure 4 (a–c) Side gaps of type 3, 4, and 5, respectively.

2. Let $0 < a < b < 1$ and $0 < c < d < 1$ such that $c < b$ and $d < b + c$. If $B = [0, b] \times [0, d]$ is empty and $s_1 = (b, c), s_2 = (a, d) \in S$, then the squares anchored at s_1 and s_2 form a corner gap $[0, b - c] \times [0, \min(c, d - a, d - b + a)] \cup [0, \min(a, b - c, b - d + c)] \times [0, d - a]$. See Fig. 3(b–d).
3. Let $0 < a < b < 1$ and $0 < c, d < 1$ with $\max(c, d) < b - a$. If $B = [a, b] \times [0, \min(c, d) + (b - a)]$ is empty and $s_1 = (a, c), s_2 = (b, d) \in S$, then the squares anchored at s_1 and s_2 form a side gap $[a + c, b - d] \times [0, \min(c, d)]$. See Fig. 4(a).
4. Let $0 < a < b < 1$ and $0 < c < d < 1$ with $b - a < d$. If $B = [a, b + d] \times [0, d]$ is empty and $s_1 = (a, c), s_2 = (b, d) \in S$, then the squares anchored at s_1 and s_2 form a side gap $[a + c, b] \times [0, \min(c, d - b + a)]$. See Fig. 4(b).
5. Let $0 < a < b < a' < 1$ and $0 < c < c' < d < 1$ with $b - a < d$ and $a' - b < d$. If $B = [a, a'] \times [0, d]$ is empty and $s_1 = (a, c), s_2 = (b, d), s_3 = (a', c') \in S$, then the squares anchored at s_1, s_2, and s_3 form a side gap $[a + c, \min(b, a' - d + c')] \times [0, \min(c, d - b + a)] \cup [\min(b, a' - d + c'), a' - c'] \times [0, \min(c', d - a' - b)]$. See Fig. 4(c) for an example.

▶ **Lemma 6.** *Every gap C of type 1–5 is disjoint from all empty squares that are anchored at points in S and lie in the exterior of the defining box B of C. Consequently, C is bounded by ∂U and some empty squares anchored at points in $S \cap B$.*

Proof. In each of the five cases, $\partial B \cap \mathrm{int}(U)$ is covered by empty squares anchored at the points in S that define C. More precisely, each point in $\partial B \cap \mathrm{int}(U)$ lies in an empty square anchored at a point in $S \cap \partial B$ blocked by some point in $S \cap \partial B$ or $\partial U \cap \partial B$. For $s_i \in S$ lying in the exterior of B, let Q_i be a square anchored at s_i. If Q_i intersects B, then its interior intersects $\partial B \cap \mathrm{int}(U)$, hence it intersects a square Q_j anchored at some $s_j \in \partial B$ and blocked by some point $p_j \in S \cap \partial B$ or $\partial U \cap \partial B$. Since $\mathrm{int}(Q_i)$ contains neither s_j nor p_j, we have $Q_i \cap B \subset Q_j$, and so Q_i is disjoint from the gap C, as claimed. ◀

We prove the following classification result for the gaps in the full paper.

▶ **Lemma 7.** *Every gap of a nontrivial instance is of one of the five types defined above.*

It is now easy to check that the following properties hold for all five types of gaps.

▶ **Corollary 8.**

(i) *Each gap is either a rectangle incident to a side of U, or the union of two rectangles incident to the same side of U (which we call an* L-shaped *gap).*

(ii) *Every edge uv of a gap is contained in either ∂U or a maximal anchored rectangle. Consequently, the square built on the side uv outside of the gap lies either outside of U or in $R(S)$.*

We say that a point $p \in \partial U$ is a *lead* if it is a vertex of a maximal anchored square q_i^j, and ps_i is a diagonal of q_i^j. We observe that one or two vertices of a gap along ∂U is a lead.

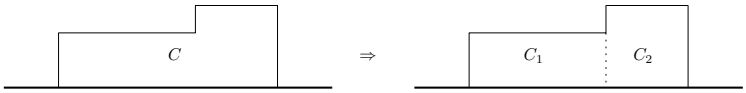


Figure 5 An L-shaped side gap C is subdivided into two rectangles $C_1, C_2 \in \mathcal{C}^*$.

▶ **Corollary 9.**

- *If C is a rectangular gap, then at least one endpoint of $C \cap \partial U$ is a lead,*
- *otherwise both endpoints of $C \cap \partial U$ are leads.*

2.3 Charging Scheme

For every gap C, we define a region $R_C \subset R(S)$; and then we show that $\text{area}(R_C) \geq \text{area}(C)$ and the regions R_C are pairwise interior-disjoint.

For ease of exposition, we subdivide every L-shaped side gap C into two rectangles $C = C_1 \cup C_2$, then define interior-disjoint regions R_{C_1} and R_{C_2}, and let $R_C := R_{C_1} \cup R_{C_2}$. Specifically, let $\mathcal{C}^*$ be a set of regions that contains: (1) all corner gaps, (2) all rectangular side gaps, and (3) for each L-shaped side gap C, the two interior-disjoint rectangles C_1 and C_2, such that $C = C_1 \cup C_2$ and both C_1 and C_2 have a common side with ∂U (see Fig. 5 for an example). By Corollary 9, at least one vertex of every rectangle in $\mathcal{C}^*$ is a lead, and two vertices of every L-shaped corner gap in $\mathcal{C}^*$ are leads.

We are now ready to define a region R_C for each region $C \in \mathcal{C}^*$.

- Let $C = (a, b, c, d)$ be a rectangle in $\mathcal{C}^*$. Assume w.l.o.g. that bc is contained in the bottom side of U, and c is a lead (a symmetric construction applies if bc is contained in another side of U or b is the only lead). Refer to Fig. 6(a). Let ℓ_1 and ℓ_2 be lines of slope 1 passing through a and c, respectively. Let p_1 be the intersection of ℓ_1 with the vertical line through cd, and let p_2 be the intersection of ℓ_2 with the horizontal line through da. Let z_1 (resp., z_2) be the intersection point of ℓ_2 (resp., ℓ_1) with the line of slope -1 passing through p_1 (resp., p_2). Then R_C is the smaller pentagon out of (a, d, c, z_1, p_1) and (a, d, c, p_2, z_2).
- Let $C = (a, b, c, d, e, f)$ be a L-shaped corner gap in $\mathcal{C}^*$. Assume w.l.o.g. that b is the lower-left corner of U. By Corollary 9, both a and c are leads. Refer to Fig. 6(b). Let ℓ_1 and ℓ_2 be lines of slope 1 passing through a and c, respectively. Let $p_1, p_2 \in S$ be the anchors on ℓ_1 and ℓ_2, respectively (which exist since both a and c are leads). Let z_1 (resp., z_2) be the intersection point of line ℓ_2 (resp., ℓ_1) with the line of slope -1 passing through p_1 (resp., p_2). Then R_C is the smaller heptagon out of $(a, f, e, d, c, z_1, p_1)$ and $(a, f, e, d, c, p_2, z_2)$.

▶ **Lemma 10.** *For every $C \in \mathcal{C}^*$, we have:*

(P1) $R_C \subseteq R(S)$,

(P2) $\text{int}(R_C)$ *does not contain any anchors, and*

(P3) $\text{area}(C) \leq \text{area}(R_C)$.

Proof. The region $C \in \mathcal{C}^*$ is either a gap or a rectangle within an L-shaped side gap; Fig. 6(b). Let C^* be the gap that contains C, and B the box defining the gap C^*. For all five types of gaps, $\text{int}(B)$ does not contain any anchor, and $B \setminus C^* \subset R(S)$. By Corollaries 8(ii) and 9, the points p_1 and p_2 lie in B. Consequently, $R_C \subset B$, hence $R_C \subset B \setminus C^*$. This confirms (P1) and (P2).

To prove (P3), we distinguish two cases. First assume that C is an $x \times y$ rectangle. Let T be an isosceles right triangle whose hypotenuse has length $x + y$. It is easy to check that $\text{area}(C) \leq \text{area}(T)$. Indeed, $\text{area}(T) = \left(\frac{1}{2}(x+y)\right)^2 = \frac{1}{2}\left(\frac{x^2}{2} + xy + \frac{y^2}{2}\right) \geq xy = \text{area}(C)$.

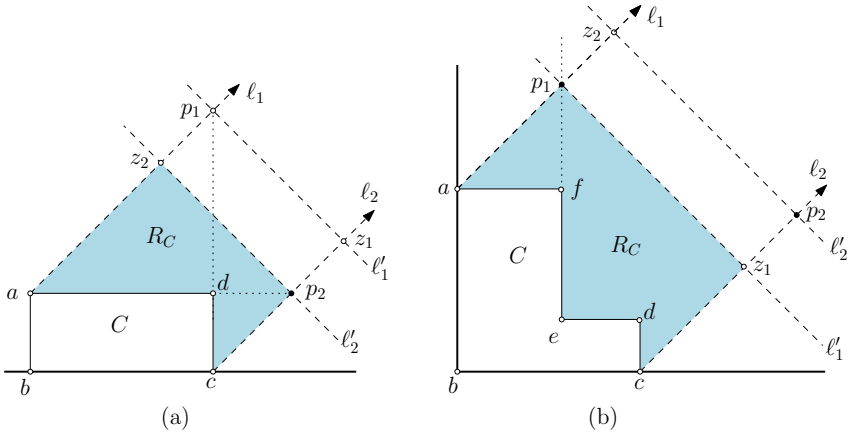


Figure 6 Region R_C. (a) $C \in \mathcal{C}^*$ is a rectangle. (b) $C \in \mathcal{C}^*$ is an L-shaped corner gap.

By definition, R_C contains a triangle congruent to T, consequently $\text{area}(C) \leq \text{area}(T) \leq \text{area}(R_C)$, as claimed.

Next assume that $C \in \mathcal{C}^*$ is an L-shaped corner gap; Fig. 6(b). Assume that C is formed by three interior-disjoint axis-aligned rectangles defined by diagonals ae, be, and ce. Let their dimensions respectively be $x \times y$, $x \times z$, and $w \times z$. Let T_1 and T_2 be isosceles right triangles whose hypotenuses are of length $x + y$ and $w + z$, respectively. Let T_3 and T_4 be isosceles right triangles whose legs are of length x and z, respectively. By definition, R_C contains interior-disjoint triangles congruent to T_1, T_2, T_3, and T_4: the hypotenuses of the respective triangles are in the same supporting lines as ef, ed, ap_1, and cp_2 respectively. Using the same argument as in the previous case, we can show that $\text{area}(T_1)$ and $\text{area}(T_2)$ are, respectively, greater or equal than the areas of the $x \times y$ and $w \times z$ rectangles. It remains to show that $\text{area}(T_3) + \text{area}(T_4)$ is greater or equal than the area of the $x \times z$ rectangle. By definition, we have $\text{area}(T_3) + \text{area}(T_4) = \left(\frac{x^2}{2} + \frac{z^2}{2}\right) \geq xz$ for all $x, z > 0$. ◀

We prove that the regions R_C, $C \in \mathcal{C}^*$, are pairwise interior-disjoint in the full paper.

▶ **Lemma 11.** *For every two regions $C, C' \in \mathcal{C}^*$, $C \neq C'$, we have $\text{int}(R_C) \cap \text{int}(R_{C'}) = \emptyset$.*

▶ **Theorem 12.** *For every finite set $S \subset U$, we have $\text{area}(R(S)) \geq \frac{1}{2}$, and this bound is the best possible.*

Proof. By Lemma 5, it suffices to prove the lower bound when $S \subset \text{int}(U)$ and no two points in S have the same x- or y-coordinate. For all gaps $C \subset U \setminus R(S))$, we have defined interior-disjoint regions $R_C \subset R(S)$ such that $\text{area}(C) \leq \text{area}(R_C)$. Consequently, $\sum_C \text{area}(C) \leq \sum_C \text{area}(R_C) \leq \text{area}(R(S))$, which immediately yields $\text{area}(R(S)) = 1 - \sum_C \text{area}(C) \geq 1 - \text{area}(R(S))$, and $\text{area}(R(S)) \geq 1/2$, as claimed. This bound is the best possible: the point set $S = \{(\frac{1}{2}, 0)\}$ attains $\text{area}(R(S)) = \frac{1}{2}$. ◀

3 Algorithm for Computing the Reach

In this section, we show how to compute efficiently the reach of a given point set.

▶ **Theorem 13.** *For a set $S \subset U$ of n points, $R(S)$ can be computed in $O(n \log n)$ time.*

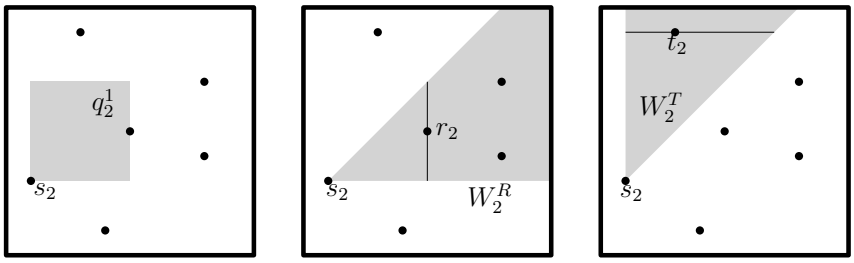


Figure 7 A set S of 6 anchors in $U = [0,1]^2$. (a) The maximal empty anchored square q_2^1; (b) wedge W_2^R; and (c) wedge W_2^T.

Recall that, for a set $S = \{s_1, \ldots, s_n\}$, the reach is defined as a union of $4n$ squares, $R(S) = \bigcup_{i=1}^{n} \bigcup_{j=1}^{4} q_i^j$, where q_i^1 is the maximal axis-aligned empty square in U whose lower-left corner is s_i, and q_i^2, q_i^3, and q_i^4 are defined similarly where s_i is the upper-left, upper-right, and lower-right corner, respectively. Since any two squares cross in at most two points, the $4n$ squares q_i^j ($i = 1, \ldots, n$ and $j = 1, \ldots, 4$) form a pseudo-circle arrangement. It is well known that the union of $O(n)$ pseudo-circles has $O(n)$ vertices [11]. The union of $4n$ axis-aligned *squares* can be computed by a sweep-line algorithm in $O(n \log n)$ time [3].

We note that Bentley's sweep-line algorithm can compute the *area* of the union of n axis-aligned *rectangles* in $O(n \log n)$ time (without computing the union itself, which may have $\Theta(n^2)$ complexity). Computing the volume of the union of axis-aligned hyper-rectangles in $\mathbb{R}^d$ is known as *Klee's measure problem*, and the current best algorithms [4] for $d \geq 3$ run in $O(n^{d/2})$ time in general, and in $O(n^{(d+1)/3} \operatorname{polylog}(n))$ time for hypercubes (see also [17]).

It remains to compute the $4n$ anchored maximal empty squares q_i^j. We focus on the n *lower-left* anchored squares q_i^1 ($i = 1, \ldots, n$), the other three types can be computed analogously. For every $i = 1, \ldots, n$, the lower-left corner of q_i^1 is s_i, and its left *or* top side contains another anchor or a point in ∂U; we say that this point is the *blocker* of q_i^1. For each i, we find a first point that may block the square q_i^1 on the left and on the top side, independently. The blocker of q_i^1 is the points closest to s_i in L_∞ norm. We continue with the details. We define two wedges with apex at the origin: Let $W^L = \{(x, y) \in \mathbb{R}^2 : 0 < y < x\}$ and $W^T = \{(x, y) \in \mathbb{R}^2 : 0 < x < y\}$; see Fig. 7(b–c). The Minkowski sums $W_i^L := s_i + W_L$ and $W_i^T := s_i + W^T$ are the translates of these wedges with apex at s_i. Let r_i be a point of minimum x-coordinate in $W_i^L \cap (S \cup \partial U)$; and let t_i be a point of minimum y-coordinate in $W_i^T \cap (S \cup \partial U)$. Then the blocker of q_i^1 is either r_i or t_i, whichever is closer to s_i in L_∞ norm.

For every $i = 1, \ldots, n$, we find points r_i and t_i, independently. Consider the points $r_i \in W_i^R$, for $i = 1, \ldots, n$ (the case of the points $t_i \in W_t^T$ is analogous). We use a data structure originally developed for computing Θ-graphs in the context of geometric spanners by Narasimhan and Smid [14, Section 4.1.2]. They developed the following dynamic data structure for n points in the plane:

▶ **Lemma 14** ([14], Lemma 4.1.9). *Let H be a nonvertical line through the origin. There is a data structure that maintains a set P of n points in the plane and supports the following queries: (i)* MinBelow(p)*: Given a query point $p \in P$, compute a point with the minimum x-coordinate among all points in P that are below $p + H$; (ii) insert a point into P; (iii) delete a point from P. The data structure has $O(n)$ space, $O(n \log n)$ preprocessing time, and $O(\log n)$ query time.*

▶ **Corollary 15.** *Given a point set $S = \{s_i : i = 1\ldots, n\} \subset U$, the points r_i and t_i $(i = 1, \ldots, n)$ can be computed in $O(n \log n)$ time. Consequently, the squares q_i^1 can also be computed in $O(n \log n)$ time.*

Proof. Assume that S is sorted in decreasing order by their y-coordinates. We use the data structure in Lemma 14 with the line $H : y = x$ as follows. Initially $P = \emptyset$. For $i = 1, \ldots, n$, we insert s_i into P. If MinBelow(s_i) returns a point in P, then let this be r_i, otherwise let r_i be the point in the right side of U that has the same y-coordinate as s_i. Since P contains all points in S whose y-coordinates are greater or equal to that of s_i, if wedge W_i^R contains any anchor, then MinBelow(s_i) returns one with the minimum x-coordinate. This shows that r_i is computed correctly for all $i = 1, \ldots, n$.

The points t_i $(i = 1, \ldots, n)$ can be computed analogously in $O(n \log n)$ time. In $O(1)$ additional time for each $i = 1, \ldots, n$, we can compare r_i and t_i, find the blocker of q_i^1, and determine the maximal anchored square q_i^1. ◀

Proof of Theorem 13. By a repeated application of Corollary 15, we can compute all $4n$ anchored squares q_i^j $(i = 1, \ldots, n;\ j = 1, \ldots, 4)$. As noted above, a sweep-line algorithm can compute the union $R(S) = \bigcup_{i=1}^{n} \bigcup_{j=1}^{4} q_i^j$ in $O(n \log n)$ time. This completes the proof. ◀

4 NP-Hardness of Maximum-Area Anchored Square Packings

We now prove that the maximum-area anchored square packing problem is NP-complete. We define the decision version of the problem as follows. Instead of the unit square $[0, 1]^2$, we use the square $U = [0, W]^2$, for some integer $W > 0$. For a finite set $S \subset [0, W]^2$ of anchors with integer coordinates, we ask whether there is an anchored square packing of area W^2.

We prove NP-hardness by a reduction from Planar-Monotone-3SAT (described below). For every instance of Planar-Monotone-3SAT, we construct an instance $S \subset [0, W]^2$. We say that an anchored empty square is *forced* if every packing of area W^2 contains it. An anchor in S is *forced* if it is the anchor of a forced square; otherwise it is *free*. A forced square A and its anchor $s \in S$ form a *forced pair* (A, s). We construct an instance in which most of the anchors are forced, and a small number of anchors encode the truth value of the variables in a 3SAT instance.

To prove that the two instances are equivalent, we shall argue that a set of squares and anchors are forced. In an intermediate step, we assume that $\mathcal{F} = \{(A_i, s_i) : i = 1\ldots, f\}$ is a set of forced square-anchor pairs, and we would like to show that another square-anchor pair (A, s) is also forced. Let $P = U \setminus \bigcup_{i=1}^{f} A_i$ be the complement of the forced squares in $\mathcal{F}$. By construction, P is an orthogonal polygon (possibly with holes). An anchor s is *undecided* if there is no forced pair (A, s) anchored at s in $\mathcal{F}$ (i.e., s is either free or its forced pair is not in $\mathcal{F}$).

We show (Lemma 16) that the following two properties each imply that the pair (A, s) is forced (given that all pairs in $\mathcal{F}$ are forced). We define both properties for the orientations shown in Fig. 8, but they generalize to all other orientations obtained through the symmetry group of U. Let $s \in S$, and let A be a maximal empty square anchored at s such that $\text{int}(A) \subset P$. Without loss of generality, assume that s is the upper-right corner of A.

1. The lower-left corner of A is a convex vertex of P and there is no undecided anchor in the closure of the bottom and left edges of A.

Figure 8 Identifying forced points. (a) Property 1. (b) Property 2.

2. The lower-left corner of A is a convex vertex of P, the side length of A is greater than 1, and the bottom edge of A is contained in ∂P. There is a unique undecided anchor $s' \in \partial A$ located one unit above the lower-left corner of A. There is no undecided anchor one unit to the right of s or to the right of the lower-right corner of A.

▶ **Lemma 16.** *Given a set of forced pairs $\mathcal{F} = \{(A_i, s_i) : i = 1 \ldots, f\}$, if a pair (A, s) has properties 1 or 2, then (A, s) is a forced pair.*

Proof. Suppose, to the contrary, that there is a anchored square packing Q of area W^2 that does not use the square A anchored at s. Let $B \subseteq A$ be the unit square incident to the lower-left corner of A. Since all anchors have integer coordinates, every empty square containing B is also contained in A. If a pair (A, s) has property 1, apart from A, no such empty square has a corner at an undecided anchor and, hence, B cannot be covered. If a pair (A, s) has property 2, B must be covered by a square anchored at its upper-left corner, which is undecided by hypothesis. Hence, $B \in Q$. Let B' be a unit square with integer coordinates to the right of B. Then the maximal empty square in $P \setminus B$ containing B' satisfies property 1, but that there is no point at its upper-right corner. In this case, B' is not covered. ◀

▶ **Theorem 17.** *It is NP-hard to compute the maximum area anchored square packing of a given set S of n anchors with integer coordinates in a square $U = [0, W]^2$.*

Proof. We reduce from Planar-Monotone-3SAT which is NP-complete [5]. An instance of such problem consists of a boolean formula Φ in 3CNF with n variables $\{x_1, \ldots, x_n\}$ and m clauses, and a planar rectilinear drawing of the a bipartite graph of Φ. The drawing given by an Planar-Monotone-3SAT instance represents variables and clauses by rectangles, and edges by vertical line segments. It has the additional property that the rectangles of variables (and only variables) intersect the line $y = 0$ and the rectangles of clauses lies in the upper (resp., lower) half-plane contain only positive (resp., negative) literals. A literal is called *negative* if it is the negation of a variable, and *positive* otherwise. We need to decide whether we can satisfy all m clauses, each of which is a disjunction of three literals.

For a given instance of Planar-Monotone-3SAT, we construct an instance $S \subset [0, W]^2$ of the maximal area anchored square packing problem, and then show that the two instances are equivalent. We first modify the rectilinear graph of the Planar-Monotone-3SAT instance in the following way. Replace each rectangle by a cycle along its boundary and denote by G the resulting geometric graph. Delete the left, right, and the top (resp., bottom) edges of the rectangles representing positive (resp., negative) clauses. Each clause is now represented by a horizontal segment (a path of length 2 in G). We designate the middle vertex of this path, which has degree 3, as a *clause vertex*. For each cycle in G that represents a variable, delete the right vertical edges, and designate the left vertical edge as a *variable edge*. All remaining edges in G called *wires* and all remaining vertices of degree 3 are called *split vertices*. We orient the wires such that they form directed paths from the variable edges to clause vertices. Assume that the feature size of the resulting rectilinear graph is 1 and the

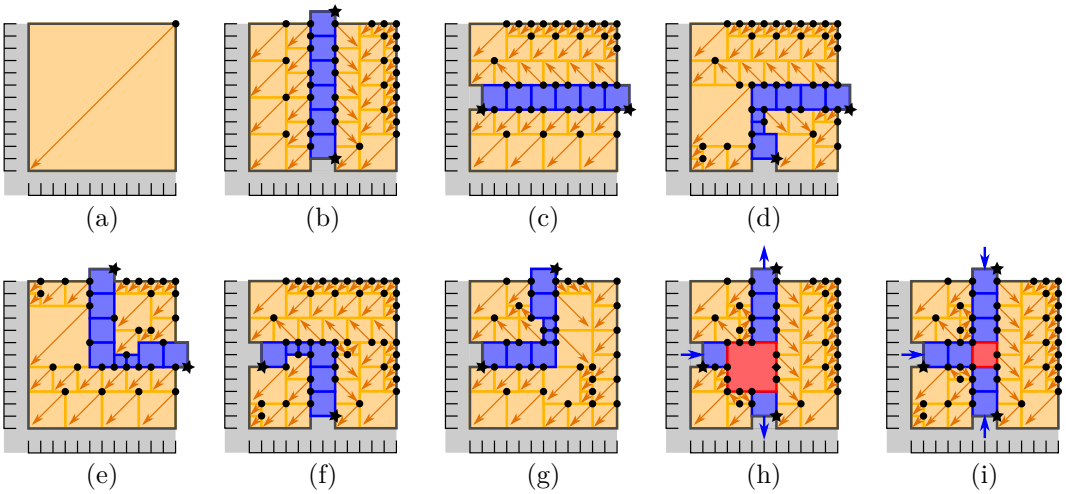


Figure 9 Gadgets. (a) is a *filler* gadget, (b–c) are *wire* gadgets, (d–g) are *turn* gadgets, (h) is a split gadget, and (i) is a *clause* gadget. The rhombus in (h) represents 2 anchors placed at the same position.

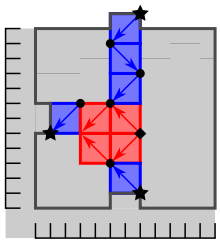


Figure 10 Square packing for the split gadget connected to negative wires.

side length of a minimum enclosing axis-aligned square is k. We set $W = 48k + 48$ and let $U = [0, W]^2$. Scale up the drawing by a factor of 48 and place it in U so that every vertex is at distance at least 24 from ∂U.

We tile U with orthogonal polygons. Every tile is congruent to one of the tiles shown in Fig. 9. We call these tiles *gadgets*: (a) is a *filler* gadget, (b–c) are *wire* gadgets, (d–g) are *turn* gadgets, (h) is a split gadget, and (i) is a *clause* gadget. The filer gadget is a 12×12 square, all other tiles are constructed from a 12×12 square by possibly adding or deleting 1×2 rectangular features in two or three side of the squares. In a tiling of $[0, W]^2$, each such feature matches a feature of an adjacent tile. Choose a tile for each variable that contains part of the variable edge and add the anchors shown in Fig. 9(b) (only the star contained in the 12×12 square is added). Do the same for split and clauses using the tiles shown in Fig. 9(h) and (i), respectively. Connect the gadgets as they are connected in the original drawing using wires and turns. The directions of the wires attached to split and clause gadgets are indicated by blue arrows in Fig. 9. For all remaining tiles, we use filler gadgets with one anchor. This completes the description of the instance $S \subset [0, W]^2$.

We now prove that a Planar-Monotone-3SAT instance admits a positive solution if and only if the corresponding point set $S \subset [0, W]^2$ admits an anchored square packing of area W^2.

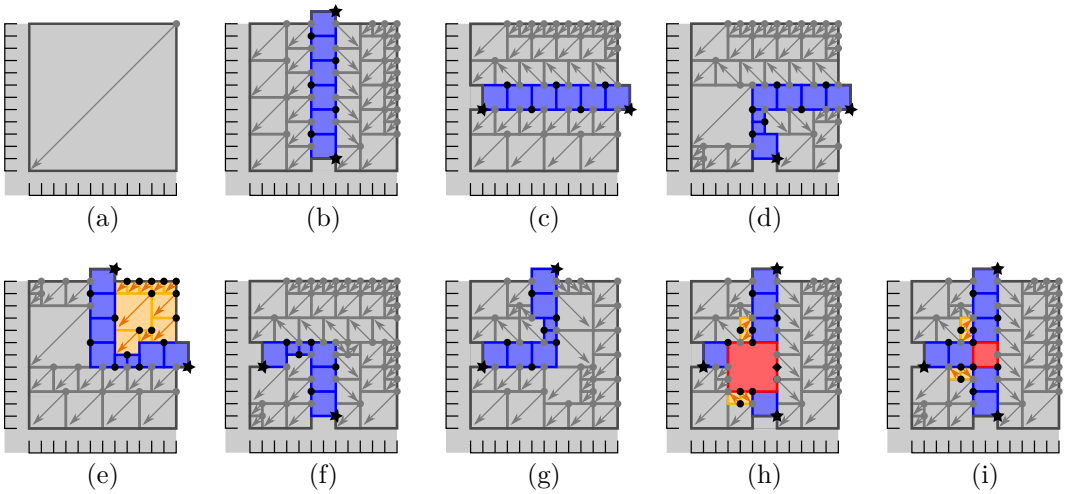


Figure 11 Forced squares and anchors.

Assume that the Planar-Monotone-3SAT instance admits a positive solution. We show that the corresponding instance $S \subset [0, W]^2$ admits an anchored square packing of area W^2. Choose every orange square in each gadget assigning its anchor as the origin of the arrow contained in it (as shown in Fig. 9). For each of the n wire gadgets placed on variable edges, if the corresponding variable is assigned `true` (resp., `false`) add all blue squares assigning its anchor as the only point on its bottom (resp., top) edge. Every connected component formed by blue squares represents a path of wires. We say that a wire is *positive* if it is in the upper half of U and its corresponding variable is assigned `true`, or if it is in the lower half of U and its corresponding variable is assigned `false`. A wire is *negative* otherwise. If a wire is positive (resp., negative), assign the corner that is behind (resp., ahead of) the blue square as its anchor, considering the direction of the wire. For split gadgets connected to positive wires assign the top-left corner of the red square as its anchor. For negative wires, the red square is reached by four equal squares as shown in Fig. 10. Since there exists at least one positive wire connected to a clause vertex, there will be at least one point in a corner of the red square in the clause gadget that has not yet been assigned a square. We complete the square packing by adding such a square with a corresponding anchor. Since the anchored squares cover all gadgets, the overall area of the square packing is W^2.

Assume that the anchored square packing instance $S \subset [0, W]^2$ admits a positive solution (of area W^2). Recall that $[0, W]^2$ is tiled with gadgets. Sort them in lexicographic order by the coordinates of their lower-left corners (i.e., the first gadget is incident to the origin). We use Lemma 16 to prove the following property for each gadget:

> **Property (i).** If the left and bottom boundaries are part of the perimeter of a forced polygon P and contain no free anchor relative to P except for the points shown by a star, then (i.a) every orange and blue square shown in Fig. 9 in the corresponding gadget is forced; and (i.b) if P' is the union of P and the orange and blue squares inside the gadget, then there is no free anchor relative to P' on the boundary of the gadget except for points shown by a star.

Initially, in every gadget, we can determine at least one pair of a forced square and a corresponding forced anchor using Lemma 16. Fig. 11 shows the result of recursively adding a forced square into the forced polygon, and applying Lemma 16 to another pair until there

are no more forced pairs in the gadget. We now show that all blue squares are forced. In each case, we can take the lower-left blue square and conclude that if the square packing covers it entirely, then it is covered by a square anchored at one of its corners. After we add this square to the forced polygon P, the same argument holds for every lower-left blue square not in P. Consequently, all blue squares are forced. The remaining orange squares are forced by recursively applying Lemma 16. Then, if property (i) is satisfied, every gadget satisfies (i.a) and (i.b). Property (i) is trivially satisfied for the lower-left gadget and inductively satisfied by assuming that all gadgets to the left and below satisfy (i.a) and (i.b).

We now show how to convert a square packing of area W^2 into a solution of the Planar-Monotone-3SAT instance. Wire gadgets have two points indicates by a star: one that is ahead and one behind using the direction of the wire (recall that the direction points from the variable edge to the clause). A wire gadget that does not use the star that is ahead in its direction as an anchor for one of the squares contained in it is called *positive*. A wire gadget is called *negative* otherwise. For all wire gadgets satisfying (i.a) and (i.b), if a star is not used as an anchor for a square in the gadget, then the other point marked by a star must be used as anchor in this gadget. This implies that, for a pair of adjacent wire gadgets, if the one ahead in the wire direction is positive, so is the other gadget. Now assume that one of the outputs of the split gadget satisfying (i.a) and (i.b) is connected to a positive gadget. A point in the middle of an edge of the red square in Fig. 11(h) must be used as an anchor of a blue square. Then, the only way to cover all the red area is to use a single square anchored at its upper-left corner. Therefore, the wire connected to the input of the split gadget must also be positive. Finally, assume that the red square in a clause gadget that satisfies (i.a) and (i.b) (see Fig. 11(i)) is covered. Then, it must be anchored at one of its corners. If it is anchored at the upper-left (resp., bottom-left, bottom-right) corner, then the star at the top (resp., left, bottom) of the gadget is used as an anchor of a blue square in this gadget. Therefore, it must be adjacent to a positive wire. Combining all arguments, we set a variable `true` if its first wire gadget (that was placed on the variable edge) is positive and `false` otherwise, and then this assignment will satisfy the boolean formula of the Planar-Monotone-3SAT instance. ◀

5 Open Problems

We have shown that at least half of the area of the unit square $U = [0, 1]^2$ can be reached by empty squares anchored at S for any finite set $S \subset U$, and this bound is the best possible. We have also given the first NP-hardness proof for a packing problem over geometric objects of arbitrary sizes. Our results raise several intriguing open problems. Does our result generalize to higher dimensions, that is, is there a lower bound for the maximal volume covered by empty hypercubes anchored at a finite set of points in $[0, 1]^d$ for $d > 2$? Axis-aligned squares are balls in L_∞-norm: Over all finite sets S of anchors in a unit-diameter ball U in L_p-norm, $p \geq 1$, what is the maximum area of a packing of L_p-balls that each contain an anchor? Is there a polynomial-time algorithm for computing the minimum area *lower-left anchored* square packing for a given set S of n points in the unit square $[0, 1]^2$? Is it NP-hard to compute the maximum area anchored *rectangle* packing of a given set $S \subset [0, 1]^2$? For the last two problems, simple greedy strategies achieve constant-factor approximations [6], and a QPTAS is available for rectangles and a PTAS for squares [1].

References

1 Kevin Balas, Adrian Dumitrescu, and Csaba D. Tóth. Anchored rectangle and square packings. *Discrete Optimization*, 26:131–162, 2017. doi:10.1016/j.disopt.2017.08.003.

2 Kevin Balas and Csaba D. Tóth. On the number of anchored rectangle packings for a planar point set. *Theor. Comput. Sci.*, 654:143–154, 2016. doi:10.1016/j.tcs.2016.03.007.

3 Jon L. Bentley. Solutions to Klee's rectangle problems. unpublished manuscript, 1977.

4 Timothy M. Chan. Klee's measure problem made easy. In *Proc. 54th Annual IEEE Symposium on Foundations of Computer Science*, pages 410–419, 2013. doi:10.1109/FOCS.2013.51.

5 Mark de Berg and Amirali Khosravi. Optimal binary space partitions for segments in the plane. *Int. J. Comput. Geometry Appl.*, 22(3):187–206, 2012. URL: http://www.worldscinet.com/doi/abs/10.1142/S0218195912500045.

6 Adrian Dumitrescu and Csaba D. Tóth. Packing anchored rectangles. *Combinatorica*, 35(1):39–61, 2015. doi:10.1007/s00493-015-3006-1.

7 Michael Formann and Frank Wagner. A packing problem with applications to lettering of maps. In Robert L. Scot Drysdale, editor, *Proc. 7th Annual Symposium on Computational Geometry*, pages 281–288. ACM, 1991. doi:10.1145/109648.109680.

8 Claudia Iturriaga and Anna Lubiw. Elastic labels around the perimeter of a map. *J. Algorithms*, 47(1):14–39, 2003. doi:10.1016/S0196-6774(03)00004-X.

9 Joo-Won Jung and Kyung-Yong Chwa. Labeling points with given rectangles. *Inf. Process. Lett.*, 89(3):115–121, 2004. doi:10.1016/j.ipl.2003.09.017.

10 Konstantinos G. Kakoulis and Ioannis G. Tollis. Labeling algorithms. In Roberto Tamassia, editor, *Handbook on Graph Drawing and Visualization.*, pages 489–515. Chapman and Hall/CRC, 2013. URL: https://www.crcpress.com/Handbook-of-Graph-Drawing-and-Visualization/Tamassia/9781584884125.

11 Klara Kedem, Ron Livne, János Pach, and Micha Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete & Computational Geometry*, 1:59–70, 1986. doi:10.1007/BF02187683.

12 Donald E. Knuth and Arvind Raghunathan. The problem of compatible representatives. *SIAM J. Discrete Math.*, 5(3):422–427, 1992. doi:10.1137/0405033.

13 Atsushi Koike, Shin-Ichi Nakano, Takao Nishizeki, Takeshi Tokuyama, and Shuhei Watanabe. Labeling points with rectangles of various shapes. *Int. J. Comput. Geometry Appl.*, 12(6):511–528, 2002. doi:10.1142/S0218195902001018.

14 Giri Narasimhan and Michiel H. M. Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.

15 William T. Tutte, editor. *Recent Progress in Combinatorics*, New York, 1969. Academic Press. Proceedings of the 3rd Waterloo Conference on Combinatorics, May, 1968.

16 Marc J. van Kreveld, Tycho Strijk, and Alexander Wolff. Point labeling with sliding labels. *Comput. Geom.*, 13(1):21–47, 1999. doi:10.1016/S0925-7721(99)00005-X.

17 Hakan Yildiz and Subhash Suri. Computing Klee's measure of grounded boxes. *Algorithmica*, 71(2):307–329, 2015. doi:10.1007/s00453-013-9797-9.

Deterministically Counting Satisfying Assignments for Constant-Depth Circuits with Parity Gates, with Implications for Lower Bounds

Ninad Rajgopal[1]
Department of Computer Science, University of Oxford, Oxford, United Kingdom
ninad.rajgopal@cs.ox.ac.uk

Rahul Santhanam[2]
Department of Computer Science, University of Oxford, Oxford, United Kingdom
rahul.santhanam@cs.ox.ac.uk

Srikanth Srinivasan
Department of Mathematics, IIT Bombay, Mumbai, India
srikanth@math.iitb.ac.in

Abstract

We give a deterministic algorithm for counting the number of satisfying assignments of any $\mathrm{AC}^0[\oplus]$ circuit C of size s and depth d over n variables in time $2^{n-f(n,s,d)}$, where $f(n,s,d) = n/O(\log(s))^{d-1}$, whenever $s = 2^{o(n^{1/d})}$. As a consequence, we get that for each d, there is a language in E^{NP} that does not have $\mathrm{AC}^0[\oplus]$ circuits of size $2^{o(n^{1/(d+1)})}$. This is the first lower bound in E^{NP} against $\mathrm{AC}^0[\oplus]$ circuits that beats the lower bound of $2^{\Omega(n^{1/2(d-1)})}$ due to Razborov and Smolensky for large d. Both our algorithm and our lower bounds extend to $\mathrm{AC}^0[p]$ circuits for any prime p.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases circuit satisfiability, circuit lower bounds, polynomial method, derandomization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.78

1 Introduction

In circuit complexity, we are interested in understanding the power and weaknesses of various circuit models. This understanding can take various forms for any given circuit class $\mathfrak{C}$. One indication of a deeper understanding is to be able to show *lower bounds* against $\mathfrak{C}$, i.e., prove that some "explicit" function cannot be computed by small circuits in $\mathfrak{C}$. Other indications come from efficient or at least non-trivial solutions for various *meta-algorithmic* tasks involving $\mathfrak{C}$, such as satisfiability algorithms for $\mathfrak{C}$, learning algorithms for $\mathfrak{C}$ or pseudo-random generators useful against $\mathfrak{C}$. There are some formal connections between lower bounds and efficient solvability of meta-algorithmic tasks for classes $\mathfrak{C}$ satisfying some natural closure properties, for example, an equivalence between pseudo-random generators against $\mathfrak{C}$ and average-case lower bounds in linear exponential time against $\mathfrak{C}$ [18], an implication from

[1] This work was supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2014)/ERC Grant Agreement No. 615075.
[2] This work was supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2014)/ERC Grant Agreement No. 615075.

non-trivial satisfiability algorithms for $\mathfrak{C}$ to lower bounds in non-deterministic exponential time against $\mathfrak{C}$ [26], and an implication from non-trivial learning algorithms for $\mathfrak{C}$ to lower bounds in probabilistic exponential time against $\mathfrak{C}$ [20].

For the class of Boolean circuits in general, our understanding is very limited in these terms. We have no super-linear lower bounds against general Boolean circuits for any explicit function, nor do we have non-trivial solutions for any of the meta-algorithmic tasks mentioned. The situation is far better for more restricted classes of circuits, especially circuits of *constant depth* where the gates have unbounded fan-in, both with respect to lower bounds and to meta-algorithmic questions.

In this paper, we focus on constant-depth circuits with AND, OR and PARITY gates (or more generally, AND, OR and MODp gates for some prime p). This is a class that has been intensively studied. Razborov [21] and Smolensky [23] showed super-polynomial size lower bounds against this class for very simple functions such as the MAJORITY function and the MODq function where q is a prime different from 2. As we discuss in Section 2, non-trivial randomized algorithms for deciding satisfiability of circuits from this class are implicit in [27, 16]. Recently, [5] gave quasi-polynomial time algorithms for learning $\mathrm{AC}^0[\oplus]$ circuits in the membership query model.

However, there are still several gaps in our understanding of these $\mathrm{AC}^0[\oplus]$ circuits. With regard to lower bounds, we still do not have *tight* lower bounds for functions like Majority or MODq. The Razborov-Smolensky approximation method yields size lower bounds of the form $2^{\Omega(n^{1/2(d-1)})}$ for Majority against $\mathrm{AC}^0[\oplus]$ circuits of depth d over n variables. The best known upper bound is $2^{\tilde{O}(n^{1/(d-1)})}$ using a standard divide-and-conquer strategy. Closing this quadratic gap in the exponent between upper and lower bounds has been a long-standing open question in the complexity theory of constant-depth circuits. Note that, in contrast, tight bounds are known up to constant factors in the exponent when only AND and OR gates are allowed - Parity is known to have complexity $2^{\Theta(n^{1/(d-1)})}$ in this simpler model [1, 9, 29, 11].

With regard to meta-algorithmic tasks, despite the learning breakthrough of [5] mentioned previously, the situation for pseudo-random generators (PRGs) and satisfiability algorithms is still unclear. While we have super-polynomial worst-case lower bounds against $\mathrm{AC}^0[\oplus]$ circuits, we do not have good average-case lower bounds, and as a consequence, do not have good PRGs. In the case of satisfiability algorithms, randomized algorithms improving non-trivially over brute force search are implicit in previous work, but good deterministic algorithms were unknown prior to this work.

Deterministic satisfiability algorithms are important for a couple of reasons. First, they indicate an improved structural understanding of the circuit class in question, often requiring new techniques to design. Second, they imply circuit lower bounds via the connection of Williams [26] - such an implication is not known from randomized algorithms.[3]

Note that even under standard derandomization assumptions, it is unclear how to get a non-trivial deterministic satisfiability algorithm from a non-trivial randomized satisfiability algorithm. The reason is that derandomization inherently incurs a quadratic slowdown. The deterministic simulation of a randomized algorithm running in time T will take time at least T^2 when a PRG is used to do the derandomization, as the range of the PRG will have size at least T. This quadratic slowdown is unaffordable in the parametric regime

[3] One-sided error randomized algorithms, and more generally, co-non-deterministic algorithms for satisfiability for the circuit class also imply lower bounds via the result of Williams [26]. However, it is easy to check that the previous randomized algorithms for $\mathrm{AC}^0[\oplus]$ were two-sided error algorithms.

under consideration here - we are interested in algorithms running in time $2^{n-g(n)}$, for some $g(n) = o(n)$. Hence the determinization procedure cannot be black-box, but must rather use refined structural information about the circuit class in question.

The main result of this paper is a deterministic algorithm for counting satisfying assignments to $\mathrm{AC}^0[\oplus]$ circuits that improves non-trivially over brute force search.

▶ **Theorem 1** (Main theorem). *The following holds for some absolute constant $\varepsilon_0 > 0$. There is a deterministic algorithm that given an $\mathrm{AC}^0[\oplus]$ circuit C over n variables such that C has depth at most d and size $s \leq 2^{(\varepsilon_0 n)^{1/d}}$, counts the number of satisfying assignments to C in time 2^{n-t} where $t = t(n, s, d) = n/O(\log s)^{d-1}$.*

▶ Remark. It is easy to generalize our results to work with $\mathrm{AC}^0[\mathrm{Mod}_p]$ circuits for any fixed prime p.

In terms of parameters, the savings over brute force search matches the savings in the randomized algorithm of [14] for AC^0 circuits when the circuit size is $n^{1+\Omega(1)}$. Thus any further improvement in savings in our result would give a corresponding improvement for AC^0 algorithms, and moreover via the connection of Williams [26] to circuit lower bounds, a strong improvement would give super-polynomial formula size lower bounds for non-deterministic exponential time.

A minor caveat is that we require the circuit size to be $2^{O(n^{1/d})}$ in Theorem 1, for technical reasons. One would expect that the analysis can be extended to circuit size up to $2^{n^{1/(d-1)}}$.

We use Theorem 1 to get better lower bounds against $\mathrm{AC}^0[\oplus]$ circuits than known before by using the connection of Williams [26]. In fact, we need a refinement of the connection due to [4]. Our lower bound holds for a language in E^{NP}. An intriguing open question is to use the more refined structural information about $\mathrm{AC}^0[\oplus]$ circuits exploited in the proof of Theorem 1 to prove a similar lower bound for more explicit problems or even for MAJORITY.

▶ **Theorem 2.** *For any positive integer d, there is a language in E^{NP} which does not have $\mathrm{AC}^0[\oplus]$ circuits of depth d and size $2^{o(n^{1/(d+1)})}$.*

2 Proof Outline for the Main Theorem

Our starting point is a *randomized* algorithm for the problem of checking satisfiability of an $\mathrm{AC}^0[\oplus]$ circuit C that runs in time 2^{n-m} where $m = n/O(\log s)^{d-1}$, and s, d represent the size, depth of C respectively (we also assume that s is suitably upper bounded, but we ignore it in this section). This algorithm is essentially due to Williams [27] and Lokshtanov, Paturi, Tamaki, Williams and Yu [16], though it does not appear explicitly in either of these papers.

The idea is to use a result of Razborov [21] that essentially says that small $\mathrm{AC}^0[\oplus]$ circuits C can be "approximated" by polynomials of small degree. More formally, there is a randomized algorithm that, when given a circuit C of size s over n variables, produces a (random) polynomial $\boldsymbol{P} \in \mathbb{F}_2[x_1, \ldots, x_n]$ of degree $O(\log s)^{d-1}$ that agrees with the value of a circuit C on any given input with good probability (say 0.9). Along with a fast polynomial evaluation algorithm [25], this immediately yields an *enumeration* algorithm for C (i.e. an algorithm to output the truth table of C) that runs in time $\mathrm{poly}(n)2^n + \mathrm{poly}(s)$, which beats (for large enough s) the trivial algorithm that simply evaluates C on each input and hence takes time $s \cdot 2^n$. Repeating the algorithm $\mathrm{poly}(n)$ times and taking the majority vote on each input, we get an enumeration algorithm that works with high probability.

To obtain a randomized satisfiability algorithm that runs in better-than-brute-force time, we use the above idea along with the "blowup-trick" [28, 6, 16]. For any $a \in \{0,1\}^m$, let C_a be the circuit obtained by setting the last m variables of C to a. Note that the satisfiability

of C can be computed by checking the satisfiability of $C' = \bigvee_{a\in\{0,1\}^m} C_a$, and C' is a circuit of larger size ($s \cdot 2^m$) but fewer variables ($n - m$). We now run the enumeration algorithm above on C' to check if it is satisfiable. Since the circuit is larger, the polynomial produced has larger degree: a careful analysis reveals the degree to be $m \cdot O(\log s)^{d-1}$. Setting $m = n/\Theta(\log s)^{d-1}$, we obtain a polynomial of degree $\ll n$, which can be computed in better-than-brute-force time. Running the enumeration algorithm as above gives the required satisfiability algorithm for C, which now runs in time $2^{n-n/\Theta(\log s)^{d-1}}$.

The above algorithm can further be modified to *count* satisfying assignments, by instead defining $C' = \sum_a C_a$ (a sum over $\mathbb{Z}$) instead of using an OR. Now, an additional idea is required since the polynomials $\boldsymbol{P}_a$ approximating each C_a are $\mathbb{F}_2$-polynomials whereas the sum is over the integers (in the satisfiability case above, the OR gate can further be approximated by a constant-degree polynomial using an idea of Razborov [21], but this idea is not available here). What comes to our rescue is an idea of Toda [24] and its subsequent quantitative refinement due to Beigel and Tarui [3] which tells us that we can simulate a sum (over $\mathbb{Z}$) of K many $\mathbb{F}_2$-polynomials of degree at most D as a polynomial (over $\mathbb{Z}$) of degree at most $D \log K$. Using this idea, we are able to obtain a polynomial of degree $m^2 \cdot O(\log s)^{d-1}$. Overall, this yields an algorithm with slightly worse running time $2^{n-\sqrt{n/\Theta(\log s)^{d-1}}}$.

A partial derandomization of these algorithms was obtained by Chan and Williams [6] in the case that the $\mathrm{AC}^0[\oplus]$ circuits are k-CNFs and generalized by Lokshtanov et al. [16] to the case of ANDs of degree-k polynomials.[4] Chan and Williams [6] observed that Razborov's random construction of polynomials could be suitably derandomized using *ε-biased spaces* [17]. Using this idea (and more work), it was shown that the number of satisfying assignments to a set of degree k-polynomials in n variables could be computed in time $2^{n-n/\Theta(k)}$, which meets the running time of the satisfiability algorithm mentioned above in this special case.

However, it is unclear how to extend the ideas to the setting of general $\mathrm{AC}^0[\oplus]$ circuits since these results used a very special property of the randomized polynomial construction for a single OR gate (and, dually, a single AND gate): namely, that there is a *constant-degree* polynomial whose bias perfectly predicts whether the input to an OR-gate is a 1 input or a 0 input.[5] Unfortunately, such a strong property is not known for general $\mathrm{AC}^0[\oplus]$ circuits: the best we can hope for is to construct a polynomial that with high probability, say $1-\varepsilon$, equals the output of the circuit on any given input, but then we have to pay for this precision in terms of the degree of the polynomial constructed. Further, it is not clear how to derandomize this general inductive construction.

We start by derandomizing the higher-depth random polynomial construction. Once again, ε-biased spaces play a crucial role, and we need to further use derandomized sampling using expanders for a near-optimal derandomization. Using this along with the idea of Beigel and Tarui [3] would yield a deterministic algorithm for counting satisfying assignments in time $2^{n-\sqrt{n/\Theta(\log s)^{d-1}}}$.

However, we further improve the running time to $2^{n-n/\Theta(\log s)^{d-1}}$, matching the running time of the randomized algorithm for checking satisfiability. The principal idea here is to observe (by looking inside the Razborov construction) that the polynomials $\boldsymbol{P}_a$ computed for approximating the individual circuits C_a mentioned above have a very special form: each $\boldsymbol{P}_a$

[4] Note that any k-clause is in particular a degree-k polynomial and hence the latter result generalizes the former.

[5] Briefly, the Razborov polynomial for the OR function on input bits $x_1, \ldots, x_s$ is as follows. Choose $a_1, \ldots, a_s \in \mathbb{F}_2$ independently and uniformly at random and compute $\boldsymbol{\ell}(x) = \sum_i a_i x_i$. Now, note that if $\mathrm{OR}(x_1, \ldots, x_s) = 1$, then $\ell(x)$ computes a uniformly random element of $\mathbb{F}_2$ and otherwise, $\ell(x) = 0$ with probability 1.

is a majority of $\ell = O(m)$ many polynomials $\boldsymbol{P}_{a,1}, \ldots, \boldsymbol{P}_{a,\ell}$ of degree $O(\log s)^{d-1}$ each. We use this and some basic Fourier analysis of Boolean functions to write $\boldsymbol{P}_a$ as a real-valued sum of polynomials of degree at most $O(\log s)^{d-1}$; this idea is inspired by a recent result of Chen and Papakonstantinou [7], who use it to give an improved depth-reduction result for constant-depth circuits with Mod_q gates for composite q. The advantage of doing this is that the degree blow-up in the randomized #SAT algorithm outline above is restricted to applying the idea of Beigel and Tarui, which means that the degree drops to $m \cdot O(\log s)^{d-1}$. Setting m suitably, we now obtain a deterministic algorithm running in time $2^{n-n/O(\log s)^{d-1}}$.

3 Preliminaries

We will consider polynomials over the fields $\mathbb{R}$ and $\mathbb{F}_2$. We identify $\mathbb{F}_2$ with $\{0,1\}$ in the natural way. We use $\binom{n}{\leq k}$ to denote $\sum_{i=0}^{k} \binom{n}{i}$. All algorithms will be implemented in the standard Turing machine with RAM model.

We recall that any function $f : \{0,1\}^n \to R$ for any commutative ring R has a unique representation as a multilinear polynomial. Given two such polynomials representing possibly different functions f_1 and f_2, we can compute the multilinear polynomial corresponding to their product by multiplying the polynomials f_1 and f_2 and "multilinearizing" by replacing each copy of x_i^2 by x_i. In particular, this idea yields the following easy algorithm.

▶ **Fact 3.** *Let R be either $\mathbb{F}_2$ or $\mathbb{Z}$. There is a deterministic algorithm which, when given as input multilinear polynomials $f_1, \ldots, f_t \in R[x_1, \ldots, x_n]$ (as a sum of monomials) of degree $d_1, \ldots, d_t$ such that $\sum_i d_i \leq D$, computes the multilinear polynomial corresponding to the product $f = f_1 \cdots f_t$. The algorithm runs in time $\mathrm{poly}(\binom{n}{\leq D})$ when $R = \mathbb{F}_2$ and time $\mathrm{poly}(\binom{n}{\leq D}, B)$ where B is the bit-complexity of the coefficients of $f_1, \ldots, f_t$ when $R = \mathbb{Z}$.*

3.1 Polynomials over $\mathbb{F}_2$ and Probabilistic polynomials

▶ **Definition 4** (Probabilistic Polynomials)**.** We recall [21, 23] that a *Probabilistic polynomial* from $\mathbb{F}_2[x_1, \ldots, x_n]$ is a random multilinear polynomial $\mathbf{P}$ (chosen according to some distribution) from $\mathbb{F}_2[x_1, \ldots, x_n]$. We say that $\mathbf{P}$ has degree at most D if the distribution of $\mathbf{P}$ is supported on polynomials of degree at most D (or equivalently $\Pr_{\mathbf{P}}[\deg(\mathbf{P}) \leq D] = 1$).

We say that $\mathbf{P}$ is an ε-error probabilistic polynomial for a function $f : \{0,1\}^n \to \{0,1\}$ if for each $a \in \{0,1\}^n$, we have $\Pr_{\mathbf{P}}[\mathbf{P}(a) \neq f(a)] \leq \varepsilon$.

3.2 Polynomials over $\mathbb{R}$ and Modulus-amplification

We recall the following basic facts about writing Boolean functions as multilinear polynomials over the the *reals*. See, e.g. O'Donnell [19] for proofs.

▶ **Fact 5.** *Let $f : \{0,1\}^\ell \to \{0,1\}$ be any Boolean function.*

1. *f can written as a unique* real-valued *linear combination*

$$f(x) = \sum_{S \subseteq [\ell]} \alpha_S \chi_S(x)$$

where $\chi_S(x) = \bigoplus_{i \in S} x_i$ (note that we interpret $\chi_S(x) \in \{0,1\}$ as a real number and the sum above is taken over $\mathbb{R}$).
2. *For each S, $\alpha_S \in [-1,1]$ and is moreover an integral multiple of $2^{-(\ell+1)}$.*
3. *There is a deterministic algorithm $\mathcal{C}$ which, given as input f (via its truth table), computes all the above α_S's in time $2^{O(\ell)}$.*

We also define the Fourier l_1 norm of f as $\|f\|_1 = \sum_S |\alpha_S|$.

The following is a useful *Modulus-amplification* lemma due to Beigel and Tarui [3]. This particular version is from the work of Chan and Williams [6].

▶ **Lemma 6** (Beigel-Tarui [3]). *For every positive integer t, the degree $2t-1$ polynomial $F_t(y) \in \mathbb{Z}[y]$ defined by*

$$F_t(y) = 1 - (1-y)^t \sum_{j=0}^{t-1} \binom{t+j-1}{j} y^j$$

has the property that for all $b \in \mathbb{Z}$,

- *if $b \equiv 0 \pmod 2$, then $F_t(b) \equiv 0 \pmod{2^t}$, and*
- *if $b \equiv 1 \pmod 2$, then $F_t(b) \equiv 1 \pmod{2^t}$.*

Many satisfiability algorithms for circuits are based on evaluating multivariate polynomials efficiently over grids. The following lemma can be found in, e.g., [25].

▶ **Lemma 7** (Fast Polynomial Evaluation). *There is a deterministic algorithm* FPE, *which given as input a multilinear polynomial $P \in \mathbb{Z}[x_1, \ldots, x_n]$ as a sum of monomials, computes the values $(P(a))_{a \in \{0,1\}^n}$ in time $\mathrm{poly}(n, B) \cdot 2^n$ where B is an upper bound on the bit complexity of the coefficients of P.*

3.3 Small-biased sets

We need the notion of *ε-biased sets* [17, 2], which are a standard tool in the derandomization literature.

▶ **Definition 8** (ε-biased sets [17]). For $\varepsilon \in \left(0, \frac{1}{2}\right)$, a set $S \subseteq \{0,1\}^n$ of n-dimensional vectors is ε-biased if for all non-zero $v \in \{0,1\}^n$,

$$\Pr_{w \in S}\{\langle v, w\rangle = 0 \pmod 2\} \in \left(\frac{1}{2} - \varepsilon, \frac{1}{2} + \varepsilon\right)$$

There are many explicit constructions for ε-biased sets. We use the following construction:

▶ **Theorem 9** ([2]). *There is a deterministic algorithm that, given as input n and $\varepsilon \in (0, 1/2)$, produces an ε-biased set $S \subseteq \{0,1\}^n$ of size $O(n^2/\varepsilon^2)$. The algorithm runs in time* $\mathrm{poly}(n/\varepsilon)$.

For any subspace W of $\{0,1\}^n$, define the indicator function $\mathbf{1}_W : \{0,1\}^n \to \{0,1\}$ as $\mathbf{1}_W(z) = 1$ if and only if the vector $z \in W$. From an Observation in O'Donnell's book [19], we see that

▶ **Observation 10** (Proposition 3.11 of [19]). *Let W be a subspace of $\{0,1\}^n$ and $W_\perp$ be its orthogonal complement such that $dim(W_\perp) = k$. Then, the constant term in the Fourier expansion of the indicator function $\mathbf{1}_W$ is $\frac{1}{2^k}$. Moreover, $\|\mathbf{1}_W\|_1 = 1$.*

Essentially, the proof for Observation 10 is based on the fact that a vector $z \in W$ if and only if the dot product of z with every basis vector of $W_\perp$ is 0.

De, Etesami, Trevisan and Tulsiani [8] observed that ε-biased spaces also fool functions with small Fourier l_1-norm.

▶ **Lemma 11** (Lemma 2.5 of [8]). *Let S be an ε-biased set. For every function $f : \{0,1\}^n \to \mathbb{R}$ we have,*

$$\left| \mathop{\boldsymbol{E}}_{y \in S}[f(y)] - \mathop{\boldsymbol{E}}_{x \sim U_n}[f(x)] \right| \leq \varepsilon \|f\|_1$$

where y is picked uniformly at random from S and U_n is the uniform distribution over $\{0,1\}^n$.

From Observation 10 and Lemma 11, we state the following useful corollary.

▶ **Corollary 12.** *Let W be a subspace of $\{0,1\}^n$, such that the co-dimension of W is k and let S be an ε-biased set. Then*

$$\Pr_{x \in S}\{x \in W\} \in \left(\frac{1}{2^k} - \varepsilon, \frac{1}{2^k} + \varepsilon\right)$$

where x is picked uniformly at random from S.

Intuitively speaking, Corollary 12 states that an ε-biased set also "fools" conjunctions of parities.

3.4 Expanders

Proofs of the following well-known facts may be found in the monograph of Hoory, Linial and Wigderson [13].

Given an undirected Δ-regular multigraph G, we denote by $A(G)$ its adjacency matrix and $\tilde{A}(G) = (1/\Delta)A(G)$ its *normalized* adjacency matrix. Then, we have the following.

- The all-1s vector $v \in \mathbb{R}^n$ is an eigenvector of $\tilde{A}(G)$ with eigenvalue 1.
- G is connected if and only if v is the only such eigenvector (up to scalar multiplication).

▶ **Definition 13** (Expanders [13])**.** An undirected multigraph G is an (N, Δ, λ) *expander* if it is a Δ-regular connected graph on N vertices (which we will identify with $[N]$), and all the eigenvalues (counted with multiplicity) of $\tilde{A}(G)$ other than 1 are bounded by λ in absolute value.

Let $(G_n)_{n\geq 1}$ be a sequence of expander graphs with G_n being an $(f(n), \Delta, \lambda)$-expander for some increasing function $f : \mathbb{N} \to \mathbb{N}$ and constants Δ and λ. We say that $(G_n)_{n\geq 1}$ is *explicit* if there is a deterministic algorithm that, when given as input n, produces the graph G_n in time $\mathrm{poly}(n, f(n))$.

We use Reingold, Vadhan and Wigderson's [22] explicit construction for an expander graph.

▶ **Theorem 14** ([22])**.** *For every fixed $\lambda > 0$ there exists a constant $\Delta > 0$ and an explicit sequence of expander graphs $(G_n)_{n\geq 1}$, where G_n is a $(2^n, \Delta, \lambda)$ expander graph, for some large enough constant Δ. Further, we can assume that Δ is a power of 2.*

We will need the following expander-based Chernoff bound due to Gillman [10]. The version below is due to Healy [12].

▶ **Theorem 15** ([10, 12])**.** *Let G be an (N, D, λ)-graph and let $S \subseteq [N]$ be a subset of the vertices of G such that $|S| = \beta N$. Consider the natural ℓ-step random walk on G defined by choosing a uniformly random vertex $u_1 \in [N]$ and repeatedly choosing random neighbours $\ell - 1$ times to obtain a (random) sequence $(u_1, \ldots, u_\ell)$ of vertices of G. Let X_S denote the number of $i \in [\ell]$ such that $u_i \in S$. For any fixed $\rho \in (0,1)$, we have*

$$\Pr_{u_1,\ldots,u_\ell}[|X_S - \beta\ell| \geq \rho\ell] \leq 2\exp\left(-\frac{1}{4}\rho^2(1-\lambda)\ell\right).$$

4 The #SAT algorithm

In this section we prove Theorem 1. We start with a deterministic algorithmic version of a lemma of Razborov [21] regarding approximating $\mathrm{AC}^0[\oplus]$ circuits by low-degree polynomials. Using this version, we then state formally the #SAT algorithm and analyze it.

4.1 Derandomized construction of probabilistic polynomials for $\mathbf{AC^0}[\oplus]$

The following lemma is an algorithmic version of a result of Razborov [21] (see also Kopparty-Srinivasan [15] for the dependence on ε). It can be viewed as a derandomization of a randomized algorithm due to Williams [27].

▶ **Lemma 16.** *For any $\varepsilon > 0$, an $\mathrm{AC}^0[\oplus]$ circuit C over n variables of depth d and size at most s has an ε-error probabilistic polynomial $\mathbf{P}$ from $\mathbb{F}_2[x_1, \ldots, x_n]$ of degree at most $D = (O(\log s)^{d-1} \cdot \log(1/\varepsilon))$. Moreover $\mathbf{P} = \mathrm{Maj}(\mathbf{P}_1, \ldots, \mathbf{P}_\ell)$, where $\mathbf{P}_1, \ldots, \mathbf{P}_\ell$ are probabilistic polynomials of degree $D_1 = O(\log s)^{d-1}$ and $\ell = O(\log(1/\varepsilon))$.*

Moreover, there is a deterministic procedure $\mathcal{S}$, which when given as input the circuit C, the parameter ε, and a uniformly random Boolean string $\boldsymbol{\sigma}$ of length $r = O(\log(s/\varepsilon))$, produces a random sample of the polynomials $\mathbf{P}_1, \ldots, \mathbf{P}_\ell$ as sums of monomials. The procedure $\mathcal{S}$ runs in time $\mathrm{poly}\left(\ell, s, \binom{n}{\leq D_1}\right)$.

Before we go into the proof of Lemma 16, we prove a weaker version that will be useful in the proof of Lemma 16. This result is a higher-depth analogue of a result of Chan and Williams [6] which itself may be viewed as a derandomization of the construction of Razborov [21] for depth-1 circuits.

▶ **Lemma 17.** *For every $\mathrm{AC}^0[\oplus]$ circuit C over n variables of depth d and size s, there exists a probabilistic polynomial $\mathbf{P}'$ with error at most $\frac{1}{4}$ with degree at most $D_1 = O(\log s)^{d-1}$. Further, there is a deterministic algorithm $\mathcal{S}_1$ that produces a random sample of $\mathbf{P}'$ as a sum of monomials given as input s, C and $O(\log s)$ random bits. The algorithm $\mathcal{S}_1$ runs in time* $\mathrm{poly}\left(s, \binom{n}{\leq D_1}\right)$.

Proof. Let C be the circuit input to the sampling algorithm. Let $m = s\log(40s), \varepsilon = 1/(20s)$ and $S \subseteq \{0,1\}^m$ be an ε-biased set of size $\mathrm{poly}(s)$ given by Theorem 9. Fix an input $a \in \{0,1\}^n$.

Fix some enumeration $g_1, \ldots, g_s$ of all the gates of C. Let $h \in \{g_1, \ldots, g_s\}$ be the output gate of C and let $C_1, \ldots, C_r$ ($r \leq s$) be the depth-$(d-1)$ sub-circuits of C feeding into h. First we construct a probabilistic polynomial of degree $O(\log s)$ for each gate $g \neq h$ in the circuit, following which we construct a constant degree probabilistic polynomial for the gate h. Composing these polynomials together gives the probabilistic polynomial for the circuit C of degree $O(\log s)^{d-1}$.

Fix any gate $g \neq h$ in C. If g is a NOT gate or a $\oplus$ gate, we can easily get a polynomial P_g of degree 1 which always agrees with the function computed by g and needs no random bits. Therefore, let g be an OR gate in C (a dual construction works for AND gates). Let $g_{i_1}, \ldots, g_{i_k}$ be the gates that are inputs to g and $v_g \in \{0,1\}^s$ such that $v_g[i] = 1$ iff $i \in \{i_1, \ldots, i_k\}$ and $g_i = 1$ (on the fixed input a). Observe that g outputs 1 iff v_g is not the zero vector.

Let $t = \log(40s)$. Construct the vectors $u_1, \ldots, u_t \in \{0,1\}^m$, such that for each u_p, $1 \leq p \leq t$, we divide u_p into t blocks, such that the p^{th} block contains v_g and all the other bits of u_p are set to 0. The dimension of the vector space $V \subseteq \mathbb{F}_2^m$ spanned by $\{u_1, \ldots, u_t\}$ is equal to t if v_g is a non-zero vector. Let $\boldsymbol{y} \in S$ be picked uniformly at random from S. We split $\boldsymbol{y}$ into t blocks $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_t$ of size s each. W.l.o.g. consider the vector u_1. The inner product $\langle u_1, \boldsymbol{y}\rangle$ which is exactly equal to the inner product $\langle v_g, \boldsymbol{y}_1\rangle$, can be calculated by hardwiring into a MOD_2 gate all the input gates of g for which the corresponding bit in $\boldsymbol{y}$ equals 1. In other words, the inner product $\langle u_1, \boldsymbol{y}\rangle$ is represented by the polynomial $q_1 = \sum_{j \in \{i_1, \ldots, i_k\}} g_j \cdot (\boldsymbol{y}_1)_j$ in the inputs $g_{i_1}, \ldots, g_{i_k}$. Repeating this construction for all t

vectors $u_1, \ldots, u_t$, we get polynomials $q_1^{\boldsymbol{y}}, \ldots, q_t^{\boldsymbol{y}}$. Finally, we compute the disjunction of these t terms using the polynomial $P_g^{\boldsymbol{y}} = 1 - \prod_{p=1}^{t}(1 - q_p^{\boldsymbol{y}})$, which is of degree $t = O(\log s)$ in the input variables of g.

We now analyze the behaviour of $P_g^{\boldsymbol{y}}$ on a fixed input to the gate g. If the gate g outputs 0, then for any $y \in S$, we get $\langle u_p, y\rangle = 0$ for every $1 \le p \le t$. In particular, $P_g^{\boldsymbol{y}}$ outputs 0 with probability 1. On the other hand, if g outputs 1, we see that $P_g^{\boldsymbol{y}}$ errs on this input iff for each $1 \le p \le t$, $q_p^{\boldsymbol{y}} = 0$. Let $V_\perp$ be the orthogonal complement to $V = \mathrm{span}(\{u_1, \ldots, u_t\})$. Note that the codimension of the vector space $V_\perp$ is t. We now use Corollary 12 to see that, if g outputs 1, a uniformly random element $\boldsymbol{y}$ from S belongs to $V_\perp$, or in other words, yields $\langle u_p, \boldsymbol{y}\rangle = 0$ for every $1 \le p \le t$, with probability at most $\frac{1}{2^t} + \varepsilon$. Thus, $P_g^{\boldsymbol{y}}$ disagrees with the output of g on *any fixed input* with probability at most $\frac{1}{2^t} + \varepsilon = \frac{1}{40s} + \frac{1}{20s} = \frac{3}{40s}$.

We pick a *single* $\boldsymbol{y}$ uniformly at random from S and then use it (for each OR and AND gate) to get the probabilistic polynomial $P_g^{\boldsymbol{y}}$ for each gate $g \neq h$ in C. For each $j \in [r]$, composing the polynomials representing the gates in C_j, we obtain a probabilistic polynomial $\mathbf{Q}_j$ of degree at most $O(\log s)^{d-1}$. Following this, we use the first $t_1 = 3$ blocks of $\boldsymbol{y}$ in a similar fashion to get a probabilistic polynomial $\mathbf{P}_h$ of degree $O(1)$ for the function computed by the output gate h with error at most $\frac{1}{2^{t_1}} + \varepsilon$. Now, for any input $a \in \{0,1\}^n$, $\mathbf{P}' = \mathbf{P}_h(\mathbf{Q}_1, \ldots, \mathbf{Q}_r)$ satisfies

$$\begin{aligned}
\Pr_{\mathbf{P}'}[C(a) \neq \mathbf{P}'(a)] &\le \Pr_{\mathbf{Q}_1,\ldots,\mathbf{Q}_r}[\exists j \in [r] : C_j(a) \neq \mathbf{Q}_j(a)] \\
&\quad + \Pr_{\mathbf{P_h}}[h(C_1(a), \ldots, C_r(a)) \neq \mathbf{P}_h(C_1(a), \ldots, C_r(a))] \\
&\le s \cdot \frac{3}{40s} + \frac{1}{2^{t_1}} + \varepsilon \\
&\le \frac{3}{40} + \frac{1}{8} + \frac{1}{20s} \le \frac{1}{4}
\end{aligned}$$

where the second inequality uses a union bound over the (at most s) gates g in C.

This gives us a probabilistic polynomial $\mathbf{P}'$ of degree $O(\log s)^{d-1}$ for the circuit C, with error at most $\frac{1}{4}$ and we need $O(\log |S|) = O(\log s)$ random bits to get this sample. This polynomial is multilinear and by expanding the monomials at each step we can ensure multilinearity of the intermediate polynomials. Using Fact 3 we see that the running time of the sampling algorithm is given by poly $\left(s, \binom{n}{\le D_1}\right)$. ◀

Proof of Lemma 16. Let $k = O(\log s)$ be the number of random bits needed by the algorithm $\mathcal{S}_1$ from Lemma 17. Consider an explicit $(2^k, \Delta, \lambda)$ expander graph G on $V = \{0,1\}^k$ given by Theorem 14, where Δ is a large enough constant given by the construction and $\lambda = \frac{1}{2}$. The graph G can be constructed in time poly$(2^k) =$ poly(s).

Let $\ell = 200 \log\left(\frac{2}{\varepsilon}\right)$. We define the algorithm $\mathcal{S}$ to be the following deterministic procedure which takes as input the circuit C, the parameter ε and a random string $\boldsymbol{\sigma}$ of length $r = k + (\ell - 1) \cdot \log \Delta$ and produces a random sample of the probabilistic polynomials $\mathbf{P}_1, \ldots, \mathbf{P}_\ell$, where each of the $\mathbf{P}_i$ is an instantation of the probabilistic polynomial constructed in Lemma 17.

1. Perform a length ℓ random walk in G using the bits of $\boldsymbol{\sigma}$ to obtain $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_\ell \in V = \{0,1\}^k$. I.e. first choose $\boldsymbol{u}_1$ uniformly at random from V and for each $i \in \{2, \ldots, \ell\}$, let $\boldsymbol{u}_i$ be a random neighbour of $\boldsymbol{u}_{i-1}$ in the graph G.
2. For each $1 \le i \le \ell$, use $\boldsymbol{u}_i$ as the input string of random bits to algorithm $\mathcal{S}_1$ from Lemma 17 to obtain a random sample $\mathbf{P}_i$ of the 1/4-error probabilistic polynomial $\mathbf{P}'$ for C.

The number of random bits used by $\mathcal{S}$ to obtain a random sample of $\mathbf{P}_1, \ldots, \mathbf{P}_\ell$ is $r = O(k+\ell) = O(\log s + \log\left(\frac{1}{\varepsilon}\right)) = O\left(\log(s/\varepsilon)\right)$. At each step of the random walk we spend poly $\left(s, \binom{n}{\leq D_1}\right)$ time to get a sample and thus, the overall running time of $\mathcal{S}$ is poly $\left(\ell, s, \binom{n}{\leq D_1}\right)$.

Now, define the probabilistic polynomial $\mathbf{P} = \mathrm{Maj}(\mathbf{P}_1, \ldots, \mathbf{P}_\ell)$. Since, the majority of ℓ bits is a polynomial of degree at most ℓ, the degree of $\mathbf{P}$ is $O(\log s)^{d-1} \cdot \log(1/\varepsilon)$.

To show that $\mathbf{P}$ is a probabilistic polynomial with error ε for the circuit C, fix an input $a \in \{0,1\}^n$. For any $u \in V$, let P^u be the polynomial sampled by the algorithm $\mathcal{S}_1$ when given u as input. Let B be the set of vertices $u \in V$ such that $P^u(a) \neq C(a)$. Note that as $\mathcal{S}_1$ samples a $1/4$-error probabilistic polynomial for C, we must have $|B| \leq |V|/4$. Now, we have $\mathbf{P}(a) \neq C(a)$ iff a majority of the vertices on the random walk sampled by $\mathcal{S}$ belong to B. Using Theorem 15 we show that this event happens with a probability at most ε.

Let X_B be the random variable which denotes the number of $i \in [\ell]$ such that $i \in B$. Using Theorem 15 with the settings $\beta = \frac{|B|}{|V|} \leq \frac{1}{4}, \lambda = \frac{1}{2}, \rho = \frac{1}{4}$ and $\ell = 200\log(2/\varepsilon)$, we see that

$$\Pr_{u_1,\ldots,u_\ell}\left\{\left|X_B - \ell/4\right| > \ell/4\right\} \leq 2\exp\left(-\frac{1}{4}\rho^2(1-\lambda)\cdot 200\log\left(\frac{2}{\varepsilon}\right)\right) < 2\cdot 2^{-\log\left(\frac{2}{\varepsilon}\right)} < \varepsilon \quad \blacktriangleleft$$

4.2 The algorithm and its analysis

We begin by describing the #SAT algorithm $\mathcal{A}$.

Algorithm $\mathcal{A}$.

The algorithm $\mathcal{A}$ has the following desired input-output behaviour.

Input: An $\mathrm{AC}^0[\oplus]$ circuit C over n variables of size at most s and depth at most d. Recall that $s \leq 2^{(\varepsilon_0 n)^{1/d}}$ for some absolute constant $\varepsilon_0 > 0$ (to be chosen below). We assume $s \geq n$.

Desired Output: The number of satisfying assignments of C.

Notation. Let $m = \gamma n/(\log s)^{d-1}$ for a suitable absolute constant $\gamma > 0$ that will be fixed below. Let $\varepsilon = 1/2^{10m}$. For s and ε as defined above, choose $r = O(\log(s/\varepsilon))$ suitably so that the sampling algorithm $\mathcal{S}$ from Lemma 16 works as stated.

For each $\sigma \in \{0,1\}^r$, let $(P_1^\sigma, \ldots, P_\ell^\sigma)$ be the output of the algorithm $\mathcal{S}$ on string σ (note that the probabilistic polynomials $(\boldsymbol{P}_1, \ldots, \boldsymbol{P}_\ell)$ from Lemma 16 are exactly the polynomials $(P_1^{\boldsymbol{\sigma}}, \ldots, P_\ell^{\boldsymbol{\sigma}})$ for a uniformly random $\boldsymbol{\sigma}$ and hence $\boldsymbol{P} = \mathrm{Maj}(P_1^{\boldsymbol{\sigma}}, \ldots, P_\ell^{\boldsymbol{\sigma}})$).

For fixed $\sigma \in \{0,1\}^r$ and $c \in \{0,1\}^m$, let $P_i^{\sigma,c} \in \mathbb{F}_2[x_1, \ldots, x_{n-m}]$ be defined by $P_i^{\sigma,c} = P_i^\sigma(x_1, \ldots, x_{n-m}, c_1, \ldots, c_m)$ (i.e. the last m variables of P_i^σ are fixed to bits of c). Let $P^{\sigma,c} = \mathrm{Maj}(P_1^{\sigma,c}, \ldots, P_\ell^{\sigma,c})$.

For $S \subseteq [\ell]$, let $P_S^{\sigma,c} = \bigoplus_{i\in S} P_i^{\sigma,c}$. Let $Q_S^{\sigma,c}$ be the polynomial with integer coefficients obtained by treating the $\mathbb{F}_2$-coefficients of $P_S^{\sigma,c}$ as integers. Note that for each $b \in \{0,1\}^{n-m}$, $P_S^{\sigma,c}(b) \equiv Q_S^{\sigma,c}(b) \pmod 2$.

1. Using the algorithm $\mathcal{C}$ from Fact 5, compute[6] integers $k_S \in \{2^{-(\ell+1)}, \ldots, 2^{(\ell+1)}\}$ for each $S \subseteq [\ell]$ such that $\mathrm{Maj}(z_1, \ldots, z_\ell) = \frac{1}{2^{\ell+1}} \sum_{S\subseteq[\ell]} k_S \bigoplus_{i\in S} z_i$.

[6] There is actually an explicit description of the integers k_S (see, e.g., O'Donnell [19]) using which each k_S can each be computed in time poly(ℓ) as opposed to the $2^{O(\ell)}$ time taken by the algorithm $\mathcal{C}$. However, the algorithm we give here doesn't need this and works for any Boolean function in place of Maj.

2. For each $c \in \{0,1\}^m$, $\sigma \in \{0,1\}^r$ and $S \subseteq [\ell]$, construct (as a sum of monomials) the polynomial $Q_S^{\sigma,c}(x_1, \ldots, x_{n-m})$ using the algorithm $\mathcal{S}$ from Lemma 16.
3. Construct as a sum of monomials (Fact 3) the multilinear polynomial $R \in \mathbb{Z}[x_1, \ldots, x_{n-m}]$ defined by

$$R(x_1, \ldots, x_{n-m}) = \sum_{c \in \{0,1\}^m} \sum_{\sigma \in \{0,1\}^r} \sum_{S \subseteq [\ell]} k_S \cdot F_t(Q_S^{\sigma,c}(x_1, \ldots, x_{n-m}))$$

   where F_t is the modulus amplifying polynomial given by Lemma 6 and $t = A \log(s/\varepsilon)$ for a large absolute constant $A > 0$ chosen below.
4. Evaluate $R(b)$ for each $b \in \{0,1\}^{n-m}$ using the algorithm FPE from Lemma 7.
5. Let $R_t(b) = R(b) \pmod{2^t} \in \{0, \ldots, 2^t - 1\}$. Output $\sum_{b \in \{0,1\}^{n-m}} [R_t(b)/2^{\ell+1+r}]$ where $[x]$ denotes the integer closest to x (if x is a half-integer, $[x]$ is defined arbitrarily).

Theorem 1 follows directly from Lemmas 18 and 19 below.

▶ **Lemma 18** (Running time). *For any constant $A > 0$, there exist constants $\gamma > 0$ and $\varepsilon_0 > 0$ such that the algorithm $\mathcal{A}$, on an input circuit C of depth at most d and size at most $s \leq 2^{(\varepsilon_0 n)^{1/d}}$, has running time* $\mathrm{poly}(s) \cdot 2^{n/10} + \mathrm{poly}(n) \cdot 2^{n-m}$.

Proof. We analyse the running time of $\mathcal{A}$ by looking at the running times for each of its individual steps. From Fact 5, we see that Step 1 of the algorithm takes $2^{O(\ell)}$ running time. Since $\ell = O(\log(1/\varepsilon)) = O(m) = O(n/(\log s)^{d-1}) = o(n)$, this step takes at most $2^{o(n)} < 2^{n/10}$ time to run.

For Step 2, we see that the running time is $2^{\ell+m+r} \cdot \mathrm{poly}\left(\ell, s, \binom{n}{\leq D_1}\right)$, as we construct $2^{\ell+m+r}$ many polynomials using the algorithm $\mathcal{S}$, each of which takes $\mathrm{poly}(\ell, s, \binom{n}{\leq D_1})$ time to construct, as seen in Lemma 16. For the parameters we pick, we see that $2^{\ell+m+r} = 2^{o(n)}$ and $\binom{n}{\leq D_1} \leq n^{D_1} = n^{O(\log s)^{d-1}} = 2^{O(n^{(d-1)/d} \log n)}$. Thus, Step 2 takes at most $2^{n/10} \cdot \mathrm{poly}(n, s)$ time.

Step 3 takes time $2^{\ell+m+r} \cdot \mathrm{poly}\left(\ell, \binom{n}{\leq 2tD_1}\right)$, as the modulus amplifying polynomial F_t blows the degree of the polynomial up by a factor of $(2t-1)$ and the number of monomials in the multilinear expansion of the polynomial $R(x_1, \ldots, x_{n-m})$ is $\mathrm{poly}\left(\ell, \binom{n}{\leq 2tD_1}\right)$. To upper bound this running time, let $c' > 0$ be a constant such that the degree parameter D_1 from Lemma 16 is at most $c' \cdot (\log s)^{d-1}$. Then the degree of the polynomial R is at most

$$\begin{aligned} 2tD_1 &\leq 2t \cdot c'(\log s)^{d-1} \\ &= 2A \log(s/\varepsilon) \cdot c'(\log s)^{d-1} \\ &= 2Ac'(\log s)^d + 20Amc'(\log s)^{d-1} \\ &= 2Ac'(\log s)^d + 20A\gamma c' n \end{aligned}$$

where we have used $\log(1/\varepsilon) = 10m$ and $m = \gamma n/(\log s)^{d-1}$.

Fix the constants $\varepsilon_0 = \left(\frac{1}{400Ac'}\right)$ and $\gamma = \frac{1}{4000Ac'}$. This ensures that the $2tD_1 \leq 0.01n$. From this we see that, $\binom{n}{\leq 2tD_1}$ is at most $\left(\frac{ne}{2tD_1}\right)^{2tD_1} \leq \left(\frac{ne}{0.01n}\right)^{0.01n} < 2^{0.09n}$ and we see that step 3 takes at most $2^{n/10}$ time.

Note that each k_S computed in Step 1 is an $(\ell+1)$-bit integer and hence, the bit complexity of the coefficients of R is at most $O(\ell + m + r) \leq n$. From Lemma 7, we see that Step 4 takes $2^{n-m}\,\mathrm{poly}(n)$ time and Step 5 runs in the same time trivially. Thus, the algorithm $\mathcal{A}$ takes a total of $\mathrm{poly}(n, s) \cdot 2^{n/10} + \mathrm{poly}(n) \cdot (2^{n-m})$ to run. ◀

▶ **Lemma 19** (Correctness). *Assume that $A > 0$ is chosen large enough so that $t > m + r + \ell + 10$. Then the algorithm $\mathcal{A}$ above outputs the number of satisfying assignments of C.*

Proof. We need to show that the algorithm $\mathcal{A}$ computes correctly the number of satisfying assignments of C. To this end, define the function C' on $n-m$ input bits by

$$C'(x_1, \ldots, x_{n-m}) = \sum_{c \in \{0,1\}^m} C(x_1, \ldots, x_{n-m}, c_1, \ldots, c_m)$$

where the sum is over $\mathbb{Z}$. It suffices to show that, for every $b \in \{0,1\}^{n-m}$, $[R_t(b)/2^{\ell+1+r}]$ is a correct estimate of $C'(b)$ since this implies that the number of satisfying assignments of C, which is equal to $\sum_{b \in \{0,1\}^{n-m}} C'(b)$, is computed correctly.

From the definition of $R_t(b)$, we see that for any $b \in \{0,1\}^{n-m}$

$$\begin{aligned} R_t(b) &= R(b) \pmod{2^t} \\ &= \left(\sum_{c \in \{0,1\}^m} \sum_{\sigma \in \{0,1\}^r} \sum_{S \subseteq [\ell]} k_S \cdot F_t(Q_S^{\sigma,c}(b)) \right) \pmod{2^t} \\ &= \left(\sum_{c \in \{0,1\}^m} \sum_{\sigma \in \{0,1\}^r} \sum_{S \subseteq [\ell]} k_S \cdot \left(F_t(Q_S^{\sigma,c}(b)) \pmod{2^t} \right) \right) \pmod{2^t} \end{aligned}$$

For every $\sigma \in \{0,1\}^r, c \in \{0,1\}^m, S \subseteq [\ell]$ and $b \in \{0,1\}^{n-m}$, we use the property of the modulus amplifying polynomial F_t from Lemma 6, to observe that $F_t(Q_S^{\sigma,c}(b)) \pmod{2^t} = P_S^{\sigma,c}(b) = \oplus_{i \in S} P_i^{\sigma,c}(b)$. This observation, along with Step 1 of the algorithm $\mathcal{A}$ implies that the sum $\sum_{S \subseteq [\ell]} k_S \cdot (F_t(Q_S^{\sigma,c}(b)) \pmod{2^t})$ is the same as $2^{\ell+1}\mathrm{Maj}(P_1^{\sigma,c}(b), \ldots, P_\ell^{\sigma,c}(b)) = 2^{\ell+1} P^{\sigma,c}(b)$. In other words,

$$R_t(b) = \left(2^{\ell+1} \sum_{c \in \{0,1\}^m} \sum_{\sigma \in \{0,1\}^r} P^{\sigma,c}(b) \right) \pmod{2^t} = 2^{\ell+1} \sum_{c \in \{0,1\}^m} \sum_{\sigma \in \{0,1\}^r} P^{\sigma,c}(b)$$

where the last equality follows from the fact that $t > m + \ell + r + 10$.

Now, for every $b \in \{0,1\}^{n-m}$ and $c \in \{0,1\}^m$, we have

$$\sum_{\sigma \in \{0,1\}^r} P^{\sigma,c}(b) \begin{cases} \geq 2^r(1-\varepsilon) & \text{if } C(b,c) = 1 \\ \leq 2^r \varepsilon & \text{if } C(b,c) = 0 \end{cases}$$

where $\varepsilon = 2^{-10m}$. Since $C'(b) = \sum_{c \in \{0,1\}^m} C(b,c)$, we now have that for every $b \in \{0,1\}^{n-m}$,

$$\begin{aligned} R_t(b) &\geq 2^{\ell+1} \cdot 2^r(1-\varepsilon) C'(b), \text{ and} \\ R_t(b) &\leq 2^{\ell+1} \cdot (2^r(1-\varepsilon)C'(b) + \varepsilon(2^m - C'(b)) \cdot 2^r) \\ &\leq 2^{\ell+1} \cdot (2^r(1-\varepsilon)C'(b) + \varepsilon 2^m \cdot 2^r)\,. \end{aligned}$$

In particular, since $\varepsilon = 2^{-10m}$, for every $b \in \{0,1\}^{n-m}$, we see that the estimate returned by the algorithm, which is $[R_t(b)/2^{\ell+1+r}]$, is equal to $C'(b)$. ◀

4.3 A Consequence for Lower Bounds

Our #SAT algorithm can be used to obtain improved lower bounds against $\mathrm{AC}^0[\oplus]$ circuits (and more generally, against $\mathrm{AC}^0[p]$ circuits for prime p), using Williams' connection between algorithms and lower bounds. These lower bounds are, however, not very explicit - they hold for a language in E^{NP}.

We first remind the reader of the best explicit lower bounds that are known against $\mathrm{AC}^0[\oplus]$ circuits.

▶ **Theorem 20.** *[21, 23] For each positive integer d, there is a language computable in polynomial time that requires depth-d $\mathrm{AC}^0[\oplus]$ circuits of size $2^{\Omega(n^{1/2(d-1)})}$.*

In fact, there is a fixed explicit language in polynomial time, namely Majority, for which the lower bounds of Theorem 20 hold for all d.

In terms of parameters, the bound in Theorem 20 is weaker than the best known bound for AC^0 circuits as a function of d. Parity is known to require depth-d circuits of size $2^{\Omega(n^{1/(d-1)})}$. The exponent in the bound of Theorem 20 is quadratically smaller. It has been a longstanding open problem to improve the bound in Theorem 20 to match the known bounds for constant-depth circuits without prime modulus circuits. Using the algorithmic method of Williams and its refinements, we are able to use our #SAT algorithm to make progress on this problem.

The following lemma can be shown using the proof technique of Theorem 1.5 in [4].

▶ **Lemma 21.** *[26, 4] Let s be a size function and d a positive integer such that satisfiability can be solved deterministically in time $2^n/n^{\omega(1)}$ on $\mathrm{AC}^0[\oplus]$-circuits of size $O(s(n))$ and depth at most d on n variables. Then there is a language in E^{NP} which does not have $\mathrm{AC}^0[\oplus]$ circuits of depth $d-1$ and size $o(s(n))$.*

We now apply the lemma to get better lower bounds than Theorem 20 in terms of size against $\mathrm{AC}^0[\oplus]$ circuits when the depth is at least 3. A similar lower bound against $\mathrm{AC}^0[p]$ circuits can be shown for prime p using the analogue of Theorem 1 for $\mathrm{AC}^0[p]$ circuits. The following result is simply a re-statement of Theorem 2.

▶ **Theorem 22.** *For any positive integer d, there is a language in E^{NP} which does not have $\mathrm{AC}^0[\oplus]$ circuits of depth d and size $2^{o(n^{1/(d+1)})}$.*

Proof. Pick $\varepsilon < \varepsilon_0$, where ε_0 is the constant in Theorem 1. By Theorem 1, for any size $s \leq 2^{(\varepsilon n)^{1/(d+1)}}$, there is a deterministic algorithm solving satisfiability of $\mathrm{AC}^0[\oplus]$ circuits of size at most s and depth at most $d+1$ in time $2^n/n^{\omega(1)}$. Now using Lemma 21, we have that there is a language in E^{NP} which does not have $\mathrm{AC}^0[\oplus]$ circuits of depth d and size $o(s(n))$, which establishes our claim. ◀

References

1. Miklos Ajtai. Σ^1_1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24(1):1–48, 1983.
2. Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple construction of almost k-wise independent random variables. *Random Struct. Algorithms*, 3(3):289–304, 1992.
3. Richard Beigel and Jun Tarui. On ACC. *Computational Complexity*, 4:350–366, 1994.
4. Eli Ben-Sasson and Emanuele Viola. Short PCPs with projection queries. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 163–173, 2014.
5. Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 10:1–10:24, 2016.
6. Timothy M. Chan and Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1246–1255, 2016.

7 Shiteng Chen and Periklis A. Papakonstantinou. Depth-reduction for composites. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 99–108. IEEE Computer Society, 2016. URL: `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7781469`, `doi:10.1109/FOCS.2016.20`.

8 Anindya De, Omid Etesami, Luca Trevisan, and Madhur Tulsiani. Improved pseudorandom generators for depth 2 circuits. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 504–517. Springer, 2010.

9 Merrick Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

10 David Gillman. A Chernoff bound for random walks on expander graphs. *SIAM Journal on Computing*, 27(4):1203–1220, 1998.

11 Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Symposium on Theory of Computing* (STOC), pages 6–20, 1986.

12 Alexander D Healy. Randomness-efficient sampling within NC. *Computational Complexity*, 17(1):3–37, 2008.

13 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc. (N.S.)*, 43(4):439–561, 2006. `doi:10.1090/S0273-0979-06-01126-8`.

14 Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for AC^0. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 961–972, 2012.

15 Swastik Kopparty and Srikanth Srinivasan. Certifying polynomials for $\mathrm{AC}^0[\oplus]$ circuits, with applications. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, pages 36–47, 2012.

16 Daniel Lokshtanov, Ramamohan Paturi, Suguru Tamaki, R. Ryan Williams, and Huacheng Yu. Beating brute force for systems of polynomial equations over finite fields. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2190–2202. SIAM, 2017. `doi:10.1137/1.9781611974782.143`.

17 Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993.

18 Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

19 Ryan O'Donnell. *Analysis of Boolean functions*. Cambridge University Press, 2014.

20 Igor Carboni Oliveira and Rahul Santhanam. Conspiracies between learning algorithms, circuit lower bounds, and pseudorandomness. In *32nd Computational Complexity Conference, CCC 2017, July 6-9, 2017, Riga, Latvia*, pages 18:1–18:49, 2017.

21 Alexander A Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

22 Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. *Annals of Mathematics*, 155(1):157–187, 2002.

23 Roman Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 77–82. ACM, 1987.

24 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. `doi:10.1137/0220053`.

25 Ryan Williams. Guest column: a casual tour around a circuit complexity bound. *SIGACT News*, 42(3):54–76, 2011. doi:10.1145/2034575.2034591.

26 Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3):1218–1244, 2013.

27 Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 194–202, 2014.

28 Ryan Williams. Nonuniform ACC circuit lower bounds. *Journal of the ACM (JACM)*, 61(1):2, 2014.

29 Andrew Chi-Chih Yao. Separating the polynomial-time hierarchy by oracles (preliminary version). In *Symposium on Foundations of Computer Science* (FOCS), pages 1–10, 1985.

Results on the Dimension Spectra of Planar Lines

Donald M. Stull
Inria Nancy-Grand Est, 615 rue du jardin botanique, 54600 Villers-les-Nancy, France
donald.stull@inria.fr

Abstract

In this paper we investigate the (effective) dimension spectra of lines in the Euclidean plane. The dimension spectrum of a line $L_{a,b}$, $\mathrm{sp}(L)$, with slope a and intercept b is the set of all effective dimensions of the points $(x, ax + b)$ on L. It has been recently shown that, for every a and b with effective dimension less than 1, the dimension spectrum of $L_{a,b}$ contains an interval. Our first main theorem shows that this holds for every line. Moreover, when the effective dimension of a and b is at least 1, $\mathrm{sp}(L)$ contains a *unit* interval.

Our second main theorem gives lower bounds on the dimension spectra of lines. In particular, we show that for every $\alpha \in [0, 1]$, with the exception of a set of Hausdorff dimension at most α, the effective dimension of $(x, ax + b)$ is at least $\alpha + \frac{\dim(a,b)}{2}$. As a consequence of this theorem, using a recent characterization of Hausdorff dimension using effective dimension, we give a new proof of a result by Molter and Rela on the Hausdorff dimension of Furstenberg sets.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic

Keywords and phrases algorithmic randomness, geometric measure theory, Hausdorff dimension, Kolmogorov complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.79

1 Introduction

This paper is concerned with the algorithmic dimension of points on a given line in the Euclidean plane. The most well-studied algorithmic dimensions for a point $x \in \mathbb{R}^n$ are the *effective Hausdorff dimension*, $\dim(x)$, and its dual, the *effective packing dimension*, $\mathrm{Dim}(x)$ [3, 1]. Given the pointwise nature of effective dimension, it is natural to consider the dimension spectrum, $\mathrm{sp}(A)$, of a set $A \subseteq \mathbb{R}^n$, which is defined to be the set of $\dim(x)$ for all $x \in A$.

In this paper, we study the behavior of $\mathrm{sp}(L_{a,b})$, where $L_{a,b}$ is the line with slope a and intercept b. Turetsky [13] gave the first result on the dimension spectra of lines, showing that, for every $n \geq 2$, the set of all points in $\mathbb{R}^n$ with effective Hausdorff 1 is connected, implying that $1 \in \mathrm{sp}(L_{a,b})$. It was then asked by J. Lutz, with the expectation of a negative answer, if there were lines in the plane whose dimension spectrum was the singleton $\{1\}$. N. Lutz and Stull [9] showed that this cannot happen by proving the following theorem.

▶ **Theorem 1** (N. Lutz and Stull [9]). *For all $a, b, x \in \mathbb{R}$,*

$$\dim(x, ax + b) \geq \dim^{a,b}(x) + \min\{\dim(a, b), \dim^{a,b}(x)\}.$$

Theorem 1 implies that, when $\dim(a, b) < 1$, the dimension spectrum of $L_{a,b}$ contains the interval $[2\dim(a, b), \dim(a, b) + 1]$. With this result, it is natural to conjecture that the dimension spectrum of every line $L_{a,b}$ contains an interval. Indeed, in a recent survey on effective dimension, N. Lutz [7] proposed the question of whether *every* line $L_{a,b}$ has a dimension spectrum containing a *unit* interval. Building upon the techniques of [9], N. Lutz and Stull [10] showed that this is the case for a restricted class of lines.

Table 1 Previously known results about the dimension spectra of lines.

$\forall a, b$	$1 \in \mathrm{sp}(L_{a,b})$ [13]
$\dim(a,b) = 2$	$\mathrm{sp}(L_{a,b}) = [1,2]$
$\dim(a,b) \geq 1$	$\mathrm{sp}(L_{a,b})$ infinite [10]
$\dim(a,b) = d < 1$	$[2d, 1+d] \subseteq \mathrm{sp}(L_{a,b})$ [9]
$\dim(a,b) = 0$	$\mathrm{sp}(L_{a,b}) = [0,1]$
$\dim(a,b) = \mathrm{Dim}(a,b) = d$	$[\min\{1,d\}, 1+\min\{1,d\}] \subseteq \mathrm{sp}(L_{a,b})$ [10]

▶ **Theorem 2** (N. Lutz and Stull [10]).
1. *If* $\dim(a,b) = \mathrm{Dim}(a,b)$*, then* $\mathrm{sp}(L_{a,b})$ *contains a unit interval.*
2. *If* $\dim(a,b) \geq 1$*, then* $\mathrm{sp}(L_{a,b})$ *is infinite.*

The second item, combined with Theorem 1, shows that for every line $L_{a,b}$, the dimension spectrum of $L_{a,b}$ is infinite. Table 1 gives a summary of these results.

The question of whether the dimension spectrum of every line contains an interval has remained open. Our first main theorem settles this question.

▶ **Theorem 3.** *Let* $(a,b) \in \mathbb{R}^2$ *such that* $\dim(a,b) \geq 1$*. Then, for every real number* $d \in [0,1]$*, there is a point* x *such that*

$$\dim(x, ax+b) = 1 + d.$$

Our second main theorem deals with providing lower bounds on the dimension spectrum of a given line in the plane. The previously discussed theorems have all focused on results proving that the spectrum of a given line contains certain values. However, very little is known about the lower bound of $\mathrm{sp}(L_{a,b})$ for arbitrary lines $L_{a,b}$. Our second main theorem gives a lower bound of the spectrum of arbitrary lines, disregarding a set of small Hausdorff dimension.

▶ **Theorem 4.** *For every* $a, b \in \mathbb{R}$ *and* $\alpha \in (0,1)$*, the set*

$$A = \{x \,|\, dim(x, ax+b) \leq \alpha + \frac{\dim(a,b)}{2}\}$$

has Hausdorff dimension at most α*.*

Apart from being intrinsically interesting, the study of the effective dimension of points on a line has strong connections to important problems in the field of Fractal Geometry. This connection is mediated by the following theorem relating the two notions of the effective dimension of points with the Hausdorff and packing dimension of sets.

▶ **Theorem 5** (Point-to-set principle [4]). *Let* $n \in \mathbb{N}$ *and* $E \subseteq \mathbb{R}^n$*. Then*

$$\dim_H(E) = \min_{A \subseteq \mathbb{N}} \sup_{x \in E} \dim^A(x), \text{ and}$$

$$\dim_P(E) = \min_{A \subseteq \mathbb{N}} \sup_{x \in E} \mathrm{Dim}^A(x).$$

Recent work has used effective dimension and the point-to-set principle to prove new results in Fractal Geometry [6, 8]. In particular, the point-to-set principle combined with Theorem 1 gives improved lower bounds on the Hausdorff dimension of a certain class of *Furstenberg sets* [9], an important open problem in Fractal Geometry (see Section 5 for definitions). As our final result, we show that our second main theorem, Theorem 4, gives a new proof of a result by Molter and Rela [12] on the dimension of Furstenberg sets.

2 Preliminaries

2.1 Kolmogorov Complexity in Discrete Domains

The *conditional Kolmogorov complexity* of $\sigma \in \{0,1\}^*$ given $\tau \in \{0,1\}^*$ is

$$K(\sigma|\tau) = \min_{\pi \in \{0,1\}^*} \{\ell(\pi) : U(\pi,\tau) = \sigma\} ,$$

where U is a fixed universal prefix-free Turing machine and $\ell(\pi)$ is the length of π. Any π that achieves this minimum is said to *testify* to the value $K(\sigma|\tau)$. The *Kolmogorov complexity* of σ is $K(\sigma) = K(\sigma|\lambda)$, where λ is the empty string. An important property, due to Levin, of Kolmogorov complexity is the *symmetry of information*:

$$K(\sigma|\tau, K(\tau)) + K(\tau) = K(\tau|\sigma, K(\sigma)) + K(\sigma) + O(1) .$$

Kolmogorov complexity extends naturally to other discrete domains (e.g., integers, rationals, etc.) via standard binary encodings.

We will also frequently use *relativized Kolmogorov complexity*. Letting U be a universal *oracle* machine, we may *relativize* the definition in this section to an arbitrary oracle set $A \subseteq \mathbb{N}$. The definitions of $K^A(\sigma|\tau)$ and $K^A(\sigma)$ are then identical to those above, except that U is given oracle access to A.

2.2 Kolmogorov Complexity in Euclidean Spaces

In this section we show how to lift the definition of Kolmogorogov complexity to Euclideans spaces by introducing precision parameters [5, 4]. Let $x \in \mathbb{R}^m$, and let $r, s \in \mathbb{N}$.[1]

The *Kolmogorov complexity of x at precision r* is

$$K_r(x) = \min \{K(p) \,:\, p \in B_{2^{-r}}(x) \cap \mathbb{Q}^m\} .$$

The *conditional Kolmogorov complexity of x at precision r given $q \in \mathbb{Q}^m$* is

$$\hat{K}_r(x|q) = \min \{K(p|q) \,:\, p \in B_{2^{-r}}(x) \cap \mathbb{Q}^m\} .$$

The *conditional Kolmogorov complexity of x at precision r given $y \in \mathbb{R}^n$ at precision s* is

$$K_{r,s}(x|y) = \max \left\{\hat{K}_r(x|q) \,:\, q \in B_{2^{-r}}(y) \cap \mathbb{Q}^n\right\} .$$

We abbreviate $K_{r,r}(x|y)$ by $K_r(x|y)$.

We will frequently use the following lemma, which shows that increasing an estimate of a point is at most linearly correlated with the number of extra bits.

▶ **Lemma 6** (Case and J. Lutz [2]). *There is a constant $c \in \mathbb{N}$ such that for all $n, r, s \in \mathbb{N}$ and $x \in \mathbb{R}^n$,*

$$K_r(x) \leq K_{r+s}(x) \leq K_r(x) + K(r) + ns + a_s + c ,$$

where $a_s = K(s) + 2\log(\lceil \frac{1}{2}\log n \rceil + s + 3) + (\lceil \frac{1}{2}\log n \rceil + 3)n + K(n) + 2\log n$.

In Euclidean spaces, we have a weaker version of symmetry of information.

[1] If we are given a nonintegral positive real as a precision parameter, we will always round up to the next integer. For example, $K_r(x)$ denotes $K_{\lceil r \rceil}(x)$ whenever $r \in (0, \infty)$.

▶ **Lemma 7** (J. Lutz and N. Lutz [4], N. Lutz and Stull [9]). *Let $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$. For all $r, s \in \mathbb{N}$ with $r \geq s$,*

1. $K_r(x, y) = K_r(x|y) + K_r(y) + O(\log r)$.
2. $K_r(x) = K_{r,s}(x|x) + K_s(x) + O(\log r)$.

As in the case of Kolmogorov complexity in discrete domains, we can relativize the definitions of this section. We define $K_r^A(x)$ and $K_r^A(x|y)$ as before, except we replace the unrelativized complexity K with K^A.

2.3 Effective Dimensions

Although effective Hausdorff dimension was initially developed by J. Lutz using generalized martingales [3], it was later shown by Mayordomo [11] that it may be equivalently defined using the Kolmogorov complexity of Euclidean points of the previous section. We will be using this Kolmogorov characterization here as a definition.

The *effective Hausdorff dimension* and *effective packing dimension* of a point $x \in \mathbb{R}^n$ are

$$\dim(x) = \liminf_{r\to\infty} \frac{K_r(x)}{r} \quad \text{and} \quad \mathrm{Dim}(x) = \limsup_{r\to\infty} \frac{K_r(x)}{r}.$$

Intuitively, these dimensions measure the density of algorithmic information in the point x. Recently, J. Lutz and N. Lutz [4], developed the *lower* and *upper conditional dimension* of points $x \in \mathbb{R}^m$ given $y \in \mathbb{R}^n$, defined by

$$\dim(x|y) = \liminf_{r\to\infty} \frac{K_r(x|y)}{r} \quad \text{and} \quad \mathrm{Dim}(x|y) = \limsup_{r\to\infty} \frac{K_r(x|y)}{r}.$$

Again, we can relativize the definitions of this section. We define $\dim^A(x)$, $\mathrm{Dim}^A(x)$ $\dim^A(x|y)$, and $\mathrm{Dim}^A(x|y)$ as before, except we replace the unrelativized complexity K_r with K_r^A.

Of particular importance in this paper is the complexity of a point x *relative to another point* y, written $K_r^y(x)$. This is achieved by encoding the binary expansion of y into an oracle $A_y \subseteq \mathbb{N}$ in the standard fashion. We then write $K_r^y(x)$ for $K_r^{A_y}(x)$. J. Lutz and N. Lutz showed that $K_r^y(x) \leq K_{r,t}(x|y) + K(t) + O(1)$ [4].

3 Dimension Spectra of Lines of High Dimension

3.1 Approach and Previous Work

In this section we state the technical lemmas that underlie the proof of our first main theorem (Theorem 3). These lemmas were first stated and proved by N. Lutz and Stull [9, 10][2].

▶ **Lemma 8.** *Let $a, b, x \in \mathbb{R}$, $k \in \mathbb{N}$. Suppose that $r_1, \ldots, r_k \in \mathbb{N}$, $\delta \in \mathbb{R}_+$, and $\varepsilon, \eta_1, \ldots, \eta_k \in \mathbb{Q}_+$ satisfy the following conditions for every $1 \leq i \leq k$.*

1. $r_i \geq \log(2|a| + |x| + 6) + r_{i-1}$.
2. $K_{r_i}(a, b) \leq (\eta_i + \varepsilon)\, r_i$.
3. *For every $(u, v) \in \mathbb{R}^2$ such that $t = -\log \|(a, b) - (u, v)\| \in (r_{i-1}, r_i]$ and $ux + v = ax + b$, $K_{r_i}(u, v) \geq (\eta_i - \varepsilon)\, r_i + \delta \cdot (r_i - t)$.*

[2] Lemma 8 is stated here in a slightly stronger form than the version of [10]. The proof, however, is nearly identical. For completeness we give a proof in the Technical Appendix.

Then for every oracle set $A \subseteq \mathbb{N}$,

$$K^A_{r_k}(a,b,x \mid x, ax+b) \leq 2^k \left(K(\eta_1,\dots,\eta_k) + K(\varepsilon) + \frac{4\varepsilon}{\delta} r_k + O(\log r_k) \right) .$$

We will briefly describe the intuition behind Lemma 8. For $k = 1$, Lemma 8 roughly states that, if x and (a,b) satisfy the following properties, then we can compute an approximation of (a,b) given an approximation of $(x, ax+b)$.

1. $K_r(a,b)$ is small.
2. For every (u,v) such that $ux+v = ax+b$ either
   - $K_r(u,v)$ is large, or
   - (u,v) is close to (a,b)

This follows outputting a (u,v) of low complexity such that $ux+v = ax+b$. Under the above assumptions, any such pair must be close to (a,b), and so we can recover (a,b) with a small amount of extra information. Roughly, when $k > 1$, we do this procedure iteratively. That is, we begin by computing (a,b) to precision r_1 in the manner described above. Having done so, we do the same procedure, except that we restrict to finding a pair (u,v) *within* 2^{-r_1} of (a,b), and so on. This is useful when we can only guarantee that $K_r(u,v)$ is large when (u,v) is somewhat close to (a,b), which is the case in the proof of Theorem 3.

The next two lemmas will ensure that item (1) and (2) hold for a given pair (a,b).

▶ **Lemma 9** (N. Lutz and Stull [9]). *Let $a,b,x \in \mathbb{R}$. For all $(u,v) \in \mathbb{R}^2$ such that $ux+v = ax+b$ and $t = -\log \|(a,b)-(u,v)\| \in (0,r]$,*

$$K_r(u,v) \geq K_t(a,b) + K^{a,b}_{r-t}(x) - O(\log r) .$$

▶ **Lemma 10** (N. Lutz and Stull [10]). *Let $z \in \mathbb{R}^n$, $\eta \in \mathbb{Q} \cap [0, \dim(z)]$, and $k \in \mathbb{N}$. For all $r_1,\dots,r_k \in \mathbb{N}$, there is an oracle $D = D(r_1,\dots,r_k,z,\eta)$ such that*

1. *For every $t \leq r_1$, $K^D_t(z) = \min\{\eta r_1, K_t(z)\} + O(\log r_k)$*
2. *For every $1 \leq i \leq k$,*

$$K^D_{r_i}(z) = \eta r_1 + \sum_{j=2}^{i} \min\{\eta(r_j - r_{j-1}), K_{r_j, r_{j-1}}(z \mid z)\} + O(\log r_k) .$$

3. *For every $t \in \mathbb{N}$ and $x \in \mathbb{R}$, $K^{z,D}_t(x) = K^z_t(x) + O(\log r_k)$.*

3.2 First Main Theorem

In this section we prove our first main theorem, Theorem 3. To do so, we will break the proof into two cases. In the first we assume that, for arbitrarily long intervals, $K_r(a,b)$ is arbitrarily close to 1. In this case, it is "locally" as if $\dim(a,b) = \mathrm{Dim}(a,b)$, and we can use a similar proof to that of Theorem 2 in [10]. This case is formalized in the following lemma, whose proof if deferred to the appendix.

▶ **Lemma 11.** *Let $(a,b) \in \mathbb{R}^2$ such that $\dim(a,b) \geq 1$. Assume that, for every $\tau > 0$ and every $M \in \mathbb{N}$ there are infinitely many $R \in \mathbb{N}$ such that*

$$K_s(a,b) \leq (1+\tau)s,$$

for every natural number $s \in [R, MR]$. Then, for every real number $d \in (0,1]$, there is a point x such that $\dim(x, ax+b) = 1+d$.

Proof. Let $d \in (0,1]$. For every $n \in \mathbb{N}$, let $\tau_n = \frac{1}{n}$ and $M_n = \frac{2^n}{d}$. Let $R_1, R_2, \ldots$ be a sequence of natural numbers such that the following hold.

1. $2^{R_n} < R_{n+1}$.
2. For every n, R_n satisfies the hypothesis for the choices of τ_n and M_n.

We now define a real number x such that

$$\dim(x, ax+b) = 1 + d. \tag{1}$$

Let $y \in \mathbb{R}$ be a real number that is random relative to (a,b). That is, for every $r \in \mathbb{N}$,

$$K_r^{a,b}(y) \geq r - \log r.$$

For every $n \in \mathbb{N}$, define $h_n = \frac{(M_n - 1)R_n}{2}$. For every $r \in \mathbb{N}$, let

$$x[r] = \begin{cases} 0 & \text{if } \frac{r}{h_n} \in (d,1] \text{ for some } n \in \mathbb{N} \\ y[r] & \text{otherwise} \end{cases}$$

where $x[r]$ is the rth bit of x. Define $x \in \mathbb{R}$ to be the real number with this binary expansion.

We first claim that the dimension of $(x, ax+b)$ is at most $1+d$. For every $n \in \mathbb{N}$, by our construction of x and choice of y,

$$\begin{aligned} K_{h_n}(x) &= K_{dh_n}(x) + O(\log h_n) \\ &= K_{dh_n}(y) + O(\log h_n) \\ &\leq dh_n + O(\log h_n). \end{aligned}$$

Therefore, by the above bound and Lemma 7,

$$\begin{aligned} \dim(x, ax+b) &= \liminf_{r\to\infty} \frac{K_r(x, ax+b)}{r} \\ &= \liminf_{r\to\infty} \frac{K_r(x) + K_r(ax+b \,|\, x) + O(\log r)}{r} \\ &\leq \liminf_{r\to\infty} \frac{K_r(x) + r + O(\log r)}{r} \\ &\leq \liminf_{n\to\infty} \frac{K_{h_n}(x) + h_n + O(\log h_n)}{h_n} \\ &= d + 1. \end{aligned}$$

To complete the proof, it suffices to show that, for every $\eta \in \mathbb{Q} \cap (0,1)$ and $\varepsilon \in \mathbb{Q}_+$,

$$\dim(x, ax+b) \geq \eta + d - \varepsilon. \tag{2}$$

To that end, let $\eta \in \mathbb{Q} \cap (0,1)$ and $\varepsilon \in \mathbb{Q}_+$. To prove inequality (2), we will partition $\mathbb{N}$ into intervals, and focus on the complexity of $(x, ax+b)$ at each precision r in these intervals. For every $n \in \mathbb{N}$, let $I_n = (dh_n, dh_{n+1}]$.

Fix $n \in \mathbb{N}$, and let $m = \frac{1-d}{1-\eta}$. We will first consider $r \in (dh_n, mh_n]$. Let $k = \frac{r}{dh_n}$, and define $r_i = idh_n$ for every $1 \leq i \leq k$. It is important to note that k is bounded by a constant depending only on η and d. In particular, this implies that $o(r_k)$ is sublinear for all r_i. Let $D_r = D(r_1, \ldots, r_k, a, b, \eta)$ be the oracle defined in Lemma 10. We first note that, by our assumption of (a,b) on the interval $[R_n, MR_n]$ and Lemma 7,

$$\begin{aligned} K_{r_i, r_{i-1}}(a, b \,|\, a, b) &= K_{r_i}(a,b) - K_{r_{i-1}}(a,b) - O(\log r_i) \\ &\geq r_i - o(r_i) - (1 + \frac{1}{n}) r_{i-1} - O(\log r_i) \\ &= r_i - r_{i-1} - \frac{r_{i-1}}{n} - o(r_i), \end{aligned}$$

for all sufficiently large r. Since k is bounded by a constant, for all sufficiently large n, we have

$$K_{r_i,r_{i-1}}(a,b \,|\, a,b) > \eta(r_i - r_{i-1}) - o(r_i).$$

Hence, by Lemma 10,

$$|K^{D_r}_{r_i}(a,b) - \eta r_i| < o(r_k), \tag{3}$$

for all $1 \leq i \leq k$.

We now show that the conditions of Lemma 8 are satisfied relative to D_r. Item 1 of Lemma 8 holds for all sufficiently large r. For item 2, by the construction of D_r, for every $1 \leq i \leq k$,

$$\begin{aligned} K^{D_r}_{r_i}(a,b) &= \eta r_1 + \sum_{j=2}^{i} \min\{\eta(r_j - r_{j-1}), K_{r_j,r_{j-1}}(z \,|\, z)\} + O(\log r_k) \\ &\leq \eta r_1 + \sum_{j=2}^{i} \eta(r_j - r_{j-1} + O(\log r_k) \\ &\leq \eta r_i + O(\log r_k) \\ &\leq (\eta + \varepsilon) r_i, \end{aligned}$$

for all sufficiently large r.

Let $\delta = 1 - \eta$. To see that item 3 of Lemma 8 is satisfied for $i = 1$, let $(u,v) \in B_1(a,b)$ such that $ux + v = ax + b$ and $t = -\log \|(a,b) - (u,v)\| \leq r_1$. Then, by Lemmas 9 and 10, and our construction of x,

$$\begin{aligned} K^{D_r}_{r_1}(u,v) &\geq K^{D_r}_{t}(a,b) + K^{D_r}_{r_1-t,r_1}(x|a,b) - O(\log r_1) \\ &\geq \min\{\eta r_1, K_t(a,b)\} + K_{r_1-t}(x) - o(r_k) \\ &\geq \min\{\eta r_1, t - o(t)\} + (\eta + \delta)(r_1 - t) - o(r_k) \\ &\geq \min\{\eta r_1, \eta t - o(t)\} + (\eta + \delta)(r_1 - t) - o(r_k) \\ &\geq \eta t - o(t) + (\eta + \delta)(r_1 - t) - o(r_k)\,, \end{aligned}$$

We conclude that $K^{D_r}_{r_1}(u,v) \geq (\eta - \varepsilon) r_1 + \delta(r_1 - t)$, for all sufficiently large r. To see that that item 3 is satisfied for $1 < i \leq k$, let $(u,v) \in B_{2^{-r_{i-1}}}(a,b)$ such that $ux + v = ax + b$ and $t = -\log \|(a,b) - (u,v)\| \leq r_i$. Since $(u,v) \in B_{2^{-r_{i-1}}}(a,b)$,

$$r_i - t \leq r_i - r_{i-1} = idh_j - (i-1)dh_j \leq dh_j + 1 \leq r_1 + 1\,.$$

Therefore, by Lemma 9, inequality (3), and our construction of x,

$$\begin{aligned} K^{D_r}_{r_i}(u,v) &\geq K^{D_r}_{t}(a,b) + K^{D_r}_{r_i-t,r_i}(x|a,b) - O(\log r_i) \\ &\geq \min\{\eta r_i, K_t(a,b)\} + K_{r_i-t}(x) - o(r_i) \\ &\geq \min\{\eta r_i, t - o(t)\} + (\eta + \delta)(r_i - t) - o(r_i) \\ &\geq \min\{\eta r_i, \eta t - o(t)\} + (\eta + \delta)(r_i - t) - o(r_i) \\ &\geq \eta t - o(t) + (\eta + \delta)(r_i - t) - o(r_i)\,. \end{aligned}$$

We conclude that $K^{D_r}_{r_i}(u,v) \geq (\eta - \varepsilon) r_i + \delta(r_i - t)$, for all sufficiently large r. Hence the conditions of Lemma 8 are satisfied. Therefore, by applying Lemma 8 and appealing to

inequality (3),

$$\begin{aligned}
K_r(x, ax+b) \geq & K_r^{D_r}(x, ax+b) \\
\geq & K_r(a,b,x) - 2^k \left(K(\varepsilon) + K(\eta) + \frac{4\varepsilon}{1-\eta} r_k + O(\log r_k) \right) \\
= & K_r(a,b) + K_r(x \,|\, a,b) \\
& - 2^k \left(K(\varepsilon) + K(\eta) + \frac{4\varepsilon}{1-\eta} r_k + O(\log r_k) \right) \\
\geq & dr + \eta r - 2^k \left(K(\varepsilon) + K(\eta) + \frac{4\varepsilon}{1-\eta} r_k + O(\log r_k) \right).
\end{aligned}$$

To complete the proof, we give lower bounds of $K_r(x, ax+b)$ for every $r \in [mh_n, dh_{n+1})$. By Lemma 7 and our construction of x,

$$\begin{aligned}
K_r(x) &= K_{r,h_n}(x \,|\, x) + K_{h_n}(x) - o(r) \\
&= r - h_n + dh_n - o(r) \\
&\geq \eta r - o(r).
\end{aligned}$$

The proof of Theorem 1 gives

$$\begin{aligned}
K_r(x, ax+b) &\geq K_r(x) + \dim(x) r - o(r) \\
&\geq \eta r + dr - o(r) \\
&\geq r(d+\eta) - \varepsilon r.
\end{aligned}$$

Putting together the lower bounds of $K_r(x, ax+b)$ on the intervals (dh_n, mh_n) and $[mh_n, dh_{n+1}]$ shows that

$$\dim(x, ax+b) \geq 1 + d,$$

and the proof is complete. ◀

If (a,b) is not of the first case, then there is a bound $(1+\tau) > 1$ so that $K_r(a,b) \geq r(1+\tau)$ for some r in every sufficiently large interval. This implies that, for almost every precision r, the conditional complexity $K_{s,r}(a,b \,|\, a,b) > s - r$, for some s at most a constant multiple of r. This fact allows us to use the procedure outlined in Section 3.1 at precision s. We will now formalize this intuition.

▶ **Theorem 3.** *Let $(a,b) \in \mathbb{R}^2$ such that $\dim(a,b) \geq 1$. Then, for every real number $d \in [0,1]$, there is a point x such that*

$$\dim(x, ax+b) = 1 + d.$$

Proof. Let $(a,b) \in \mathbb{R}^2$ such that $\dim(a,b) \geq 1$. For $d = 1$, we may choose an $x \in \mathbb{R}$ that is random relative to (a,b). That is, there is some constant $c \in \mathbb{N}$ such that for all $r \in \mathbb{N}$, $K_r^{a,b}(x) \geq r - c$. By Theorem 1,

$$\begin{aligned}
\dim(x, ax+b) &\geq \dim^{a,b}(x) + \min\{\dim(a,b), 1\} \\
&= \liminf_{r\to\infty} \frac{K_r(x)}{r} + 1 \\
&= 2,
\end{aligned}$$

and the conclusion holds. For $d = 0$, the conclusion follows from Turetsky's theorem [13]. We therefore assume that $d \in (0, 1)$.

If (a, b) satisfy the conditions of Lemma 11, then the conclusion is immediate. So assume that the conditions of Lemma 11 do not hold. Let $\tau > 0$ and $M > 0$ be constants such that, for almost every $R \in \mathbb{N}$,

$$K_s(a, b) > (1 + \tau)s,$$

for some $s \in [R, MR]$.

Let $y \in \mathbb{R}$ be random relative to (a, b). Define the sequence of natural numbers $\{h_n\}_{n\in\mathbb{N}}$ inductively as follows. Define $h_0 = 1$. For every $n > 0$, let

$$h_n = \min\left\{h \geq 2^{h_{n-1}} : K_h(a, b) \geq \left(\operatorname{Dim}(a, b) - \frac{1}{n}\right) h\right\}.$$

Note that h_n always exists. For every $r \in \mathbb{N}$, let

$$x[r] = \begin{cases} 0 & \text{if } \frac{r}{h_n} \in (d, 1] \text{ for some } n \in \mathbb{N} \\ y[r] & \text{otherwise} \end{cases}$$

where $x[r]$ is the rth bit of x. Define $x \in \mathbb{R}$ to be the real number with this binary expansion. Then $K_{dh_n}(x) = dh_n + O(\log dh_n)$.

Claim 1: $\dim(x, ax + b) \leq 1 + d$.

Let $\eta \in \mathbb{Q} \cap (0, 1)$, $\varepsilon \in \mathbb{Q}$, $n \in \mathbb{N}$, and let $m = \frac{1-d}{1-\eta}$. We first give lower bounds of the complexity of $K_r(x, ax + b)$ on the interval (dh_n, mh_n). To begin, consider $r = h_n$. Let $k = \frac{r}{dh_n} = \frac{1}{d}$, and define $r_i = idh_n$ for every $1 \leq i \leq k$. As in the proof of Lemma 11, is important to note that k is bounded by a constant depending only on η and d.

Claim 2: $K_{h_n}(x, ax + b) \geq dh_n + \eta h_n - 2^k \left(K(\varepsilon) + kK(\eta) + \frac{4\varepsilon}{1-\eta} h_n + O(\log h_n)\right)$.

With this bound on the complexity of $(x, ax + b)$ at precision h_n, we will use a symmetry of information argument to give a lower bound on the complexity at precision $r \in (dh_n, h_n)$. We defer the proof of this claim to the appendix.

Claim 3: For all $r \in [dh_n, h_n)$,

$$K_r(x, ax + b) \geq r(d + \eta) - 2^k \left(K(\varepsilon) + K(\eta) + \frac{4\varepsilon}{1 - \eta} h_n + O(\log h_n)\right).$$

Note that this lower bound is useful for $r \in (dh_n, h_n)$, since h_n is a fixed constant multiple of r.

We now turn to proving lower bounds for the complexity of $(x, ax + b)$ on the interval (h_n, mh_n). To do so, we will make use of our assumption that the complexity of $K_s(a, b)$ is at least $(1 + \tau)s$. In particular, this assumption implies that there is a *fixed* constant c such that $K_{ch_n,h_n}(a, b \,|\, a, b) \geq \eta(ch_n - h_n)$. Moreover, this constant is independent of η and ε. To see this, let $s > h_n$ be a precision such that $K_s(a, b) \geq (1 + \tau)s$. By Lemma 7 and our assumption of a, b,

$$\begin{aligned} K_{s,h_n}(a, b \,|\, a, b) &\geq K_s(a, b) - K_{h_n}(a, b) - O(\log s) \\ &\geq (1 + \tau)s - \operatorname{Dim}(a, b)h_n - O(\log s) \\ &\geq (1 + \tau)s - 2h_n - O(\log s). \end{aligned}$$

Thus,

$$K_{s,h_n}(a, b \,|\, a, b) \geq \eta(s - h_n),$$

for any such $s > ch_n$, for some fixed constant c depending only on τ. With this fact we are able to show the following, whose proof is deferred to the appendix.

Claim 4: There is a precision $h_n < j \leq ch_n$ such that

$$K_j(x, ax+b) \geq j - (h_n - dh_n) + \eta j - 2^k \left(K(\varepsilon) + kK(\eta) + \frac{4\varepsilon}{1-\eta} j + O(\log j) \right).$$

With this bound, we will again prove lower bounds at precisions $r \in (h_n, j)$ using symmetry of information arguments. While this is similar in spirit to the proof of Claim 3, there is an important difference. At precisions greater than h_n, the complexity of x begins increasing again. In particular, the construction of x and Lemma 6 implies the following.

$$K_j(x, ax+b) \leq K_r(x, ax+b) + 2(j-r).$$

We are still, however, able to achieve the required lower bounds on the complexity of $(x, ax+b)$ for all $r \in (h_n, j)$. The proof of this claim is deferred to the appendix.

Claim 5: For every $r \in (h_n, j)$,

$$K_r(x, ax+b) \geq r(d+\eta) - cr(1-\eta) - 2^k \left(K(\varepsilon) + K(\eta) + \frac{4\varepsilon}{1-\eta} cr + O(\log cr) \right).$$

This lower bound is useful since $j \leq ch_n$, and c is a constant depending only on τ. In particular, this allows us to make $\frac{cr(1-\eta)}{r}$ arbitrarily small by having η go to 1.

To complete the proof for the interval (dh_n, mh_n), we will apply the same method as in Claims 4 and 5, except that we use j instead of h_n. Specifically, we choose the first $j_2 > j$ such that

$$K_{j_2,j}(a, b \,|\, a, b) \geq \eta(j_2 - j),$$

and note that $j_2 \leq cj$. We then apply the proof of Claim 4 to $K_{j_2}(x, ax+b)$, and the proof of Claim 5 to the interval (j, j_2). We then repeat this argument until we have given the appropriate lower bound for all $r \in (h_n, mh_n)$.

Finally, taking Claims 1, 2, 3, 4 and 5 together yields the following. For every $r \in (dh_n, m_n)$,

$$K_r(x, ax+b) \geq r(d+\eta) - cr(1-\eta) - 2^k \left(K(\varepsilon) + K(\eta) + \frac{4\varepsilon}{1-\eta} cr + O(\log cr) \right). \quad (4)$$

To complete the proof, we bound $K_r(x, ax+b)$ for every $r \in [mh_n, dh_{n+1})$. By Lemma 7 and our construction of x,

$$\begin{aligned} K_r(x) &= K_{r,h_n}(x \,|\, x) + K_{h_n}(x) + o(r) \\ &= r - h_n + nh_n + o(r) \\ &\geq \eta r + o(r). \end{aligned}$$

The proof of Theorem 1 gives $K_r(x, ax+b) \geq K_r(x) + \dim(x)r - o(r)$, and so $K_r(x, ax+b) \geq r(d+\eta) - \varepsilon r$. Combined with inequality (4), for every $r \in (dh_n, dh_{n+1})$,

$$\begin{aligned} \frac{K_r(x, ax+b)}{r} &\geq r(d+\eta) - cr(1-\eta) - 2^k \left(K(\varepsilon) + K(\eta) + \frac{4\varepsilon}{1-\eta} cr + O(\log cr) \right) \\ &= d + \eta - 2^k \left(\frac{K(\varepsilon)}{r} + \frac{K(\eta)}{r} + \frac{4\varepsilon}{1-\eta} c + \frac{O(\log cr)}{r} \right) \\ &\geq d + \eta - 2^{\frac{m}{d}} \left(\frac{K(\varepsilon)}{r} + \frac{K(\eta)}{r} + \frac{4\varepsilon}{1-\eta} c + \frac{O(\log cr)}{r} \right) \end{aligned}$$

for all sufficiently large n. Since η and ε were chosen arbitrarily and independently,

$$\dim(x, ax+b) \geq d+1,$$

and the proof is complete. ◀

4 Lower Bounding the Dimension Spectrum of a Line

In this section we give the first nontrivial lower bounds of the dimension spectrum of an arbitrary line. For intuition behind the proof, first note the following simple observation.

▶ **Observation 12.** *For every $x, y, a, b \in \mathbb{R}$,*

$$K_r(x, y, a, b) \leq K_r(x, ax+b) + K_r(y, ay+b) + 2t,$$

where $t = -\log \|x-y\|$.

Essentially, this is true since any two points identify a line, and this can be done in a computable way. The $2t$ extra information is due to the fact the precision which we can compute (a, b) to is linearly correlated to the distance between x and y. This immediately suggests an approach to give the lower bound

$$\dim(x) + \frac{\dim(a,b)}{2} \geq \dim(a,b).$$

While this observation is at the core of the proof of our second main theorem, it alone does not suffice. The principle issue is that the values of $K_r(x, ax+b)$, $K_r(y, ay+b)$ might be "out of phase"; that is, $K_r(x, ax+b)$ is small when $K_r(y, ay+b)$ is large, and vice versa. Our main theorem will show that the set of these points has low Hausdorff dimension.

Our first lemma builds upon Observation 12. In particular, it shows that, if $K_r(x, ax+b)$ is small, then every other y such that $K_r(y, ay+b)$ is small must satisfy certain properties.

▶ **Lemma 13.** *Let $\alpha \in (0,1)$, $x, a, b \in \mathbb{R}$, and $n, r \in \mathbb{N}$ such that $2r^{-\frac{1}{2}} < \frac{1}{n}$. Assume that $K_r(x, ax+b) < \alpha r + \frac{K_r(a,b)}{2} - \frac{r}{n}$, and $K_r^{a,b}(x) \geq \alpha r$. Then, for every $y \in \mathbb{R}$, if $K_r(y, ay+b) < \alpha r + \frac{K_r(a,b)}{2}$, at least one of the following holds.*

1. *$t := -\log \|x-y\| \leq r^{\frac{1}{2}}$.*
2. *$K_r(y \mid a, b, x) < \alpha r$.*

Proof. Assume the hypothesis, but assume that neither condition is satisfied for some y. Then,

$$\begin{aligned} K_r(a, b, x, y) &\leq K_r(x, ax+b) + K_r(y, ay+b) + 2t \\ &< 2\alpha r + K_r(a,b) - \frac{r}{n} + 2t. \end{aligned}$$

However, by our hypothesis and Lemma 7 we have

$$\begin{aligned} K_r(a, b, x, y) &\geq K_r(a,b) + K_r(x \mid a, b) + K_r(y \mid a, b, x) - \log r \\ &\geq K_r(a,b) + 2\alpha r - \log r. \end{aligned}$$

Since condition (1) was assumed to not hold, we see that

$$\frac{1}{n} < 2r^{-\frac{1}{2}},$$

a contradiction. ◀

For every $a, b \in \mathbb{R}$ and $\alpha \in (0,1)$, define the set

$$A(\alpha, a, b) = \{x \mid \dim(x, ax+b) < \alpha + \frac{\dim(a,b)}{2}\}.$$

▶ **Theorem 4.** *For every $a, b \in \mathbb{R}$ and $\alpha \in (0,1)$, $\dim_H(A(\alpha, a, b)) \leq \alpha$.*

Proof. Our goal is to show that $\dim_H(A(\alpha, a, b) \leq \alpha$. We will actually prove a stronger theorem. For every n, define

$$A_n(\alpha, a, b) = \{x \mid (\exists^\infty r)\, K_r(x, ax+b) < \alpha r + \frac{K_r(a,b)}{2} - \frac{r}{n}\}.$$

Note that to prove $\dim_H(A(\alpha, a, b) \leq \alpha$, it suffices to show that $\dim_H(A_n(\alpha, a, b)) \leq \alpha$ for every n. To see that $A(\alpha, a, b) \subseteq \cup_n A_n(\alpha, a, b)$, let $x \in A$. Then there is an $\epsilon > 0$ such that, for infinitely many r,

$$\begin{aligned} K_r(x, ax+b) &< \alpha r + \frac{\dim(a,b)}{2} r - \epsilon r \\ &< \alpha r + \frac{K_r(a,b) + g(r)}{2} - \epsilon r, \end{aligned}$$

where g is a sublinear function. Therefore, for sufficiently large n and r, $x \in A_n(\alpha, a, b)$. Since the Hausdorff dimension of a countable union of sets $\cup_n A_n$ is the supremum of $\dim_H(A_n)$, it suffices to show that, for every n, $\dim_H(A_n(\alpha, a, b)) \leq \alpha$.

Define the set

$$U = \{x \mid (\exists^\infty r)\, K_r^{a,b}(x) \leq \alpha r\}.$$

It is immediate that $\dim^{a,b}(x) \leq \alpha$, for all $x \in U$.

For every $r \in \mathbb{N}$, choose x_r such that

$$K_r^{a,b}(x) \geq \alpha r$$
$$K_r(x, ax+b) < \alpha r + \frac{K_r(a,b)}{2} - \frac{r}{n},$$

if such an x_r exists. To reduce the notational burden we will, without loss of generality, always assume that such an x_r does exist. We then define

$$V = \{y \mid (\exists^\infty r)\, y \in (x_r - 2^{-r^{\frac{1}{2}}}, x_r + 2^{-r^{\frac{1}{2}}})\}.$$

Define oracle $R \subseteq \mathbb{N}$ which encodes the sequence $x_1, x_2, \ldots$ in the standard manner. Let $y \in V$, and let $r \in \mathbb{N}$ such that $y \in (x_r - 2^{-r^{\frac{1}{2}}}, x_r + 2^{-r^{\frac{1}{2}}})$. Then,

$$\begin{aligned} K^R_{r^{\frac{1}{2}}}(y) &\leq O(\log r) \\ &= O(\log r^{\frac{1}{2}}). \end{aligned}$$

Thus

$$\dim^R(y) = 0.$$

Let $x_1, x_2, \ldots$ be the sequence chosen above. Define

$$W = \{y \mid (\exists^\infty r)\, K_r(y \mid a, b, x_r) < \alpha r\}.$$

Let $y \in W$, and let $r \in \mathbb{N}$ such that $K_r(y \mid a, b, x_r) < \alpha r$. Then we have

$$\begin{aligned} K_r^{R,a,b}(y) &\leq K_r(y \mid a, b, x_r) + O(\log r) \\ &< \alpha r + O(\log r). \end{aligned}$$

Thus

$$\dim^{R,a,b}(y) \leq \alpha.$$

We now show that $A_n(\alpha, a, b) \subseteq U \cup V \cup W$. Let $y \in A_n(\alpha, a, b)$, and assume that $y \notin U \cup V$. So then y has the following properties.

1. For infinitely many r, $K_r(y, ay + b) < \alpha r + \frac{K_r(a,b)}{2} - \frac{r}{n}$
2. For almost every r, $K_r^{a,b}(y) > \alpha r$.
3. For almost every r, $y \notin (x_r - 2^{-r\frac{1}{2}}, x_r + 2^{-r\frac{1}{2}})$.

Let $r \in \mathbb{N}$ be a sufficiently large integer such that item (1) holds. Then by Lemma 13, we must have that

$$K_r(y \mid a, b, x_r) < \alpha r.$$

Therefore, $y \in W$, and $A_n(\alpha, a, b) \subseteq U \cup V \cup W$. Hence $\dim^{R,a,b}(y) \leq \alpha$, and the proof is complete. ◀

5 Applications to Furstenberg Sets

In this section we will use the point-to-set principle, Theorem 5, in conjunction with the theorem of the previous section to give a new proof of a result by Molter and Rela on Furstenberg sets. Let $\alpha \in [0, 1]$. A *set of Furstenberg type* with parameter α is a set $E \subseteq \mathbb{R}^2$ such that, for every $e \in S^1$, there is a line ℓ_e in the direction e satisfying $\dim_H(E \cap \ell_e) \geq \alpha$. It is an important open problem in Fractal Geometry to find the minimum possible dimension of a set of Furstenberg type with paramater α.

Molter and Rela [12] have recently introduced a generalization of Furstenberg sets, by removing the restriction that *every* direction must intersect the set. A set $E \subseteq \mathbb{R}^2$ is in the class $F_{\alpha\beta}$ if there is some set $J \subseteq S^1$ such that

1. $\dim_H(J) \geq \beta$, and
2. for every $e \in J$, there is a line ℓ_e in the direction e satisfying $\dim_H(E \cap \ell_e) \geq \alpha$.

They proved the following lower bound on the dimension of such sets.

▶ **Theorem 14.** (Molter and Rela [12]) *For all $\alpha, \beta \in (0, 1]$ and every set $E \in F_{\alpha\beta}$,*

$$\dim_H(E) \geq \alpha + \frac{\beta}{2}.$$

We will now give a new proof of this theorem, using the theorems of the previous section.

Proof of Theorem 14. Let $\alpha, \beta \in (0, 1]$, $\epsilon > 0$, and $E \in F_{\alpha\beta}$. Let $A \subseteq \mathbb{N}$ be an oracle testifying to the Hausdorff dimension of E; i.e.,

$$\dim_H(E) = \sup_{z \in E} \dim^A(z).$$

Let $e \in S^1$ satisfy $\dim^A(e) > \beta - \epsilon$. Note that such a direction exists by the point-to-set principle. Let l_e be a line in direction e such that $\dim_H(l_e \cap E) \geq \alpha$. Let $a, b \in \mathbb{R}$ be the reals such that $L_{a,b} = l_e$. Note that $\dim^A(a, b) = \dim^A(e)$ because the mapping $e \mapsto a$ is

Table 2 Updated table of the dimension spectra of lines.

$\forall a, b$	$1 \in \text{sp}(L_{a,b})$	
$\dim(a,b) = 2$	$\text{sp}(L_{a,b}) = [1,2]$	
$\dim(a,b) \geq 1$	$[1,2] \subseteq \text{sp}(L_{a,b})$	$\text{sp}(L_{a,b}) \subseteq [1,2]$ Except for a set of dimension at most $\frac{1}{2}$
$\dim(a,b) = d < 1$	$[2d, 1+d] \subseteq \text{sp}(L_{a,b})$,	$\text{sp}(L_{a,b}) \subseteq [d, 1+d]$ Except a set of dimension at most $\frac{d}{2}$
$\dim(a,b) = 0$	$\text{sp}(L_{a,b}) = [0,1]$	
$\dim(a,b) = \text{Dim}(a,b)$	$[d, 1+d\}] \subseteq \text{sp}(L_{a,b})$, $d = \min\{1, \dim(a,b)\}$	

computable and bi-Lipschitz in a neighborhood of e. Let $S = \{x \mid (x, ax+b) \in E \cap l_e\}$. Note that this implies that $\dim_H(S) \geq \alpha$. We then have that

$$\dim_H(E) \geq \sup_{x \in S} \dim^A(x, ax+b).$$

Therefore, to complete the proof, it suffices to show that there exists a point $x \in S$ such that

$$\dim^A(x, ax+b) \geq \alpha + \frac{\beta}{2} - \epsilon. \tag{5}$$

By Theorem 4, relativized to A, the set of all x such that $\dim^A(x, ax+b) \leq \alpha + \frac{\beta}{2}$ has Hausdorff dimension at most $\alpha - \epsilon$. Since $\dim_H(S) \geq \alpha$, this implies that there is a point $x \in S$ which satisfies (5), and the proof is complete. ◀

6 Conclusion and Future Directions

In this paper, we have given two new results on the dimension spectra of lines in the plane, summarized in Table 2.

The first gives a partial answer to the question posed by Lutz, asking whether, for every line $L_{a,b}$, $\text{sp}(L_{a,b})$ contains a unit interval. We showed that if $\dim(a,b) \geq 1$, then this is true. Together with a previous result of N. Lutz and Stull, this implies the following.

▶ **Corollary 15.** *For every $a, b \in \mathbb{R}$, $\text{sp}(L_{a,b})$ contains an interval.*

However we still do not have complete answer to Lutz's question. This is an important open problem, and one that seems to require new techniques to solve.

We have also given the first nontrivial lower bound on the dimension spectrum of a given line. An important open problem is to improve the bounds given here. This would be not only an intrinsically interesting result, but would likely give improved bounds on Furstenberg sets. Another interesting direction for future research is to construct lines with "many" points of small dimension. In particular, for a given $\alpha > 0$, is there a line $L_{a,b}$ such that

$$\dim_H(A(\alpha, a, b)) = \alpha?$$

References

1. Krishna B. Athreya, John M. Hitchcock, Jack H. Lutz, and Elvira Mayordomo. Effective strong dimension in algorithmic information and computational complexity. *SIAM J. Comput.*, 37(3):671–705, 2007.
2. Adam Case and Jack H. Lutz. Mutual dimension. *ACM Transactions on Computation Theory*, 7(3):12, 2015.
3. Jack H. Lutz. The dimensions of individual strings and sequences. *Inf. Comput.*, 187(1):49–79, 2003.
4. Jack H. Lutz and Neil Lutz. Algorithmic information, plane Kakeya sets, and conditional dimension. *ACM Transactions on Computation Theory*, to appear.
5. Jack H. Lutz and Elvira Mayordomo. Dimensions of points in self-similar fractals. *SIAM J. Comput.*, 38(3):1080–1112, 2008.
6. Neil Lutz. Fractal intersections and products via algorithmic dimension. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, pages 58:1–58:12, 2017.
7. Neil Lutz. Some open problems in algorithmic fractal geometry. *Open Problem Column, SIGACT News*, 48(4), 2017.
8. Neil Lutz and DM Stull. Projection theorems using effective dimension. *arXiv preprint arXiv:1711.02124*, 2017.
9. Neil Lutz and Donald M. Stull. Bounding the dimension of points on a line. In *Theory and Applications of Models of Computation - 14th Annual Conference, TAMC 2017, Bern, Switzerland, April 20-22, 2017, Proceedings*, pages 425–439, 2017.
10. Neil Lutz and Donald M. Stull. Dimension spectra of lines. In *Unveiling Dynamics and Complexity - 13th Conference on Computability in Europe, CiE 2017, Turku, Finland, June 12-16, 2017, Proceedings*, pages 304–314, 2017.
11. Elvira Mayordomo. A Kolmogorov complexity characterization of constructive Hausdorff dimension. *Inf. Process. Lett.*, 84(1):1–3, 2002.
12. Ursula Molter and Ezequiel Rela. Furstenberg sets for a fractal set of directions. *Proc. Amer. Math. Soc.*, 140:2753–2765, 2012.
13. Daniel Turetsky. Connectedness properties of dimension level sets. *Theor. Comput. Sci.*, 412(29):3598–3603, 2011.

Tight Bounds for Deterministic h-Shot Broadcast in Ad-Hoc Directed Radio Networks

Aris Pagourtzis[1]
School of ECE, National Technical University of Athens, Athens, Greece
pagour@cs.ntua.gr
https://orcid.org/0000-0002-6220-3722

Tomasz Radzik[2]
Department of Informatics, King's College London, London, the United Kingdom
tomasz.radzik@kcl.ac.uk
https://orcid.org/0000-0002-7776-5461

Abstract

We consider the classical broadcast problem in ad-hoc (that is, unknown topology) directed radio networks with no collision detection, under the additional assumption that at most h transmissions (shots) are available per node. We focus on adaptive deterministic protocols for small values of h. We provide asymptotically matching lower and upper bounds for the cases $h = 2$ and $h = 3$. While for $h = 2$ our bound is quadratic, similar to the bound obtained for oblivious protocols, for $h = 3$ we prove a sub-quadratic bound of $\Theta(n^2 \log\log n / \log n)$, where n is the number of nodes in the network. The latter is the first result showing an adaptive algorithm which is asymptotically faster than oblivious h-shot broadcast protocols, for which a tight quadratic bound is known for every constant h. Our upper bound for $h = 3$ is constructive, making use of constructions of graphs with large girth. We also show an improved upper bound of $O(n^{1+\alpha/\sqrt{h}})$ for $h \geq 4$, where α is an absolute constant independent of h. Our upper bound for $h \geq 4$ is non-constructive.

2012 ACM Subject Classification Theory of computation → Distributed algorithms, Mathematics of computing → Extremal graph theory

Keywords and phrases Ad-hoc radio networks, wireless networks, deterministic broadcast, adaptive protocols, limited transmissions

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.80

1 Introduction

1.1 Model of broadcast with limited transmissions per node

In this paper a *transmission network* is a directed graph $G = (V, E)$ with the set of nodes $V = \{0, 1, \ldots, n-1\}$, where node 0 is the *source node*, denoted also by s, and all other nodes are reachable from this node. Initially each node knows only its identifier and the size n of the network. The source node knows also the *message*, which is to be broadcast to all other nodes. Let $\mathcal{G} \equiv \mathcal{G}^{(n)}$ denote the family of all transmission networks of size n.

We consider the following model of *h-shot broadcast*. Nodes of the network transmit in globally synchronized steps (counted from 1), with each node transmitting in at most h steps. If a node v transmits in a given step, then each node w such that $(v, w) \in E$ receives the

[1] Research mainly conducted while this author was visiting the Department of Informatics, King's College London, on sabbatical leave from the National Technical University of Athens.
[2] Research supported by EPSRC grant EP/M005038/1.

transmitted message, unless a *collision* occurs at node w, that is, unless there is another edge (v', w), $v' \neq v$, with node v' transmitting in the same step. We assume that there is no collision detection: a node w cannot distinguish between no transmission from any of the neighbouring nodes and simultaneous transmissions by two or more neighbouring nodes. The only node transmitting in step 1 is the source node 0 and a node can transmit in the current step $t \geq 2$ only if it has already received the message in previous steps.

Most of the research on communication protocols for various models of radio networks has been concerned with minimizing the number of steps, without putting constraints on the number of transmissions by individual nodes or the total number of transmissions by all nodes. Limiting the maximum number of transmissions per node has received somewhat less attention, especially in the context of ad-hoc (that is, unknown) networks. This objective, however, may be important in practice, since it may mean limiting the maximum energy usage per node to keep all nodes alive for as long as possible.

An h-shot broadcast protocol can be viewed as a function $\Pi \equiv \Pi_n$ which for any node v, a time step $t \geq 1$, and the knowledge κ gathered by node v in steps $1, 2, \ldots, t-1$, tells node v whether it transmits in step t. The protocol has to ensure that, within all constraints of the model, for each transmission network $G \in \mathcal{G}$, all nodes eventually receive the message, that is, broadcast is always eventually completed. The design objective is to keep the worst-case completion time as small as possible.

An *oblivious h-shot protocol* is defined by a sequence of *transmission sets* $S_1 = \{0\}, S_2, S_3, \ldots$, which are subsets of the node set V. Once a node v receives the message in step t, it wakes up and transmits in the first h steps $\tau_i \geq t+1, 1 \leq i \leq h$, such that $v \in S_{\tau_i}$. The source node 0 is considered awake at time 0 and transmitting at step 1. (We remark that slightly different definitions of obliviousness may be used in other variants of radio network models.)

In a general *(adaptive) h-shot protocol*, nodes can take into account information which they have received in earlier steps when they decide whether to transmit in the current step. We do not put any limits on how much information can be transmitted in one step or stored in one node. In fact, for our lower bounds we assume that during a successful transmission from a node v to a node w, all knowledge accumulated so far by node v is transmitted to node w and is added to w's knowledge. We remark though that the (adaptive) protocols for our upper bounds include in the transmissions only the source message and the current count of step. They achieve a speed-up over oblivious protocols by using the current count of steps in a more subtle way.

1.2 Our results

We study adaptive deterministic protocols for h-shot broadcast (note that the term 'k-shot broadcasting' has been used in some literature for the same notion). We focus on small values of h and provide asymptotically matching lower and upper bounds on the (worst-case) number of steps for the cases $h = 2$ and $h = 3$, as well as improved upper bounds for larger values of h.

In particular, for $h = 2$ we provide a quadratic lower bound of $n^2/8 - O(n)$, showing that adaptive 2-shot broadcast protocols are not (asymptotically) faster than oblivious 2-shot protocols. On the other hand, for $h = 3$ we prove a sub-quadratic bound of $\Theta(n^2 \log\log n / \log n)$. To the best of our knowledge this is the first result showing an adaptive h-shot protocol which is asymptotically faster than oblivious h-shot protocols. For oblivious protocols a tight quadratic bound has been shown in [14] for every constant h. Our proof of existence of a $O(n^2 \log\log n / \log n)$-step 3-shot broadcast protocol is constructive, making use of constructions of graphs with large girth. The *girth of a graph* is the length of a shortest cycle.

Our improved upper bounds for $h \geq 4$ include a bound of $O(n^{1+\alpha/\sqrt{h}})$, where h is constant or grows (slowly) with n and α is an absolute constant independent of h. Our upper bounds for $h \geq 4$ are non-constructive and are based on hyper-graphs without small 2-covers. We give the precise definition of 2-covers in hyper-graphs in Section 4, noting here only that this notion can be viewed as a generalization of the notion of cycles in graphs.

1.3 Related previous work

Radio broadcasting with unlimited number of shots was first introduced by Chlamtac and Kutten [5] and has been extensively studied ever since. The first protocol, given by Bar-Yehuda, Goldreich and Itai [1], was randomized and worked in $O(D \log n + \log^2 n)$ expected time, where D is the diameter of the graph and n the number of nodes. Improved randomized protocols were later proposed in [10, 15] yielding a tight upper bound of $O(D \log(n/D) + \log^2 n)$ steps.

Deterministic radio broadcasting attracted much attention in the last two decades. Brusci and Del Pinto [4] proved a lower bound of $\Omega(D \log n)$ for undirected networks, which was subsequently improved for directed networks to $\Omega(n \log D)$ by Clementi et al. [9] and for undirected networks to $\Omega((n \log n)/\log(n/D))$ by Kowalski and Pelc [15]. The *round-robin* protocol, in which node i is the only node transmitting in steps $i + 1 + qn$, for each $q \geq 1$, gives a trivial $O(n^2)$ upper bound on deterministic broadcast. Chlebus et al. [6] presented the first sub-quadratic protocol of $O(n^{11/6})$ time complexity. The upper bound was then improved to $O(n^{5/3} \log^3 n)$ by De Marco and Pelc [17] and further by Chlebus et al. [7], who showed an $O(n^{3/2})$-time algorithm. Chrobak, Gąsieniec and Rytter [8] gave an $O(n \log^2 n)$ non-constructive protocol and De Marco [11] proved the best currently known upper bound of $O(n \log n \log \log n)$, again in a non-constructive manner.

Better upper bounds are known for undirected networks. Chlebus et al. [6] proposed a deterministic $O(n)$-time broadcasting algorithm, assuming *spontaneous wake-up* (that is, allowing the nodes to transmit before receiving the source message, learning that way the topology of the network). An optimal $O(n \log n)$-time broadcasting algorithm for undirected networks with non-spontaneous wake-up was given by Kowalski and Pelc [15].

Broadcasting with a limited number of shots ("h-shot broadcasting") in known-topology undirected networks was first studied by Gąsieniec et al. [12], who showed a lower bound of $D + \Omega((n - D)^{1/(2h)})$ and a randomized protocol which works in $D + O(hn^{1/(h-2)} \log^2 n)$ steps and has high probability of completing the broadcast. These lower and upper bounds were improved for the same setting (undirected known networks) by Kantor and Peleg [13] to $D + \Omega(h \cdot (n - D)^{1/2h})$ and $D + O(hn^{1/2h}) \log^{2+1/h} n)$, respectively. They also presented the first randomized h-shot broadcasting protocols for *unknown* undirected networks, which work in $O((D + \min\{Dh, \log n\})n^{1/(h-1)} \log n)$ steps for $h \geq 2$ and in $O(Dn^2 \log n)$ steps for $h = 1$. Still in the same setting, Berenbrink et al. [2] proposed, among other results, a randomized algorithm with optimal broadcasting time $O(D \log(n/D) + \log^2 n)$ that uses an expected number of $O(\log^2 n/\log(n/D)$ transmissions per node.

The first work to address deterministic h-shot broadcasting in directed *ad hoc* radio networks is due to Karmakar et al. [14], who proved a lower bound of $\Omega(n^2/h)$ for oblivious protocols and a matching upper bound of $O(n^2/h)$ for each $h \leq \sqrt{n}$, as well as an upper bound of $O(n^{3/2})$ for $h > \sqrt{n}$. They also presented a lower bound of $\Omega\left(n^{1+1/h}\right)$ for adaptive broadcasting protocols, leaving open the question whether there are upper bounds for adaptive h-shot broadcast which are better than the $O(n^2/h)$ bound achieved by oblivious protocols.

2 Lower bounds

2.1 Layered networks

We show lower bounds using the following *layered networks*. We assume $n \geq 4$, and in addition to the source node $s = 0$, we also distinguish the node $d = n - 1$ as the "target" of the broadcast. Node d will be the last node of a layered network to receive the message. We derive lower bounds on the number of steps needed by a broadcast protocol to deliver the message from node s to node d in the worst case.

Consider a partition $L_0, L_1, \ldots, L_k$ of the set of nodes V into $k \geq 2$ sets called *layers*, such that $L_0 = \{0\}$, $L_k = \{n-1\}$ and $L_i \neq \emptyset$ for $0 \leq i \leq k$. These layers define the following acyclic broadcast network $G \equiv G(L_0, L_1, \ldots L_k)$. For each $0 \leq i \leq k-1$, the consecutive layers L_i and L_{i+1} are fully connected, that is, there is a directed edge from each node of L_i to each node of L_{i+1}, and there are no any other edges.

For any pairwise disjoint non-empty subsets $L_0, L_1, \ldots, L_j$ of $V \setminus \{n-1\}$, where $j \geq 1$ and $L_0 = \{0\}$, we denote by $\mathcal{G}_j \equiv \mathcal{G}_j(L_0, L_1, \ldots L_j)$ the family of all layered networks $G(L_0, L_1, \ldots L_j, L_{j+1} \ldots L_k)$, where $k > j$, $L_k = \{n-1\}$ and $L_{j+1} \ldots L_{k-1}$ are non-empty sets partitioning $V \setminus \left(\{n-1\} \cup \bigcup_{i=0}^{j} L_i\right)$. In other words, $\mathcal{G}_j$ is the family of all layered networks which have the same fixed initial layers $L_0, L_1, \ldots, L_j$. In particular, $\mathcal{G}_0$ is the family of all layered networks.

We use layered networks in order to show that for any given protocol, there is an assignment of nodes to layers which makes the progress of broadcast slow because of relatively long delays at each layer.

2.2 Conditional transmission sets

Let Π be any h-shot broadcast protocol for n-node networks and let $T_{\max}$ denote the maximum broadcast time of Π over all n-node networks. We define below *conditional transmission sets* for families of layered graphs described above.

Let $i \geq 1$ and consider the family of networks $\mathcal{G}_{i-1}(L_0, L_1, \ldots, L_{i-1})$ for some arbitrary layer sets $L_0 = \{0\}, L_1, L_2, \ldots, L_{i-1}$, such that $|\bigcup_{j=0}^{i-1} L_j| \leq n-3$. This bound implies that the target node and at least two other nodes are still outside of the fixed layers. Protocol Π behaves in exactly the same way on any network $G \in \mathcal{G}_{i-1}(L_0, L_1, \ldots, L_{i-1})$ until (and including) the step T_{i-1} when the source message leaves layer $i-1$ for the first time. That is, T_{i-1} is the first step when a unique node in L_{i-1} transmits, sending the message simultaneously to all nodes in the next layer. Note that $T_0 = 1$ and step T_{i-1} is uniquely determined by the sets $L_0, L_1, \ldots, L_{i-1}$. We select the next layer L_i from the set

$$U_i = V \setminus \left(\{n-1\} \cup \bigcup_{j=0}^{i-1} L_j \right),$$

trying to maximize the *weighted delay* $(T_i - T_{i-1})/|L_i|$ at this layer.

For $t \geq 1$, the *conditional transmission set* $S_t \subseteq U_i$ contains a node $v \in U_i$, if and only if, node v transmits at step $T_{i-1} + t$, if v is included in the layer L_i. Set S_t is well defined since for each network $G \in \mathcal{G}_{i-1}(L_0, L_1, \ldots, L_{i-1})$ with $v \in L_i$, node v transmits in exactly the same steps, irrespectively of how the other nodes in $U_i \setminus \{v\}$ are distributed among the layers L_j, $j \geq i$. This follows from the fact that a node in one layer gets information, directly or indirectly, only from nodes in previous layers.

Since we consider h-shot protocols, each node $v \in U_i$ belongs to at most h conditional transmission sets S_t. We assume w.l.o.g. that v transmits in exactly h steps $T_{i-1}+t$, so it belongs to exactly h conditional transmission sets. (If v belongs to $k < h$ conditional sets, then add v to $h-k$ sets S_τ for $\tau = T_{\max} - T_{i-1} + 1, \ldots, T_{\max} - T_{i-1} + h - k$. This may create new transmission collisions, but only after step $T_{\max}$, that is, after the completion of broadcast.) For convenience, if it is clear from the context that we are discussing the selection of nodes for the layer L_i, then we will refer to the (global) transmission step $T_{i-1}+t$ as simply the transmission step t (the t-th step after step T_{i-1}). Also, "conditional transmission sets" will be abbreviated to "transmission sets".

At least one of the transmission sets S_t must be a singleton, or otherwise the message would never reach the target node in the network $G(L_0, L_1, \ldots, L_{i-1}, U_i, \{n-1\})$, that is, when layer i contains all remaining nodes (other than the target node $n-1$). Let $\tau_1 \geq 1$ be the smallest index of a singleton transmission set. Let $S_{\tau_1} = \{v_1\}$ and we also use $t_0(v_1)$ and $S_0(v_1)$ to denote τ_1 and S_{τ_1}, respectively.

Applying the same argument to sets $S'_t = S_t \setminus \{v_1\}$, we observe that there must be a singleton also among these sets. Indeed, if each non-empty set S'_t, $t \geq 1$, had size at least 2, then the message would never reach the target node in the network $G(L_0, L_1, \ldots, L_{i-1}, U_i \setminus \{v_1\}, \{v_1\}, \{n-1\})$. Let $S'_{\tau_2} = \{v_2\}$ be the first singleton among sets S'_t, and let $t_0(v_2)$ and $S_0(v_2)$ denote τ_2 and S_{τ_2}, respectively. We note that $S_0(v_2)$ is equal to either $\{v_2\}$ or $\{v_1, v_2\}$ and step $t_0(v_2)$ can be before or after step $t_0(v_1)$.

Continuing this way, we put all nodes of U_i in a sequence $v_1, v_2, \ldots, v_u$, where $u = |U_i| \geq 2$, and associate with them distinct transmission steps $t_0(v_j)$ and transmission sets $S(v_j)$ such that:

$$S_0(v_1) = \{v_1\} \text{ and } S_0(v_j) \setminus \{v_1, v_2, \ldots, v_{j-1}\} = \{v_j\}, \text{ for } 2 \leq j \leq u.$$

Note that for each $1 \leq j \leq u$, we have $\bigcup_{l=1}^{j} S_0(v_l) = \{v_1, v_2, \ldots, v_j\}$.

By construction, for any two distinct nodes v and w in U_i, the steps $t_0(v)$ and $t_0(w)$ are also distinct. Thus for at least $\lceil u/2 \rceil$ nodes in U_i, we have $t_0(v) > \lfloor u/2 \rfloor$. We denote the set of these nodes by U'_i, that is,

$$U'_i = \{v \in U_i : t(v) > \lfloor u/2 \rfloor\},$$

and let $u' = |U'_i| \geq \lceil u/2 \rceil$.

For each node $v \in U'_i$, we have designated one of the v's (conditional) transmission steps as the step $t_0(v) \geq |U_i|/2$ and we have denoted the corresponding transmission set by $S_0(v)$. We now further denote by $t_1(v) < t_2(v) < \cdots < t_{h-1}(v)$ and by $S_1(v), S_2(v), \ldots, S_{h-1}(v)$ the other $h-1$ steps when node v transmits (if in L_i) and the transmission sets at those steps. While for any two distinct nodes v' and v'' in U'_i, $t_0(v') \neq t_0(v'')$, we may have $t_q(v') = t_r(v'')$ for some $1 \leq q, r \leq h-1$.

The general idea for forcing a large weighted delay at layer i is to try to select for this layer a relatively small number of nodes $x_1, x_2, \ldots, x_k$ from U'_i which have transmission conflicts at all transmission steps $t_l(x_j)$, for $1 \leq j \leq k$ and $1 \leq l \leq h-1$. That is, for each $1 \leq j \leq k$ and $1 \leq l \leq h-1$, there is $1 \leq a \leq k$, $a \neq j$ such that $\{x_j, x_a\} \subseteq S_l(x_j)$. If we manage to select such a layer, then the progress of broadcast from this layer will have to rely on one of the steps $t_0(x_1), t_0(x_2), \ldots, t_0(x_k)$, so the weighted delay will be at least $\min\{t_0(x_1), t_0(x_2), \ldots, t_0(x_k)\}/k \geq |U_i|/(2k)$. This will be the basic case in our lower-bound analysis.

2.3 Lower bound for 2-shot broadcast

We consider now the case when each node has only two transmissions, with a node $v \in U'_i$ transmitting in steps $t_0(v)$ and $t_1(v)$. Recall that $u' = |U'_i| \geq \lceil u/2 \rceil$ and consider two cases: either there is a node $v \in U'_i$ with $t_1(v) \geq \lceil u/2 \rceil$ or, by the pigeonhole principle, there are two distinct nodes v and w in U'_i such that $t_1(v) = t_1(w) < \lceil u/2 \rceil$. We set $L_i = \{v\}$ in the former case, to get $T_i = T_{i-1} + \min\{t_0(v), t_1(v)\} \geq T_{i-1} + \lceil u/2 \rceil$, and $L_i = \{v, w\}$ in the latter case, to get $T_i = T_{i-1} + \min\{t_0(v), t_0(w)\} \geq T_{i-1} + \lceil u/2 \rceil$. Thus we can force the delay at layer i of at least $\lceil u/2 \rceil$ by putting one or two nodes into this layer, so we have the following lemma.

▶ **Lemma 1.** *For each 2-shot broadcast protocol Π for n-node networks and a family of networks $\mathcal{G}(L_0, L_1, \ldots, L_{i-1})$, where $|\bigcup_{j=0}^{i-1} L_j| \leq n-3$, there exists $L_i \subseteq U_i = V \setminus \left(\{n-1\} \cup \bigcup_{j=0}^{i-1} L_j\right)$ such that $1 \leq |L_i| \leq 2$ and for each network in $\mathcal{G}(L_0, L_1, \ldots, L_{i-1}, L_i)$, the message does not leave layer i (that is, is not delivered to the next layer $i+1$) before the step $T_{i-1} + \lceil |U_i|/2 \rceil$.*

Using this lemma iteratively, we prove the following lower bound on the worst-case time of 2-shot broadcast protocols.

▶ **Theorem 2.** *For each 2-shot broadcast protocol Π for n-node networks, there exists a network in $\mathcal{G}_0$ on which Π needs at least $n^2/8 - O(n)$ steps to complete broadcast.*

Proof. Starting from $L_0 = \{0\}$, we apply Lemma 1 iteratively for $i = 1, 2, \ldots$ to obtained a network $G(L_0 = \{0\}, L_1, L_2, \ldots, L_{k-1}, L_k = \{n-1\})$ such that $k \geq n/2$, $1 \leq |L_i| \leq 2$, for each $1 \leq i \leq k-1$, and $T_i \geq T_{i-1} + |U_i|/2$. We have $|U_1| = n-2$ and $|U_i| \geq |U_{i-1}| - 2 \geq n - 2i$, for $2 \leq i \leq k-1$, so the worst-case number of steps needed by protocol Π to complete the broadcast is at least

$$1 + \sum_{1 \leq i \leq k-1} |U_i|/2 \;\geq\; \sum_{1 \leq i \leq (n/2)-1} (n-2i)/2 \;=\; n^2/8 - O(n). \qquad \blacktriangleleft$$

2.4 Lower bound for 3-shot broadcast

We consider now a 3-shot broadcast protocol Π and, as before, the family of networks $\mathcal{G}_{i-1}(L_0, L_1, \ldots, L_{i-1})$ for some arbitrary layer sets $L_0 = \{0\}, L_1, L_2, \ldots, L_{i-1}$. The message leaves layer $i-1$ at time step T_{i-1} and we want to select nodes for the next layer i to force a relatively large weigthed delay $(T_i - T_{i-1})/|L_i|$. We refer to the notation of (conditional) transmission sets and the related terminology introduced in Sections 2.1 and 2.2. A node v in the set $U' = U'_i$ transmits in the step $t_0(v) \geq u/2$ and in steps $t_1(v) < t_2(v)$, where $u = n - 1 - \left|\bigcup_{j=0}^{i-1} L_j\right|$ and $|U'| \geq u/2$.

For an integer parameter $1 \leq p \leq u/2$, which will be set later, we put each node $v \in U'$ into one of the sets V_0, V_1 and V_2, depending on when the v's transmission steps $t_1(v)$ and $t_2(v)$ are in relation to step p:

$$\begin{aligned} V_0 &= \{v \in U' :\ p < t_1(v) < t_2(v)\}, \\ V_1 &= \{v \in U' :\ t_1(v) \leq p \text{ and } t_2(v) > p\}, \\ V_2 &= \{v \in U' :\ t_1(v) < t_2(v) \leq p\}. \end{aligned}$$

For the set V_2, we construct an undirected (multi-)graph H_2 with vertices $t_l(v)$, where $v \in V_2$ and $l = 1, 2$, and edges $\{t_1(v), t_2(v)\}$ for all $v \in V_2$. More precisely, the vertex set and the

edge (multi-)set of graph H_2 are

$$V(H_2) = \{t : t = t_l(v) \text{ for some } v \in V_2 \text{ and } 1 \le l \le 2\},$$
$$E(H_2) = \{\{t', t''\} : t' = t_1(v) \text{ and } t'' = t_2(v) \text{ for some } v \in V_2\}.$$

Thus graph H_2 has at most p vertices and exactly $|V_2| \le u'$ edges. There may be parallel edges in H_2 because there may be two nodes v', v'' in V_2 with $\{t_1(v'), t_2(v')\} = \{t_1(v''), t_2(v'')\}$. To avoid confusion, *nodes* are in the transmission *network*, while *vertices* are in the auxiliary *graph* H_2 (and in other similar auxiliary graphs constructed later). The vertices of graph H_2 correspond to (some) steps of the protocol and the edges of H_2 correspond to (some) nodes in the transmission network.

The underlying idea in our lower-bound argument is that if the number of edges in graph H_2 is relatively large in relation to p, that is, if H_2 is sufficiently dense, then it must contain a short cycle $\Gamma = (\tau_0, \tau_1, \ldots, \tau_{k-1}, \tau_k = \tau_0)$. Let $v_i \in V_2$ be the node in the transmission network which corresponds to the edge $\{\tau_i, \tau_{i+1}\}$ in this cycle, for $i = 0, 1, \ldots, k-1$. (Two parallel edges in H_2 would form a cycle of length $k = 2$.) If we set the next layer $L_i = \{v_0, v_1, \ldots, v_{k-1}\}$, then these nodes transmit in steps $\tau_0, \tau_1, \ldots, \tau_{k-1}$, but in each of these steps exactly two nodes in L_i transmit, resulting in a collision. This means that the progress of broadcast has to rely on one of the steps $t_0(v_0), t_0(v_1), \ldots, t_0(v_{k-1})$, but each of these steps is at least $u/2$, so the weighted delay at layer i is at least $u/(2k)$. Therefore we have the following lemma.

▶ **Lemma 3.** *If graph H_2 has a cycle Γ of length k, then taking for the layer L_i the set of transmission nodes which correspond to the edges of Γ gives the weighted delay at layer i at least $u/(2k)$.*

To proceed with our analysis, we need an upper bound on the *girth of a graph*, that is, on the length of a shortest cycle. The asymptotic bounds given below in Lemma 4 and in its corollary are widely known and sufficient for us, but we note that more precise bounds are available in the literature, for example, in [3].

▶ **Lemma 4.** *Every graph with p vertices and the minimum degree $d = d(p) \ge 3$ contains a cycle of length $O(\log p / \log d)$.*

Proof. Consider any graph H with p vertices and the minimum degree $d \ge 3$. Let v be any vertex in H, $k \ge 1$ and $H(v, k)$ the subgraph of G induced by the vertices within distance at most k from v. If H does not have a cycle of length $2k$ or less, then $H(v, k)$ is a tree and has more than $(d-1)^k$ vertices. This means that for $k = \lceil \log n / \log(d-1) \rceil$, $H(v, k)$ is not a tree and contains a cycle of length at most $2k = O(\log n / \log d)$. ◀

▶ **Corollary 5.** *Every graph of average degree d with p vertices contains a cycle of length $O(\log p / \log d)$.*

Proof. Any graph G of average degree d must contain a nonempty subgraph G' of minimum degree at least $d/2$. To see this, repeatedly remove from G all vertices of degree strictly less than $d/2$. Not all vertices can be removed in this process because otherwise G would contain fewer than $pd/2$ edges altogether, a contradiction. By applying Lemma 4 to G' the claim follows. ◀

▶ **Lemma 6.** *Let Π be any 3-shot broadcast protocol Π for n-node networks and consider any family of networks $\mathcal{G}(L_0, L_1, \ldots, L_{i-1})$, where $|\bigcup_{j=0}^{i-1} L_j| \le n/2$. There exists the next i-th layer $L_i \subseteq U_i = V \setminus \left(\bigcup_{j=0}^{i-1} L_j \cup \{n-1\}\right)$ such that for each network in $\mathcal{G}(L_0, L_1, \ldots, L_{i-1}, L_i)$, the weighted delay at layer i is $\Omega(n \log\log n / \log n)$.*

Proof. We set $p = n \log\log n / \log n$ and consider sets V_0, V_1 and V_2. If V_0 is not empty, then we take $L_i = \{v\}$ where v is an arbitrary node in V_0. All three steps $t_0(v), t_1(v)$ and $t_2(v)$ when node v transmits are at least p, so the weighted delay at layer i is at least p.

If $|V_1| > p$, then there must be two nodes v' and v'' in V_1 such that $t_1(v') = t_1(v'') \leq p$, but all other steps $t_0(v'), t_2(v'), t_0(v'')$ and $t_2(v'')$ when v' or v'' transmits are at least p. Taking $L_i = \{v', v''\}$ gives the weighted delay at layer i at least $p/2$.

If V_0 is empty and V_1 has fewer than p nodes, then V_2 has more than $u' - p$ nodes, so graph H_2 has $|V_2| > u' - p \geq |U_i|/2 - p \geq n/5$ edges but at most p vertices. This means that the average degree in H_2 is greater than $(2/5)n/p$, so, from Corollary 5, H_2 has a cycle Γ of length $O(\log p / \log(n/p)) = O(\log n / \log\log n)$. Thus Lemma 3 implies that taking for the layer L_i the set of transmission nodes which correspond to the edges of Γ gives the weighted delay at layer i at least $\Omega(n \log\log n / \log n)$. ◀

We are now ready to prove the lower bound for the 3-shot case.

▶ **Theorem 7.** *For each 3-shot broadcast protocol Π for n-node networks, there exists a network in $\mathcal{G}_0$ on which Π needs $\Omega(n^2 \log\log n / \log n)$ steps to complete broadcast.*

Proof. Starting from $L_0 = \{0\}$, we use Lemma 6 iteratively, obtaining layers $L_1, L_2, \ldots, L_m$ and stopping when $\bigcup_{0 \leq i \leq m} |L_i| > n/2$. From Lemma 6, there is a constant $c > 0$ such that for each layer $i = 1, 2, \ldots, m$, the weighted delay $(T_i - T_{i-1})/|L_i|$ is at least $cn \log\log n / \log n$. Therefore,

$$T_{\max} \geq T_m = 1 + \sum_{1 \leq i \leq m} (T_i - T_{i-1}) \geq \sum_{1 \leq i \leq m} (|L_i| cn \log\log n / \log n)$$
$$\geq (c/2) n^2 \log\log n / \log n.$$ ◀

3 Upper bounds for h-shot broadcast for $h \leq 3$

For the 2-shot case a trivial upper bound which matches asymptotically the $\Omega(n^2)$ lower bound of Section 2.3 is given by the oblivious Round Robin (which is actually a 1-shot broadcast protocol).

We provide in this section an upper bound of $O(n^2 \log\log n / \log n)$ for 3-shot broadcast, which matches our lower bound and shows that in contrast to the 2-shot case, the fastest adaptive 3-shot protocols are faster than the best oblivious protocols by a factor $\omega(1)$.[3] We base our approach on graph-theoretic results [16] showing that it is possible to construct relatively dense graphs of high girth. We use such graphs to specify appropriate transmission sets as detailed below.

To define the sequence of transmission sets in our protocol, we use a graph $H = H(n, p, g)$ with n edges, p vertices and girth g. Any graph $H(n, p, g)$ would do for the correctness of our protocol, but to achieve fast (worst case) broadcast, we need a graph with relatively small number of nodes p and high girth g. More precisely, to have asymptotically fastest broadcast, we need a graph $H(n, p, g)$ with $p = \Theta(n \log\log n / \log n)$ and $g = \Theta(\log n / \log\log n)$.

We identify the edge set $E(H)$ of graph H with the node set $V(G)$ of the transmission network G, and we number the vertices in H from 1 to p (in an arbitrary way). Let H_i denote the set of edges in H which are incident to vertex i. The sets $H_1, H_2, \ldots, H_p$ are (some of) the transmission sets of our protocol. Clearly, for any node v of G, v belongs to two

[3] Recall that the oblivious bound is $\Theta(n^2/k)$ for k-shot protocols and $k \leq \sqrt{n}$.

sets H_i and H_j, where $\{i,j\} \in E(H)$ is the edge identified with node v. Node v transmits in two steps with transmissions sets H_i and H_j, while the third transmission is within one Round-Robin sequence.

Formally, our protocol $\Pi(H)$ is defined by the repeated Round-Robin sequence $\langle R \rangle = (\{0\}, \{1\}, \ldots, \{n-1\})$ interleaved with the repeated sequence $\langle H \rangle = (H_1, H_2, \ldots, H_p)$. Let's say that we use the odd steps of the protocol for repeating the Round-Robin sequence and the even steps for repeating the sequence $\langle H \rangle$. If a node v receives the message in step t, then it transmits in its step of the first Round-Robin sequence which starts after step t, and in the steps H_i and H_j of the first copy of the sequence $\langle H \rangle$ which starts after step t, where $\{i,j\} \in E(H)$ is the edge identified with node v.

We now proceed with the analysis of protocol $\Pi(H)$. Consider any n-node transmission network G with source s and an arbitrary node $v \neq s$. Let $k \geq 1$ denote the distance from s to v. In order to upper-bound the time needed for the message to go from source node s to node v, we consider the partitioning $L_v(G)$ of the nodes within distance k to v into layers. These are breadth-first-search layers constructed from node v following the edges of G in reverse direction. For $0 \leq i \leq k$, the layer L_i is the set of all nodes in G with distance $k-i$ to v. Thus $L_k = \{v\}$, L_{k-1} is the set of all nodes with edges to v, and so on. The source node s belongs to layer L_0.

Note that for each $1 \leq i \leq k$, each node $u \in L_i$ and each edge (x,u), $x \in L_j$ for some $j \geq i-1$. Thus the message reaches layer L_i (any node in layer L_i) for the first time during a transmission by a node from layer L_{i-1}. We use T_i to denote the time step at which the message first reaches layer L_i. We have $T_1 = 1$ (layer L_1 must have at least one out-neighbour of the source) and the following lemma gives an upper bound on the delays at layers of relatively small cardinality.

▶ **Lemma 8.** *Consider an n-node transmission network G with source s, an arbitrary node v, the layers $L_0, L_1, \ldots, L_k$ corresponding to this node and the protocol $\Pi(H)$ defined by a graph $H = H(n,p,g)$. During the execution of this protocol, if $|L_i| < g$, then the time needed to transmit the message from L_i to L_{i+1}, that is, $T_{i+1} - T_i$, is at most $4p$.*

Proof. Let $L'_i \subseteq L_i$ be the set of nodes in L_i that have received the message by time $T_i + t$, where t is the smallest integer such that $(T_i + t) \bmod (2p) = 0$. Only nodes in L'_i will be transmitting at even steps between $T_i + t + 1$ and $T_i + t + 2p$.

Since $|L'_i| \leq |L_i| < g$, the edges corresponding to nodes of L'_i form an acyclic subgraph T of H, so for each vertex w_j in H with degree 1 in T (there must be at least two such vertices) the transmission set H_j contains exactly one node from L'_i. During each such step, the message is transmitted from layer L_i to layer L_{i+1}. Hence $T_{i+1} \leq T_i + t + 2j \leq T_i + 4p$. ◀

▶ **Theorem 9.** *Protocol $\Pi(H)$ defined by a graph $H = H(n,p,g)$ completes broadcast in an arbitrary n-node transmission network G within $O(n^2/g + np)$ steps.*

Proof. We take an arbitrary node $v \neq s$ and consider its layers $L_0, L_1, \ldots, L_k$. There can be at most n/g layers of size at least g. For each such layer L_i, when a message arrives at this layer, then it will reach the next layer L_{i+1} by the time the next full Round Robin is completed. That is, in this case $T_{i+1} \leq T_i + 4n$. Combining this with Lemma 8 gives the claimed bound on the number of steps, since the number of layers of size smaller than g is at most $n-1$. ◀

To minimize the upper bound $O(n^2/g + np) = O(n^2/\min\{g, d_{ave}\})$, where d_{ave} is the average degree in graph $H(n,p,g)$, we have to find a graph with n edges and $\min\{g, d_{ave}\}$ as large as possible. Corollary 5 implies that for all graphs, $\min\{g, d_{ave}\} = O(\min\{\log n / \log d_{ave},$

$d_{ave}\}) = O(\log n/\log\log n)$. It turns out that there are explicitly constructed graphs with n edges for which $\min\{g, d_{ave}\} = \Theta(\log n/\log\log n)$. We use the construction given in [16].

▶ **Theorem 10** ([16]). *For each positive odd integer $k \geq 3$ and a power of a prime q, there is an explicit construction of a q-regular bipartite graph $H(q,k)$ with $2q^k$ vertices and girth at least $k+5$.*

▶ **Corollary 11.** *There exists an explicit construction of a graph H with n edges, $p = \Theta(n\log\log n/\log n)$ vertices and girth $g = \Theta(\log n/\log\log n)$.*

Proof. For a given sufficiently large n, let $q \geq 4$ be the largest power of 2 not greater than $\log n/\log\log n$ and let $k = q-1 \geq 3$. Let $H(q,k)$ be the graph from Theorem 10. This graph has $2q^{q-1}$ vertices, $q^q \leq n$ edges and girth at least $q+4$.

Let H be a graph with exactly n edges obtained by taking copies of graph $H(q,k)$ as connected components. We remove (arbitrarily) some edges from the last copy of $H(q,k)$ so that the total number of edges is exactly n. We need $\lceil n/q^q \rceil$ copies of $H(q,k)$, so the number of vertices in graph H is at most $2q^{q-1}(n/q^q+1) \leq 4n/q \leq 8n\log\log n/\log n$. Graph H has the same girth as $H(q,k)$, so at least $q+4 \geq (1/2)\log n/\log\log n$. ◀

Using in protocol $\Pi(H)$ the graph H from Corollary 11, Theorem 9 gives us the following result.

▶ **Corollary 12.** *There exists a constructive 3-shot broadcast protocol which completes broadcast on any graph G with n nodes in time $O(n^2\log\log n/\log n)$.*

4 Upper bounds for h-shot broadcast for $h \geq 4$

It was shown in [14] that for any $h \geq 1$, an h-shot broadcast protocol requires $\Omega\left(n^{1+1/h}\right)$ steps. In previous sections, we improved this lower bound and provided matching upper bounds for the cases when h is equal to 2 and 3. In this section, we show upper bounds for $h \geq 4$. In particular, if h is a sufficiently large constant or is slowly growing with n, then we prove that there exist h-shot broadcast protocols with $O(n^{1+\alpha/\sqrt{h}})$ steps, where α is an absolute constant independent of h.

The general idea for h-shot broadcast protocols for $h \geq 4$ is similar to the idea of using a large girth graph to construct a 3-shot protocol. Now, however, we need to define $r = h-1 \geq 3$ transmission slots for each node (in addition to the transmissions defined by Round-Robin), so we use r-uniform hyper-graphs instead of graphs $H(n,p,g)$. Let $H_r = H_r(n,p,k)$ be an r-uniform hyper-graph (each edge is a set of r vertices) with n (hyper-)edges, p vertices, and no 2-cover of size k or smaller. A 2-*cover* of a hyper-graph is a non-empty subset $\mathcal{A}$ of edges such that each node which belongs to an edge in $\mathcal{A}$ belongs to at least two edges in $\mathcal{A}$. The notion of 2-covers in hyper-graphs generalizes the notion of cycles in graphs: minimal 2-covers in graphs are (simple) cycles.

Similarly as in the previous subsection, we identify the edge set $E(H_r)$ of the hyper-graph H_r with the node set $V(G)$ of the transmission network G. We number the vertices in H_r from 1 to p in an arbitrary order and denote by $H_r^{(i)}$ the set of edges in H_r which are incident to vertex i. If we use the sequence $\langle H_r\rangle = \langle H_r^{(1)}, H_r^{(2)}, \ldots, H_r^{(p)}\rangle$ as a sequence of transmission sets, then for each nonempty subset W of at most k nodes in G, one of these transmission sets has exactly one node from W – otherwise the set of edges in H_r corresponding to the nodes in W would form a 2-cover in H_r of size at most k.

The following simple counting argument shows how large k can be in an $H_r(n,p,k)$ hyper-graph.

▶ **Lemma 13.** *There is a constant C such that for each $n \geq 1$ and for each $p \geq r \geq 3$, there exists a hyper-graph $H_r = H_r(n, p, k)$ with $k = \lfloor p/(Crn^{2/r}) \rfloor$.*

Proof. We consider a random r-uniform hyper-graph $\mathcal{H}$ with p vertices and n edges (independently and uniformly selected from the family of sets of r vertices) and show that for k defined in the lemma (where constant C will come out from the calculations) and for a fixed $2 \leq q \leq k$, the probability that $\mathcal{H}$ has a 2-cover of size q is at most $1/2^q$. By summing up over all $2 \leq q \leq k$, we get the conclusion that there must exist a hyper-graph $H_r = H_r(n, p, k)$.

A 2-cover $\mathcal{A}$ of size q covers at most $qr/2$ vertices, or otherwise there would be a vertex belonging to exactly one edge in $\mathcal{A}$. Thus the probability that $\mathcal{H}$ has a 2-cover of size q is at most the probability that there exists in $\mathcal{H}$ a set $\mathcal{A}$ of q edges and a set X of $qr/2$ vertices such that each edge in $\mathcal{A}$ is a subset of X. Using the union bound over all possible $\mathcal{A}$ and X, the probability of the latter event is at most

$$\binom{n}{q}\binom{p}{qr/2}\frac{\binom{qr/2}{r}^q}{\binom{p}{r}^q} \leq \left(\frac{en}{q}\right)^q \left(\frac{2ep}{qr}\right)^{qr/2} \frac{(eqr/2)^{qr}/r^{qr}}{p^{qr}/r^{qr}} \leq \frac{1}{q^q}\left(\frac{Cqrn^{2/r}}{p}\right)^{qr/2} \leq \frac{1}{2^q},$$

where the second inequality holds for $C = (2e)^2$ and the last one holds for any $2 \leq q \leq p/(Crn^{2/r})$. For the first inequality, we use $\left(\frac{a}{b}\right)^b \leq \binom{a}{b} \leq \left(\frac{ea}{b}\right)^b$. ◀

For a hyper-graph $H_r = H_r(n, p, k)$, the protocol $\Pi(H_r)$ which interleaves repeated copies of $\langle H_r \rangle$ with copies of a Round-Robin sequence $\langle R \rangle$ is an h-shot broadcast protocol with $O(n^2/k + np)$ steps. This can be shown in an analogous way as in the proof of Theorem 9, by considering separately the layers with sizes at most k and the layers with sizes greater than k. If we consider hyper-graphs $H_r = H_r(n, p, k)$ with $k = \lfloor p/(Crn^{2/r}) \rfloor$, whose existence is guaranteed by Lemma 13, and take $p = r^{1/2}n^{1/2+1/r}$ to minimize $O(n^2/k + np)$, then we obtain an h-shot broadcast protocol with $O(h^{1/2}n^{3/2+1/(h-1)})$ steps. This gives, for example, upper bounds $O(n^{11/6})$ and $O(n^{7/4})$ for 4-shot and 5-shot broadcast, respectively, but no better bound than $O(n^{3/2})$ even if h grows to infinity.

To obtain upper bounds with the exponent at n decreasing to 1 for increasing values of h, we combine hyper-graphs $H_r(n, p, k)$ for a number of different values of k. More specifically, for $h = \rho^2/2+1$, where ρ is an even integer at least 4, let $H_{\rho,j} = H_\rho(n, C\rho n^{2j/\rho}, n^{2(j-1)/\rho})$, for $j = 1, 2, \ldots, J = \rho/2$, where C is the constant from Lemma 13. Our h-shot broadcast protocol Π_h is defined by the sequence of transmission sets obtained by interleaving $\rho + 1$ sequences $(\langle H_{\rho,1} \rangle, \langle H_{\rho,1} \rangle, \ldots)$, $(\langle H_{\rho,2} \rangle, \langle H_{\rho,2} \rangle, \ldots), \ldots, (\langle H_{\rho,J} \rangle, \langle H_{\rho,J} \rangle, \ldots)$, and $(\langle R \rangle, \langle R \rangle, \ldots)$, and by the following transmission schedule. For a node v in the transmission network G, if v receives the message for the first time in step t, then let $\overline{\langle H_{\rho,j} \rangle}$, for $j = 1, 2, \ldots, J$, and $\overline{\langle R \rangle}$ be, respectively, the first copies of $\langle H_{\rho,1} \rangle, \langle H_{\rho,2} \rangle, \ldots, \langle H_{\rho,J} \rangle$ and $\langle \mathcal{R} \rangle$ which start after step t. Node v transmits in the steps corresponding to the transmission sets in $\overline{\langle H_{\rho,1} \rangle}, \overline{\langle H_{\rho,2} \rangle}, \ldots, \overline{\langle H_{\rho,J} \rangle}$ and $\overline{\langle R \rangle}$ which include v. Thus v transmits in $\rho \cdot (\rho/2) + 1 = h$ steps.

▶ **Theorem 14.** *For $h = \rho^2/2+1$, where ρ is an even integer at least 4, the (non-constructive) protocol Π_h is an h-shot broadcast protocol with $O(hn^{1+\sqrt{8/(h-1)}})$ steps.*

Proof. By the definition of protocol Π, no node transmits more then h times. We show now the claimed bound on the number of steps.

Similarly to the analysis of the 3-shot protocol in Section 3, we consider an arbitrary node v and its in-neighbourhood layers $L_0, L_1, \ldots, L_k$, where $s \in L_0$ and $v \in L_k$. The delay

at layer L_i, that is, the number of steps between the time when the first node in L_i receives the message and the time when the first node in L_{i+1} receives the message (from one of the nodes in L_i), depends on the size of this layer. If $n^{2(j-2)/\rho} < |L_i| \leq n^{2(j-1)/\rho}$, for some $1 \leq j \leq J$, then the message is delivered from (one of the nodes of) layer L_i to (one of the nodes of) the next layer by the next copy of $\langle H_{\rho,j} \rangle$, so within $C\rho^2 n^{2j/\rho}$ steps. (Recall that the transmission sets of each $\langle H_{\rho,i} \rangle$ are scheduled every $\rho/2+1$ steps, hence the additional factor of ρ).

For a layer L_i such that $n^{2(J-1)/\rho} < |L_i|$, the message is delivered to the next layer by the next copy of Round-Robin, so within ρn steps. Thus the delay at each layer L_i is at most $C\rho^2 n^{4/\rho}|L_i|$ steps, so node v receives the message within $O(\rho^2 n^{1+4/\rho}) = O(hn^{1+\sqrt{8/(h-1)}})$ steps. ◀

We defined protocols Π_h only for values $h = \rho^2/2+1$, where ρ is an even integer at least 4. Since the h-shot broadcast protocol Π_h is also an h'-shot broadcast protocol for any $h' \geq h$, then Theorem 14 implies the following corollary.

▶ **Corollary 15.** *There is a constant α such that for any $1 \leq h = O(\log n)$, there exists an h-shot broadcast protocol with $O(\min\{n^2,\ n^{1+\alpha/\sqrt{h}}\})$ steps.*

Proof. It is enough to consider $h \geq 9$, since the case when $h < 9$ can be covered by taking sufficiently large α. For $h \geq 9$, take $\rho = \lfloor \sqrt{2(h-1)} \rfloor$, $\tilde{h} = \rho^2/2+1 \leq h$ and the protocol $\Pi_{\tilde{h}}$, which is an h-shot broadcast protocol. Theorem 14 implies that protocol $\Pi_{\tilde{h}}$ works in $O(\rho^2 n^{1+4/\rho})$ steps, which is $O(n^{1+5/\sqrt{h}})$ for $9 \leq h = O(\log n)$. ◀

For the cases $h = 2$ and $h = 3$, we have obtained asymptotically matching lower and upper bounds on the number of steps in h-shot broadcast protocols. For $h \geq 4$, however, we still have a gap between the lower bound of $\Omega\left(n^{1+1/h}\right)$ shown by Karmakar et al. [14] and our upper bounds.

References

1 Reuven Bar-Yehuda, Oded Goldreich, and Alon Itai. On the time-complexity of broadcast in radio networks: An exponential gap between determinism and randomization. In *PODC '87*, pages 98–108, 1987.

2 Petra Berenbrink, Colin Cooper, and Zengjian Hu. Energy efficient randomised communication in unknown adhoc networks. *Theoretical Computer Science*, 410(27-29):2549–2561, 2009. doi:10.1016/j.tcs.2009.02.002.

3 Norman Biggs. *Algebraic Graph Theory*. Cambridge University Press, Cambridge, 2nd edition, 1993.

4 Danilo Brusci and Massimiliano Del Pinto. Lower bounds for the broadcast problem in mobile radio networks. *Distributed Computing*, 10(3):129–135, 1997.

5 Imrich Chlamtac and Shay Kutten. On broadcasting in radio networks-problem analysis and protocol design. *IEEE Transactions on Communications 33*, pages 1240–1246, 1985.

6 Bogdan S. Chlebus, Leszek Gąsieniec, Alan Gibbons, Andrzej Pelc, and Wojciech Rytter. Deterministic broadcasting in unknown radio networks. In *Proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*, pages 861–870. ACM/SIAM, 2000.

7 Bogdan S. Chlebus, Leszek Gąsieniec, Anna Östlin, and John Michael Robson. Deterministic radio broadcasting. In *Proc. 27th International Colloquium on Automata, Languages and Programming (ICALP'00)*, volume 1853 of *Lecture Notes in Computer Science*, pages 717–728. Springer Verlag, 2000.

8 Marek Chrobak, Leszek Gąsieniec, and Wojciech Rytter. Fast broadcasting and gossiping in radio networks. *Journal of Algorithms*, 43(2):177–189, 2002.

9 Andrea E. F. Clementi, Angelo Monti, and Riccardo Silvestri. Distributed broadcast in radio networks of unknown topology. *Theoretical Computer Science*, 302(1-3):337–364, 2003. doi:10.1016/S0304-3975(02)00851-4.

10 Artur Czumaj and Wojciech Rytter. Broadcasting algorithms in radio networks with unknown topology. In *Proc. 44th Symposium on Foundations of Computer Science (FOCS'03)*, pages 492–501. IEEE Computer Society, 2003.

11 Gianluca De Marco. Distributed broadcast in unknown radio networks. *SIAM Journal on Computing*, 39(6):2162–2175, March 2010.

12 Leszek Gąsieniec, Erez Kantor, Dariusz R. Kowalski, David Peleg, and Chang Su. Time efficient k-shot broadcasting in known topology radio networks. *Distributed Computing*, 21(2):117–127, 2008. doi:10.1007/s00446-008-0058-0.

13 Erez Kantor and David Peleg. Efficient k-shot broadcasting in radio networks. *Discrete Applied Mathematics*, 202:79–94, 2016.

14 Sushanta Karmakar, Paraschos Koutris, Aris Pagourtzis, and Dimitris Sakavalas. Energy-efficient broadcasting in ad hoc wireless networks. *Journal of Discrete Algorithms*, 42:2–13, 2017. doi:10.1016/j.jda.2016.11.004.

15 Dariusz R. Kowalski and Andrzej Pelc. Broadcasting in undirected ad hoc radio networks. *Distributed Computing*, 18(1):43–57, 2005. doi:10.1007/s00446-005-0126-7.

16 Felix Lazebnik and Vasiliy A. Ustimenko. Explicit construction of graphs with an arbitrary large girth and of large size. *Discrete Applied Mathematics*, 60(1-3):275–284, 1995. doi:10.1016/0166-218X(94)00058-L.

17 Gianluca De Marco and Andrzej Pelc. Faster broadcasting in unknown radio networks. *Inf. Process. Lett.*, 79(2):53–56, 2001. doi:10.1016/S0020-0190(00)00178-2.

Depth Two Majority Circuits for Majority and List Expanders

Kazuyuki Amano[1]
Department of Computer Science, Gunma University,
1-5-1 Tenjin, Kiryu, Gunma 376-8515, Japan
amano@gunma-u.ac.jp

Abstract

Let MAJ_n denote the Boolean majority function of n input variables. In this paper, we study the construction of depth two circuits computing MAJ_n where each gate in a circuit computes MAJ_m for $m < n$.

We first give an explicit construction of depth two $\mathsf{MAJ}_{\lfloor n/2 \rfloor + 2} \circ \mathsf{MAJ}_{\leq n-2}$ circuits computing MAJ_n for every $n \geq 7$ such that $n \equiv 3 \pmod 4$ where MAJ_m and $\mathsf{MAJ}_{\leq m}$ denote the majority gates that take m and at most m *distinct* inputs, respectively. A graph theoretic argument developed by Kulikov and Podolskii (STACS '17, Article No. 49) shows that there is no $\mathsf{MAJ}_{\leq n-2} \circ \mathsf{MAJ}_{n-2}$ circuit computing MAJ_n. Hence, our construction reveals that the use of a smaller fan-in gates at the bottom level is essential for the existence of such a circuit. Some computational results are also provided.

We then show that the construction of depth two $\mathsf{MAJ}_m \circ \mathsf{MAJ}_m$ circuits computing MAJ_n for $m < n$ can be translated into the construction of a newly introduced version of bipartite expander graphs which we call a *list expander*. Intuitively, a list expander is a c-leftregular bipartite graph such that for a given $d < c$, every d-leftregular subgraph of the original graph has a certain expansion property. We formalize this connection and verify that, with high probability, a random bipartite graph is a list expander of certain parameters. However, the parameters obtained are not sufficient to give us a $\mathsf{MAJ}_{n-c} \circ \mathsf{MAJ}_{n-c}$ circuit computing MAJ_n for a large constant c.

2012 ACM Subject Classification Theory of computation → Circuit complexity

Keywords and phrases Boolean function, majority function, constant depth circuit, expander graph

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.81

Acknowledgements The author would like to thank anonymous reviewers for their careful reading and many constructive suggestions.

1 Introduction

Let MAJ_n denote the Boolean majority function of n variables, i.e.,

$$\mathsf{MAJ}_n(x_1, \ldots, x_n) \quad = \quad \mathbf{1}[\sum_{i=1}^{n} x_i \geq n/2],$$

here $\mathbf{1}[\cdot]$ denotes 1 if the condition in the bracket is satisfied, and 0 otherwise.

The problem of finding an efficient construction of circuits (or formulas) computing the majority function has attracted many researchers for a long time (e.g., [2, 10, 13]). The

[1] A part of this work was supported by JSPS Kakenhi 15K00006, 18K11152 and 18H04090.

famous AKS sorting network [1] can be viewed as an $O(\log n)$ depth circuit computing MAJ_n if we pick the middle bit of all the outputs. A probabilistic construction of $\sim 5.3 \log n$-depth Boolean formula by Valiant [13] is another beautiful result on this problem. Here we mention that both of these constructions are monotone, i.e., they do not use negation gates. For more backgrounds and results, see e.g., the introduction of [4] or [7].

Recently, Kulikov and Podolskii [7] initiated the study on the "down scale" version of this problem. Specifically, they studied the construction of a small depth circuit for MAJ_n consisting of gates computing MAJ_m such that $m < n$.

In this paper, we restrict our attention to the case of depth two. Namely, we consider the following type of problems: What is the minimum value of m such that MAJ_n can be computed by a depth two circuits consisting of $\mathsf{MAJ}_{\leq m}$ gates? How to compute MAJ_n by a depth two circuit consisting of $\mathsf{MAJ}_{\leq m}$ gates for $m < n$?

In spite of the simplicity of the problem statement, its solution might be highly non-trivial even for small values of n. For example, constructing a depth two circuit consisting of the 5-bit majority gates that computes the 7-bit majority function is already not trivial.

Formally, we consider depth two circuits of the form $M_2 \circ M_1$ for $M_1, M_2 \in \{\mathsf{MAJ}_m, \mathsf{MAJ}_{\leq m}\}$ where MAJ_m and $\mathsf{MAJ}_{\leq m}$ denote the majority gates that take m and at most m inputs, respectively. Here M_2 denotes the top gate and M_1 denotes the bottom gates. In addition, we impose the restriction that each MAJ_m (or $\mathsf{MAJ}_{\leq m}$) gate on the bottom level takes m (or at most m) *distinct* input variables. Note that this restriction affects the difficulties for computing MAJ_n. Kulikov and Podolskii [7, Lemma 11] showed that there is no $\mathsf{MAJ}_{n-2} \circ \mathsf{MAJ}_{n-2}$ circuit computing MAJ_n for every odd n under this restriction. On the other hand, the author of this paper and Yoshida [3] showed that such circuits do exist for every odd $n \geq 7$ if some of the bottom gates are allowed to read an input variable multiple times. For example, the following circuit (which was presented by Kulikov and Podolskii [7, Introduction]) computes MAJ_7.

$$\begin{aligned}&\mathsf{MAJ}_7(x_1, \ldots, x_7)\\ &\quad = \mathsf{MAJ}_5(\mathsf{MAJ}_5(x_1, x_2, x_3, x_4, x_5), \mathsf{MAJ}_5(x_1, x_2, x_5, x_6, x_7), \mathsf{MAJ}_5(x_1, x_3, x_4, x_6, x_6),\\ &\qquad \mathsf{MAJ}_5(x_2, x_3, x_3, x_5, x_6), \mathsf{MAJ}_5(x_2, x_4, x_5, x_7, x_7)).\end{aligned}$$

One of the most interesting problems in this line of research is to find the minimum value of m such that MAJ_n can be computed by a $\mathsf{MAJ}_{\leq m} \circ \mathsf{MAJ}_{\leq m}$ circuit. Kulikov and Podolskii [7] proved a lower bound of $m \geq n^{13/19+o(1)}$. This was then improved to $m = \Omega(n^{0.8})$ by Engels, Garg, Makino and Rao [4]. For the upper bound, the construction of such a circuit for $(n, m) = (7, 5), (9, 7)$ and $(11, 9)$ is provided by Kulikov and Podolskii [7]. This was then generalized to $(n, m) = (n, n-2)$ for all odd $n \geq 7$ by the author of this paper and Yoshida [3]. However, in all of these constructions some of the gates at the bottom level read an input variable multiple times. In our restricted setting, i.e., every bottom gate can read each variable only once, no non-trivial upper bounds were previously known.

Note that, throughout the paper, we only use the gates computing the *standard* majority, i.e., the majority gates with the threshold value $m/2$ where m denotes its fan-in. In [11], Posobin gave the construction of a depth two circuit for MAJ_n consisting of gates with fan-in $m = (2/3)n + 4$ where the threshold value of the bottom gates is not restricted to $m/2$.

Contributions

The contribution of this paper is twofolds. First, we present an explicit construction of $\mathsf{MAJ}_{\lfloor n/2 \rfloor + 2} \circ \mathsf{MAJ}_{\leq n-2}$ circuits computing MAJ_n for every $n \geq 7$ such that $n \equiv 3 \pmod 4$. The construction is quite simple, but as far as we know, this is the first non-trivial depth

two circuits for MAJ_n consisting of the standard majority gates of fan-in strictly less than n and without multiple weights. Since a graph theoretic argument developed by Kulikov and Podolskii [7] can show that there is no $\mathsf{MAJ}_{\leq n-2} \circ \mathsf{MAJ}_{n-2}$ circuit computing MAJ_n, our construction reveals that the use of a smaller fan-in is essential for the existence of such a circuit. Some computational results are also provided in the paper.

During this work, we *feel* that constructing such a circuit for (n, m) with $m \ll n$ or even $m = n - 4$ may be a hard problem. This motivates us to give an explanation or evidence on this hardness, which leads to the second contribution of this paper.

In the second part of the paper, we show that the construction of depth two $\mathsf{MAJ}_m \circ \mathsf{MAJ}_m$ circuits computing MAJ_n for $m < n$ can be translated into the construction of a newly introduced version of bipartite expander graphs which we call a *list expander*. Intuitively, a list expander is a c-leftregular bipartite graph such that for a given $d < c$, every d-leftregular subgraph of the original graph has a certain expansion property. We formalize this connection, and verify that, with high probability, a random bipartite graph is a list expander of certain parameters. However, the parameters obtained are not sufficient to give us a $\mathsf{MAJ}_{n-c} \circ \mathsf{MAJ}_{n-c}$ circuit computing MAJ_n for a large constant c.

Organization of the Paper

The organization of the paper is as follows. In Section 2, we give an explicit construction of $\mathsf{MAJ}_{\lfloor n/2 \rfloor + 2} \circ \mathsf{MAJ}_{\leq n-2}$ circuits computing MAJ_n for $n \equiv 3 \pmod 4$. The results of some computational experiments on the total fan-in of the bottom gates of a $\mathsf{MAJ}_{\leq n-1} \circ \mathsf{MAJ}_{\leq n-1}$ circuit are also provided. In Section 3, we discuss the connection between the construction of depth two $\mathsf{MAJ}_m \circ \mathsf{MAJ}_m$ circuits and the construction of a newly introduced notion of bipartite expander graphs, which we call a *list expander*. Then, in Section 4, we give an analysis on the expansion property of a random bipartite graph. We close the paper by giving the conclusions in Section 5.

2 Depth Two Majority Circuits for Majority

For an integer n, $[n]$ denotes the set $\{1, 2, \ldots, n\}$. For a set S, $|S|$ denotes the cardinality of S. For an n-bit string $\boldsymbol{x} \in \{0,1\}^n$, x_i denotes the i-th bit of $\boldsymbol{x}$ and $|\boldsymbol{x}|$ denotes the number of 1's in $\boldsymbol{x}$, i.e., $|\boldsymbol{x}| = |\{i \in [n] : x_i = 1\}|$.

In this section, we first give a simple construction of depth two majority circuits for MAJ_n.

▶ **Theorem 1.** *For every $n \geq 7$ such that $n \equiv 3 \pmod 4$, there is a $\mathsf{MAJ}_{\lfloor n/2 \rfloor + 2} \circ \mathsf{MAJ}_{\leq n-2}$ circuit computing MAJ_n.*

We should note that in Theorem 1 the use of gates with fan-in less than $n-2$ at the bottom level is necessary since Kulikov and Podolskii [7, Lemma 11] proved that, for every odd n, there is no depth two $\mathsf{MAJ}_{n-2} \circ \mathsf{MAJ}_{n-2}$ circuit computing MAJ_n, and a simple generalization of their proof can be extended to establish the non-existence of $\mathsf{MAJ}_{\leq n-2} \circ \mathsf{MAJ}_{n-2}$ circuits for MAJ_n.

Proof. The proof is by construction.

Let $n = 4k+3$ for some $k \geq 1$. We will construct a $\mathsf{MAJ}_{2k+3} \circ \mathsf{MAJ}_{\leq 4k+1}$ circuit computing MAJ_n. For $1 \leq i \leq 2k+1$, let $S_i := [n] \backslash \{2i-1, 2i\}$. Let $S_{2k+2} := \{1, 3, ..., 4k+1\}$ and $S_{2k+3} := \{2, 4, ..., 4k+2\}$. For $1 \leq i \leq 2k+3$, the i-th bottom gate g_i computes the majority

of all the x_j's such that $j \in S_i$. The top gate C computes the majority of all the g_i's, i.e., $C(\boldsymbol{x}) := \mathbf{1}[\sum_i g_i(\boldsymbol{x}) \geq k+2]$. For ease of understanding, we show below a 0/1-matrix representing our circuit (for $n = 11$). The (i, j)-th entry is 1 iff the gate g_i reads the variable x_j.

```
0 0 1 1 1 1 1 1 1 1 1
1 1 0 0 1 1 1 1 1 1 1
1 1 1 1 0 0 1 1 1 1 1
1 1 1 1 1 1 0 0 1 1 1
1 1 1 1 1 1 1 1 0 0 1
1 0 1 0 1 0 1 0 1 0 0
0 1 0 1 0 1 0 1 0 1 0
```

Below we verify that this circuit correctly computes MAJ_n. We first observe that it is sufficient to verify $C(\boldsymbol{x}) = 0$ only for every input $\boldsymbol{x}$ with $|\boldsymbol{x}| = 2k+1$. The correctness for $\boldsymbol{x}$ with $|\boldsymbol{x}| = n - (2k+1) = 2k+2$ will follow from the fact that every majority gate in our circuit has an odd number of inputs and hence computes a self-dual function. This implies that $C(x_1, \ldots, x_n) = 1 - C(\overline{x_1}, \ldots, \overline{x_n})$. Then, all other cases will follow from the monotonicity of our circuit.

It is convenient to consider a hypergraph G over the vertex set $V = \{1, 2, \ldots, n\}$ consisting of edges $E_1, \ldots, E_{2k+3}$ where $E_i = [n] \backslash S_i$ for $1 \leq i \leq 2k+3$. Note that G contains $2k+1$ edges of size two and two edges of size $2k+2$. We call the former small edges, and the latter large edges. For a set of vertices $S \subseteq V$ and an edge E, we say that S *covers* E if $|E \cap S| \geq |E|/2$.

For $\boldsymbol{x} \in \{0,1\}^n$, let $V_{\boldsymbol{x}} := \{i \in [n] \mid x_i = 1\}$. It is easy to observe that $g_i(\boldsymbol{x}) = 0$ iff $V_{\boldsymbol{x}}$ covers E_i for $|\boldsymbol{x}| = 2k+1$. Therefore, it is sufficient to verify that the number of edges that are covered by $V_{\boldsymbol{x}}$ is at least $\lceil \frac{2k+3}{2} \rceil = k+2$ for every $\boldsymbol{x} \in \{0,1\}^n$ with $|\boldsymbol{x}| = 2k+1$.

We divide the proof into two cases.

Case 1 $4k+3 \notin V_{\boldsymbol{x}}$.

Observe that $V_{\boldsymbol{x}}$ covers at least $k+1$ small edges. If $V_{\boldsymbol{x}}$ covers $k+2$ or more small edges, then we are done. Therefore, we can assume that $V_{\boldsymbol{x}}$ covers exactly $k+1$ small edges. However, in this case, $V_{\boldsymbol{x}}$ must cover one of two large edges, and hence $V_{\boldsymbol{x}}$ covers $k+2$ edges in total. This completes the proof.

Case 2 $4k+3 \in V_{\boldsymbol{x}}$.

We view that small edges form a bipartite matching between $2k+1$ left vertices (odd numbers up to $4k+1$) and $2k+1$ right vertices (even numbers up to $4k+2$). If $V_{\boldsymbol{x}}$ covers at least $k+1$ left vertices of small edges or at least $k+1$ right vertices of small edges, then at least $k+2$ edges are covered in total since $V_{\boldsymbol{x}}$ also covers one of two large edges. If $V_{\boldsymbol{x}}$ covers exactly k left vertices and k right vertices of small edges, then it also covers both large edges and hence $V_{\boldsymbol{x}}$ covers at least $k+2$ edges in total. This completes the proof. ◀

Remark that it is quite plausible that there are many other constructions of $\mathsf{MAJ}_{\leq n-2} \circ \mathsf{MAJ}_{\leq n-2}$ circuit for MAJ_n. For example, for every $n \geq 9$ with $n \equiv 3 \pmod 6$, it seems likely that the $\mathsf{MAJ}_{\frac{3}{2}n+1} \circ \mathsf{MAJ}_{\leq n-2}$ circuit given below also computes MAJ_n. Let $n = 3k$ and suppose that k is odd. For $i \in [k]$, $S_{2i-1} = [n] \backslash \{3i-2, 3i\}$ and $S_{2i} = [n] \backslash \{3i-1, 3i\}$. In

addition, we let $S_{2k+1} := \{3, 6, \ldots, 3k\}$. For $i \in [2k+1]$, the i-th bottom gate g_i is defined to be

$$g_i(\boldsymbol{x}) = \mathbf{1}\left[\sum_{i \in S_i} x_i \geq \frac{|S_i|}{2}\right],$$

and the top gate computes the majority of all the g_i's. We have computationally verified the correctness of this circuit up to $n = 27$, and it may be a good exercise to give a formal proof.

Sum of Bottom Fan-ins

Let $F(n)$ denote the smallest number of the total fan-ins of all the bottom gates of a depth two $\mathsf{MAJ}_{\leq n-1} \circ \mathsf{MAJ}_{\leq n-1}$ circuit that computes MAJ_n. It would be interesting to find the value of $F(n)$. The circuit constructed in the proof of Theorem 1 gives the upper bound of $F(n) \leq (n-2)\lfloor \frac{n}{2} \rfloor + \lfloor \frac{n}{2} \rfloor \cdot 2 = n\lfloor \frac{n}{2} \rfloor \sim n^2/2$.

We show below some computational results on $F(n)$ for small values of n. We obtain these results by using an IP solver [6]. More specifically, if we fix the fan-in of the top gate to m_2, then the optimal value of the following IP problem gives $F(n)$. Intuitively, for each $S \subseteq [n]$, G_S represents the number of wires connecting from the bottom gates $\mathbf{1}[\sum_{i \in S} x_i \geq \frac{|S|}{2}]$ to the top gate. The number of variables is $\sim 2^n$, and the number of constraints is $\sim 2\binom{n}{\lfloor n/2 \rfloor}$ as we only need to check for $\boldsymbol{x}$ with $|\boldsymbol{x}| \in \{\lceil n/2 \rceil, \lfloor n/2 \rfloor\}$ by monotonicity.

$$\begin{aligned}
\text{Minimize:} \quad & \sum_{\emptyset \neq S \subsetneq [n]} |S| G_S \\
\text{Subject to:} \quad & \sum_{\emptyset \neq S \subsetneq [n]} G_S = m_2, \\
& \sum_{S : |S \cap V_{\boldsymbol{x}}| \geq \frac{|S|}{2}} G_S \geq \frac{m_2}{2}, \quad (\forall \boldsymbol{x} \in \{0,1\}^n \text{ s.t. } \mathsf{MAJ}_n(\boldsymbol{x}) = 1), \\
& \sum_{S : |S \cap V_{\boldsymbol{x}}| \geq \frac{|S|}{2}} G_S < \frac{m_2}{2}, \quad (\forall \boldsymbol{x} \in \{0,1\}^n \text{ s.t. } \mathsf{MAJ}_n(\boldsymbol{x}) = 0). \\
& G_S \geq 0 : \text{ integer} \quad (\forall S).
\end{aligned}$$

Here $V_{\boldsymbol{x}}$ denotes $\{i \in [n] : x_i = 1\}$ as in the proof of Theorem 1.

Note that $F(n)$ is undefined for $n \leq 5$ since a simple exhaustive search shows that there is no $\mathsf{MAJ}_{\leq n-1} \circ \mathsf{MAJ}_{\leq n-1}$ circuit computing MAJ_n for $n \leq 5$. The result of our experiments is as follows.

- $F(6) = 18$.
- $F(7) = 21$. This is identical to the value given by the circuit in the proof of Theorem 1.
- $F(8) = 31$.
- $F(9) = 37$.
- $F(10) = 50$.
- $F(11) = 55$. This is identical to the value given by the circuit in the proof of Theorem 1.

We show an example of circuits that attain $F(n)$ for $n = 6, 8, 9$ and 10 in Table 1. Given these data, we conjecture that $F(n)$ is of the order of n^2.

▶ **Conjecture 2.** *$F(n) = \Theta(n^2)$. More ambitiously, $F(n) \sim \frac{n^2}{2}$.*

Table 1 The matrix representation of an example of a $\mathsf{MAJ}_{\leq n-1} \circ \mathsf{MAJ}_{\leq n-1}$ circuit for MAJ_n of a smallest total fan-ins for $n = 6, 8, 9$ and 10. Here the (i, j)-entry of each matrix represents whether the i-th bottom gate reads the variable x_j.

0 0 1 1 1 1	0 0 1 1 1 1 0 0	0 0 0 1 1 0 0 0 1	0 0 1 1 0 1 1 0 1 0
1 1 0 0 1 1	0 0 0 0 1 1 1 1	0 1 1 1 0 1 1 1 1	0 0 1 1 1 1 1 0 1 0 0
1 1 1 1 0 0	0 0 1 1 0 0 1 1	0 1 1 1 1 0 0 0 1	1 1 0 0 0 0 0 1 1 1
0 1 0 1 0 1	1 1 0 0 1 1 0 0	1 0 0 0 0 1 1 0 0	1 1 0 0 1 0 1 0 0 1
1 0 1 0 1 0	1 1 1 1 0 0 0 0	1 1 0 0 0 1 1 1 0	1 1 1 1 0 1 0 0 0 1
	1 1 0 0 0 0 1 1	1 1 1 0 1 0 1 1 1	1 1 1 1 1 0 1 1 1 0
	1 1 1 1 1 1 1 0	1 1 1 1 1 1 0 1 0	0 1 0 1 1 1 1 1 1 1
			1 0 1 0 1 1 1 1 1 1

3 Connection between Circuits and List Expanders

For a $\mathsf{MAJ}_{m_2} \circ \mathsf{MAJ}_{m_1}$ circuit on n input variables, we naturally associate a bipartite graph over m_2 left vertices and n right vertices as follows.

▶ **Definition 3.** Let C be a $\mathsf{MAJ}_{m_2} \circ \mathsf{MAJ}_{m_1}$ circuit on n input variables where the bottom gates are labeled by $g_1, \ldots, g_{m_2}$. We define a bipartite graph $G_C = (L \cup R, E)$ *associated to* C as follows: Let $L = \{g_1, \ldots, g_{m_2}\}$ and $R = \{x_1, \ldots, x_n\}$, i.e., each left vertex is corresponding to a bottom gate and each right vertex is corresponding to an input variable. For $g_i \in L$ and $x_j \in R$, $(g_i, x_j) \in E$ iff the gate g_i *does not* read x_j as an input in C.

Let $G = (V, E)$ be a graph. For a vertex $v \in V$, $N_G(v)$ denotes the neighbors of v, and for a set of vertices $S \subseteq V$, $N_G(S)$ denotes the neighbors of S, i.e., $N_G(S) := \{u : \exists v \in S \text{ s.t. } (v, u) \in E\}$.

By Definition 3, a bipartite graph G_C associated to a $\mathsf{MAJ}_{m_2} \circ \mathsf{MAJ}_{n-\ell_1}$ circuit C is ℓ_1-*leftregular*, which means that every left vertex has exactly ℓ_1 neighbors, i.e., $|N_{G_C}(v)| = \ell_1$ for every $v \in L$.

The following is the main theorem in this section, which translates the construction of depth two circuits computing MAJ_n into the construction a bipartite graph with a certain expansion property.

▶ **Theorem 4.** *Suppose that n is an odd integer. For every even numbers $\ell_1, \ell_2 > 0$, the following are equivalent.*

- **(i)** *A depth two $\mathsf{MAJ}_{n-\ell_2} \circ \mathsf{MAJ}_{n-\ell_1}$ circuit C computes MAJ_n.*
- **(ii)** *Let $G_C = (L \cup R, E)$ be a ℓ_1-leftregular bipartite graph associated to C. Then, for every $(\ell_1/2 + 1)$-leftregular subgraph $G' \subseteq G_C$ over the same set of vertices of G_C,*

$$|N_{G'}(S)| \geq \left\lceil \frac{|R|}{2} \right\rceil + 1 \text{ for every } S \subseteq L \text{ with } |S| = \left\lceil \frac{|L|}{2} \right\rceil.$$

Proof. By using an argument similar to the one used in the proof of Theorem 1, we see that the statement (i) is equivalent to the statement that for every $\boldsymbol{x} \in \{0,1\}^n$ with $|\boldsymbol{x}| = \lceil n/2 \rceil$, $C(\boldsymbol{x}) = 1$, which we refer to as (i'). Here we use the assumption that n is odd as well as the fan-in of each gate is also odd.

(i') $\to$ (ii) Assume for contradiction that there exists an $(\ell_1/2 + 1)$-leftregular subgraph G' of G_C such that $|N_{G'}(S)| \leq \lceil |R|/2 \rceil$ for some $S \subseteq L$ with $|S| = \lceil |L|/2 \rceil$. Fix an arbitrary such S. Let $X_1 \subseteq R$ be an arbitrary superset of $N_{G'}(S)$ with $|X_1| = \lceil |R|/2 \rceil = \lceil n/2 \rceil$. Define

$\boldsymbol{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ so that $x_j = 1$ iff $x_j \in X_1$ for $j \in [n]$. For every $g_i \in S$, we have $g_i(\boldsymbol{x}) = 0$ and hence $C(\boldsymbol{x}) = 0$ since $|S| = \lceil |L|/2 \rceil$. This contradicts the statement (i'), completing the proof.

(ii) $\to$ (i') Assume for contradiction that there exists an input $\boldsymbol{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ such that $|\boldsymbol{x}| = \lceil n/2 \rceil$ and $C(\boldsymbol{x}) = 0$. Fix an arbitrary such $\boldsymbol{x}$. Define $X_1 \subseteq R$ as $X_1 := \{x_j : x_j = 1\}$, and for $1 \leq i \leq |L|$, let $V_i \subseteq R$ be the set of variables x_j such that g_i reads x_j. By the construction of the graph G_C, we have

$$\left| \left\{ g_i \in L : |V_i \cap X_1| \leq \left\lfloor \frac{n - \ell_1}{2} \right\rfloor \right\} \right| \geq \left\lceil \frac{|L|}{2} \right\rceil.$$

Since $|X_1| = \lceil n/2 \rceil$, this is equivalent to

$$\left| \left\{ g_i \in L : |\overline{V_i} \cap X_1| \geq \frac{\ell_1}{2} + 1 \right\} \right| \geq \left\lceil \frac{|L|}{2} \right\rceil,$$

where $\overline{V_i}$ denotes the set $R \backslash V_i$.

Let S be an arbitrary $\lceil |L|/2 \rceil$-element subset of $\left\{ g_i \in L : |\overline{V_i} \cap X_1| \geq \frac{\ell_1}{2} + 1 \right\}$. Then we can pick an $(\ell_1/2 + 1)$-leftregular subgraph G' of G_C such that $N_{G'}(g_i) \subseteq X_1$ for every $g_i \subseteq S$. For this graph G', $N_{G'}(S) \subseteq X_1$, and hence $|N_{G'}(S)| \leq |X_1| = \lceil n/2 \rceil$, contradicting the statement (ii). ◀

We see that the statement (ii) in Theorem 4 has a similar spirit to bipartite expander graphs. This leads us to the following definition.

▶ **Definition 5.** Let $d_1 \geq d_2 > 0$ be two integers. A d_1-leftregular bipartite graph $G = (L \cup R, E)$ with $|L| = m$ and $|R| = n$ is said to be an $((m,n),(d_1,d_2),K,A)$ *list expander* if for every d_2-leftregular subgraph G' of G over the same set of vertices of G, it holds that

$$|N_{G'}(S)| \geq A \text{ for every } S \subseteq L \text{ with } |S| = K.$$

When $d_1 = d_2$ we simply call it an expander. Formally, a d_1-leftregular bipartite graph $G = (L \cup R, E)$ with $|L| = m$ and $|R| = n$ is said to be an $((m,n),d_1,K,A)$ *expander* if

$$|N_G(S)| \geq A \text{ for every } S \subseteq L \text{ with } |S| = K.$$

Note that our definition of an expander is different from the usual definition in a sense that the expansion property is only required for the sets of size equal to K.

The existence of a $\mathsf{MAJ}_{n-4} \circ \mathsf{MAJ}_{n-4}$ circuit computing MAJ_n is equivalent to the existence of an $((n-4,n),(4,3),\lceil \frac{n-4}{2} \rceil, \lceil \frac{n}{2} \rceil + 1)$ list expander. Moreover, an explicit construction of such a circuit implies an explicit construction of such an expander, and vice versa. Currently, we do not know whether such an expander can be built from known constructions of expanders in the usual sense.

A similar argument to the proof of Theorem 4 can show that a list expander of the form "d_1 choose d_2" for $d_2 > d_1/2 + 1$ captures a promise version of the problem for constructing majority circuits. The proof of Theorem 6 is very similar to the proof of Theorem 4 and omitted in this version.

▶ **Theorem 6.** *Let n be an odd integer, and $c \geq 2$ be an even integer. Let C be a $\mathsf{MAJ}_{n-c} \circ \mathsf{MAJ}_{n-c}$ circuit over $\{x_1, \ldots, x_n\}$ and let G_C be a c-leftregular bipartite graph associated to C. Suppose that d is an integer such that $d \geq \frac{c}{2} + 1$. Then, the following are equivalent.*

(i) *$C(\boldsymbol{x}) = 1$ for every $|\boldsymbol{x}| = \lfloor \frac{n}{2} \rfloor - \frac{c}{2} + d$ and $C(\boldsymbol{x}) = 0$ for every $|\boldsymbol{x}| = \lceil \frac{n}{2} \rceil + \frac{c}{2} - d$.*

(ii) *G_C is an $((n-c,n),(c,d),\lceil \frac{n-c}{2} \rceil, \lfloor \frac{n}{2} \rfloor - \frac{c}{2} + d)$ list expander.*

Algorithm 1 Construct an witness for refuting an expansion property.

1: **procedure** CONVERT(G)
2: $\tilde{S} := S$ and $\tilde{T} := N_G(S)$
3: $d := d_1$
4: **while** $d > d_2$ **do**
5: $S' := \tilde{S}$ and $T' := \tilde{T}$
6: **repeat**
7: Let v^* be a vertex $v \in T'$ minimizing $|N_G(v) \cap S'|$. Ties are broken arbitrary.
8: Put $S_{v^*} := N_G(v^*) \cap S'$
9: Remove all edges connecting v^* and a vertex in S' from G
10: $\tilde{T} := \tilde{T} \backslash \{v^*\}$
11: $S' := S' \backslash S_{v^*}$
12: $T' := T' \backslash (N_G(S_{v^*}) \cap T')$
13: **until** $|\tilde{T}| < \left(1 + \frac{d-d_2}{5d_1^2}\right) A$
14: $d := d - 1$
15: **end while**
16: **end procedure**

3.1 List Expanders imply Expanders

In this subsection, we give a result roughly stating that a list expander implies an expander of a certain expansion factor, and observe that converse of it is not true.

▶ **Theorem 7.** *Let $d_1 > d_2 > 0$ be two integers. Suppose that a bipartite graph $G = (L \cup R, E)$ is an $((m,n),(d_1,d_2),K,A)$ list expander for $A \geq K$. Then, G is an $((m,n),d_1,K,(1+\epsilon)A)$ expander for $\epsilon = (d_1 - d_2)/(5d_1^2)$.*

Note that we did not optimize the constant ϵ in Theorem 7.

Proof. The proof is by contrapositive. Put $\epsilon = (d_1 - d_2)/(5d_1^2)$. Assume that a d_1-leftregular bipartite graph $G = (L \cup R, E)$ is not an $((m,n),d_1,K,(1+\epsilon)A)$ expander for $A \geq K$. This means that there exists a set of vertices $S \subseteq L$ with $|S| = K$ such that $|N_G(S)| < (1+\epsilon)A$. We will see that for this case, G is not an $((m,n),(d_1,d_2),K,A)$ list expander. In order to show this, it is sufficient to show that there exists a d_2-leftregular subgraph G' of G such that for some $\tilde{S} \subseteq L$ with $|\tilde{S}| = K$ and for some $\tilde{T} \subseteq R$ with $|\tilde{T}| < A$, $|N_{G'}(\tilde{S})| \subseteq \tilde{T}$ holds.

We set $\tilde{S} := S$. We will find a subgraph G' and a set $\tilde{T} \subseteq T$ by using Algorithm 1 shown above.

First we observe that whenever $|T'| \geq A$, there are at least $A/2$ v's in T' such that $|N_G(v) \cap \tilde{S}| \leq 2d_1$ since if otherwise, we have $\sum_{v \in T'} |N_G(v) \cap \tilde{S}| > Ad_1 \geq Kd_1$, which contradicts the assumption that G is d_1-leftregular. Suppose that a vertex v^* chosen in Line 7 satisfies $|S_{v^*}| \leq 2d_1$. We refer to this condition as (*). Then, in Line 12, $|T'|$ decreases at most $2d_1^2$. For every fixed d, the number of iterations of Lines 6 to 13 is at least $\frac{1}{5d_1^2}A$. Since $2d_1^2 \frac{1}{5d_1^2} A < \frac{1}{2}A$, the condition (*) is always satisfied.

A crucial observation is that after each execution of the **While** loop in the algorithm, G satisfies that $|N_G(v)| \geq d$ for every $v \in \tilde{S}$ and $N_G(\tilde{S}) \subseteq \tilde{T}$. Hence, after the execution of the entire algorithm, we have a graph G such that $|N_G(v)| \geq d_2$ for every $v \in \tilde{S}$ and $N_G(\tilde{S}) \subseteq \tilde{T}$ with $|\tilde{T}| < A$. Then, an arbitrary d_2-leftregular subgraph of G is not an expander for the parameter given in the statement of the theorem. ◀

We remark that an argument below suggests that the converse of Theorem 7 is not true. Let $G = (L \cup R, E)$ be a bipartite graph with $L = R = \mathbb{Z}_p$ and edge set $\{(x, x+1), (x, x-1), (x, x^{-1})\}_{x \in \mathbb{Z}_p}$ where all arithmetic is modulo p and define 0^{-1} to be 0. It was shown that this 3-leftregular bipartite graph has an expansion property with some constant expansion factor. (This was stated implicitly in [8]. See e.g., [12, Construction 4.26] for the statement in this form.) If we add the edges $\{(x, x)\}_{x \in \mathbb{Z}_p}$ to G (and make a slight rearrangement for $x = 0$ to avoid parallel edges $(0, 0)$), then it becomes a 4-leftregular expander. However, a 3-leftregular subgraph consisting of the edges $\{(x, x+1), (x, x), (x, x-1)\}_{x \in \mathbb{Z}_p}$ is not an expander anymore.

4 Probabilistic Constructions

In this section, we show a statement roughly saying that a random c-leftregular bipartite graph is a list expander of the form "c choose d" for $d \sim \frac{2}{3}c$ with a non-negligible probability. Note that this is not sufficient for obtaining a non-trivial depth two circuit computing the majority function, for which we need $d \sim \frac{1}{2}c$.

We first show a fact that will be used in our analysis.

▶ **Fact 8.** *Suppose that a, b, p are positive integers satisfying $a - p > b - p > 0$. Then,*

$$\frac{a^{a-p}}{b^{b-p}} \geq \frac{(a-p)^{a-p}}{(b-p)^{b-p}}.$$

Proof. The fact can be verified by the following series of inequalities.

$$\begin{aligned}
&a^{a-p}(b-p)^{b-p} - (a-p)^{a-p}b^{b-p} \\
&= (a-p+a)^{a-p}(b-p)^{b-p} - (a-p)^{a-p}(b-p+b)^{b-p} \\
&= \sum_{i=0}^{a-p} \binom{a-p}{i} (a-p)^{a-p-i} a^i (b-p)^{b-p} - \sum_{i=0}^{b-p} \binom{b-p}{i} (b-p)^{b-p-i} b^i (a-p)^{a-p} \\
&\geq \sum_{i=0}^{b-p} (a-p)^{a-p-i}(b-p)^{b-p-i} \left\{ \binom{a-p}{i}(b-p)^i - \binom{b-p}{i}(a-p)^i \right\} \qquad (1) \\
&\geq 0.
\end{aligned}$$

Here the last inequality follows from the fact that the value inside the bracket in Eq. (1) is non-negative for every fixed i. ◀

The following is the main theorem in this section.

▶ **Theorem 9.** *Let $\epsilon > 0$ be an arbitrary small positive constant. Suppose that c is a constant such that $\epsilon c \geq 3$, and let $m = n - c$. Then for every sufficiently large n, there is an $((m, n), (c, d), \lceil \frac{m}{2} \rceil, \lfloor \frac{n}{2} \rfloor - \frac{c}{2} + d)$ list expander, where $d = \lceil (\frac{2}{3} + \epsilon)c \rceil$.*

By Theorem 6 and Theorem 9, we can see that for every sufficiently large n, there exists a $\mathsf{MAJ}_{n-c} \circ \mathsf{MAJ}_{n-c}$ circuit C such that $C(\boldsymbol{x}) = \mathsf{MAJ}_n(\boldsymbol{x})$ for every $\boldsymbol{x} \in \{0,1\}^n$ such that $|\boldsymbol{x}| \lesssim \frac{n}{2} - \frac{c}{6}$ or $|\boldsymbol{x}| \gtrsim \frac{n}{2} + \frac{c}{6}$.

The rest of this section is devoted to prove Theorem 9. The proof relies on relatively basic but a bit lengthy calculations.

Proof. Consider a bipartite graph over the vertex set $L \cup R$ with $|L| = m (= n - c)$ and $|R| = n$. We view a c-leftregular bipartite graph as a union of c bipartite matchings. For $i \in [c]$, let M_i be a bipartite matching over $L \cup R$ in which the left vertices are saturated. A c-leftregular bipartite graph $G = (L \cup R, E)$ is specified by a sequence of c such matchings

$E = (M_1, \ldots, M_c)$, which we denoted by G_E. Let $\mathcal{E}$ denote the set of all possible E's and thus $|\mathcal{E}| = (\frac{n!}{c!})^c$. Notice that at this moment, a graph obtained in this way may have a multiedge. We will take into account this issue later.

Roughly speaking, a c-leftregular bipartite graph is a list expander if every d-leftregular subgraph of it is an expander. For $d \leq c$, a d-leftregular subgraph of G is specified by a sequence of m sets $D_i \subseteq [c]$ $(1 \leq i \leq m)$ of size d. Given a sequence of c matchings $E = (M_1, \ldots, M_c)$ and a sequence of m sets $\mathbf{D} = (D_1, \ldots, D_m)$, let $G_{E,\mathbf{D}}$ denote a d-leftregular bipartite graph such that for every $i \in [m]$, the set of edges connecting to the i-th left vertex consists of an edge in M_j for $j \in D_i$. Let $\mathcal{D}$ denote the set of all possible $\mathbf{D}$'s and thus $|\mathcal{D}| = \binom{c}{d}^m$.

We define $\mathcal{E}' \subseteq \mathcal{E}$ as the set of all E's in $\mathcal{E}$ such that G_E is a c-leftregular bipartite graph where G_E does not contain a multiedge. The following lemma says that the cardinality of $\mathcal{E}'$ is not too small compared to the cardinality of $\mathcal{E}$. Here e denotes the base of the natural logarithm. The proof of the lemma is postponed to the end of this section.

▶ **Lemma 10.** *For all sufficiently large n, the cardinality of $\mathcal{E}' \subseteq \mathcal{E}$ satisfies*

$$|\mathcal{E}'| \quad \geq \quad \frac{(n!)^c}{e^{\binom{c}{2}} n^{c^2}}.$$

Given a set $S \subseteq L$ and a set $T \subseteq R$ and a sequence $\mathbf{D} \in \mathcal{D}$, we count the number of $E \in \mathcal{E}$ such that $G_{E,\mathbf{D}}$ satisfies $N_{G_{E,\mathbf{D}}}(S) \subseteq T$, which we refer to as the property $P_{S,T}$.

For $j \in [c]$, let $a_j := |\{i \in S : j \notin D_i\}|$. Define an auxiliary function f as

$$f((m,n),(\ell,r),a) \quad = \quad \frac{r!(n-(\ell-a))!}{(r-(\ell-a))!(n-m)!}.$$

This represents the number of left saturated bipartite matchings in the $(m+n)$-vertex complete bipartite graph such that the neighbors of some specified $l-a$ left vertices are contained in some specified r right vertices. Then, the number of $E \in \mathcal{E}$ such that $G_{E,\mathbf{D}}$ satisfies $P_{S,T}$ is given by

$$\prod_{j \in [c]} f((m,n),(|S|,|T|),a_j). \tag{2}$$

We say that the a_j's are equally distributed if, for every $i, j \in [c]$, $|a_i - a_j| \leq 1$. The following lemma is useful for our calculation.

▶ **Lemma 11.** *Eq. (2) is maximized when the a_j's are equally distributed.*

Proof. Let $b_j = \ell - a_j$, i.e. b_j is the number of edges in M_j connecting to a vertex in S in the graph $G_{E,\mathbf{D}}$. Suppose that $a_i = a_j + \alpha$ for some $i, j \in [n]$ and for some $\alpha \geq 2$. In order to prove the lemma, it is sufficient to show that

$$g(a_i)g(a_j) \leq g(a_i - 1)g(a_j + 1),$$

where $g(a)$ stands for $f((m,n),(\ell,r),a)$. This is verified by the following.

$$\begin{aligned}
& g(a_i-1)g(a_j+1)-g(a_i)g(a_j) \\
&= \left(\frac{r!}{(n-m)!}\right)^2 \left(\frac{(n-(b_i+1))!}{(r-(b_i+1))!}\frac{(n-(b_i+\alpha-1))!}{(r-(b_i+\alpha-1))!} - \frac{(n-b_i)!}{(r-b_i)!}\frac{(n-(b_i+\alpha))!}{(r-(b_i+\alpha))!}\right) \\
&= \left(\frac{r!}{(n-m)!}\right)^2 \left(\frac{(n-(b_i+1))!}{(r-(b_i+1))!}\frac{(n-(b_i+\alpha))!}{(r-(b_i+\alpha))!}\right)\left(\frac{n-(b_i+\alpha-1)}{r-(b_i+\alpha-1)} - \frac{n-b_i}{r-b_i}\right) \\
&= \left(\frac{r!}{(n-m)!}\right)^2 \left(\frac{(n-(b_i+1))!}{(r-(b_i+1))!}\frac{(n-(b_i+\alpha))!}{(r-(b_i+\alpha))!}\right)\left(\frac{(n-r)(\alpha-1)}{(r-(b_i+\alpha-1))(r-b_i)}\right) \\
&\geq 0.
\end{aligned}$$

◀

We go back to the proof of Theorem 9.

Put $d=(\frac{2}{3}+\epsilon)c$, and put $|S|=\lceil\frac{n-c}{2}\rceil$ and $|T|=\lfloor\frac{n}{2}\rfloor-\frac{c}{2}+d$. Then, Eq. (2) is equal to

$$\prod_{i\in[c]}\frac{\left(\lfloor\frac{n}{2}\rfloor+\frac{c}{6}+c\epsilon\right)!\cdot\left(n-(\lceil\frac{n}{2}\rceil-\frac{c}{2}-a_j)\right)!}{\left(\lfloor\frac{n}{2}\rfloor+\frac{c}{6}+c\epsilon-(\lceil\frac{n}{2}\rceil-\frac{c}{2}-a_j)\right)!\cdot c!} = \prod_{i\in[c]}\frac{\left(\lfloor\frac{n}{2}\rfloor+\frac{c}{6}+c\epsilon\right)!\cdot\left(\lfloor\frac{n}{2}\rfloor+\frac{c}{2}+a_j\right)!}{\left(\frac{2}{3}c+c\epsilon-1+a_j\right)!\cdot c!}.$$

By Lemma 11, this is upper bounded by

$$\left(\frac{\left(\lfloor\frac{n}{2}\rfloor+\frac{c}{6}+c\epsilon\right)!\cdot\left(\lfloor\frac{n}{2}\rfloor+\frac{c}{2}+a\right)!}{\left(\frac{2}{3}c+c\epsilon-1+a\right)!\cdot c!}\right)^c,$$

where $a=(\frac{1}{3}-\epsilon)(\lceil\frac{n}{2}\rceil-\frac{c}{2})$. By substituting the RHS for a, the inside of the outer bracket of the above formula is upper bounded by

$$\frac{\left(\lceil\frac{n}{2}\rceil+\frac{c}{6}+c\epsilon\right)!\cdot\left(\frac{2}{3}n-\frac{\epsilon n}{2}+\frac{c}{3}+\frac{c\epsilon}{2}\right)!\cdot n^{C_0}}{\left(\frac{n}{6}-\frac{\epsilon n}{2}+\frac{c}{2}+\frac{3c\epsilon}{2}\right)!\cdot c!} \tag{3}$$

for some small positive constant C_0. Since c and ϵ are constants, there are large enough constants C_1 and C_2, Eq. (3) is upper bounded by

$$\begin{aligned}
\frac{\left(\frac{n}{2}\right)!\cdot\left(\frac{2}{3}n-\frac{\epsilon n}{2}\right)!\cdot n^{C_1}}{\left(\frac{n}{6}-\frac{\epsilon n}{2}\right)!} &\leq \frac{\left(\frac{n}{2}\right)^{\frac{n}{2}}\cdot\left(\frac{2}{3}n-\frac{\epsilon n}{2}\right)^{\left(\frac{2}{3}n-\frac{\epsilon n}{2}\right)}n^{C_2}}{\left(\frac{n}{6}-\frac{\epsilon n}{2}\right)^{\left(\frac{n}{6}-\frac{\epsilon n}{2}\right)}e^n} \\
&\leq \frac{\left(\frac{n}{2}\right)^{\frac{n}{2}}\cdot\left(\frac{2}{3}n\right)^{\left(\frac{2}{3}n-\frac{\epsilon n}{2}\right)}n^{C_2}}{\left(\frac{n}{6}\right)^{\left(\frac{n}{6}-\frac{\epsilon n}{2}\right)}e^n} \\
&= n^{C_2}\cdot\frac{n^n}{e^n}\cdot\left(\frac{2^{1/3}}{3^{1/2}}\right)^n\cdot\left(\frac{1}{2}\right)^{\epsilon n}. \qquad (4)
\end{aligned}$$

Here the first inequality is by the Stirling formula, and the second inequality is by Lemma 8.

The number of $E\in\mathcal{E}$ such that

$$^\exists S\subseteq L\ ^\exists T\subseteq R\ ^\exists\mathbf{D}\in\mathcal{D}\quad G_{E,\mathbf{D}}\text{ has the property }P_{S,T} \tag{5}$$

is at most

$$\frac{2^{n-c}\cdot 2^n\cdot(\text{Eq.}(4))^c\cdot|\mathcal{D}|}{\binom{c}{d}^{\lfloor\frac{m}{2}\rfloor}} = 2^{n-c}\cdot 2^n\cdot(\text{Eq.}(4))^c\cdot\binom{c}{d}^{\lceil\frac{m}{2}\rceil}. \tag{6}$$

By using the upper bound on the binomial coefficients (see e.g., [9, Lemma 9.2])

$$\binom{c}{\alpha c}\leq 2^{cH(\alpha)},$$

where $H(p) = -p\log_2(p) - (1-p)\log_2(1-p)$ is the binary entropy function, we have

$$\binom{c}{d}^{\lceil \frac{m}{2} \rceil} \leq 2^{H(\frac{1}{3}-\epsilon)\lceil \frac{m}{2} \rceil} < 2^{H(\frac{1}{3})\frac{n}{2}} = \left(\frac{3^{1/2}}{2^{1/3}}\right)^n.$$

Therefore, we have

$$\text{(Eq. (6))} \quad < \quad 2^{2n} n^{C_2 c} \cdot \frac{n^{cn}}{e^{cn}} \cdot \left(\frac{1}{2}\right)^{\epsilon cn}. \tag{7}$$

Recall that $\mathcal{E}' \subseteq \mathcal{E}$ is the set of all E's in $\mathcal{E}$ such that G_E is a c-leftregular bipartite graph where G_E does not contain a multiedge. The proof of Theorem 9 will be completed if we verify that Eq. (7) is smaller than the cardinality of $\mathcal{E}'$ since it implies the existence of a c-leftregular bipartite graph G_E not satisfying the condition Eq. (5), i.e., G_E is a list expander of the desired parameter.

By Lemma 10, we have

$$|\mathcal{E}'| \quad \geq \quad \frac{(n!)^c}{e^{\binom{c}{2}} n^{c^2}} \geq \frac{n^{cn}}{e^{cn}} \frac{1}{n^{C_3}},$$

for a sufficiently large constant C_3. We can see that when n goes to infinity the last term in the above is larger than Eq. (7) by recalling that $\epsilon c \geq 3$. ◀

Proof of Lemma 10. We use the result on counting the number of *Latin rectangles*. A $c \times n$ Latin rectangle is a $c \times n$ matrix L with symbols from $[n]$ such that each row and each column contains only distinct symbols. Let $L_{c,n}$ denote the number of $c \times n$ Latin rectangles. It was shown by Erdős and Kaplansky [5] that, for $c = O((\log^n)^{3/2-\epsilon})$,

$$L_{c,n} \sim \frac{(n!)^c}{e^{\binom{c}{2}}}.$$

It is easy to observe that a c-leftregular bipartite graph with $n-c$ left vertices and n right vertices can be naturally represented by a $c \times (n-c)$ matrix obtained from a $c \times n$ Latin rectangle by discarding the last c columns. Since we shall discard c^2 entries with symbols from $[n]$, the number of variations of such $c \times (n-c)$ matrices is at least $L_{c,n}/(n^{c^2})$, completing the proof of the lemma. ◀

5 Conclusions

In this paper, we introduce a new notion of expander graphs and give the translation result from the construction of depth two circuits computing the majority function. Currently, the existence of a $\mathsf{MAJ}_{n-c} \circ \mathsf{MAJ}_{n-c}$ circuit computing MAJ_n is open even for $c = 4$. We hope that our translation result leads us to a better understanding on this problem with the help of a rich theory on expander graphs. In addition, it would be interesting to find other applications of list expanders.

References

1 Miklós Ajtai, János Komlós, and Endre Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3(1):1–19, 1983.

2 Kazuyuki Amano. Bounds on the size of small depth circuits for approximating majority. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pages 59–70, 2009.

3 Kazuyuki Amano and Masafumi Yoshida. Depth two $(n-2)$-majority circuit for n-majority. *to appear in IEICE Trans. Fundamentals*, E101-A(9), 2018.

4 Christian Engels, Mohit Garg, Kazuhisa Makino, and Anup Rao. On expressing majority as a majority of majorities. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:174, 2017.

5 Paul Erdős and Irving Kaplansky. The asymptotic number of latin rectangles. *Amer. J. Math*, 68:230–236, 1946.

6 LLC Gurobi Optimization. Gurobi optimizer reference manual, 2018. URL: `http://www.gurobi.com`.

7 Alexander S. Kulikov and Vladimir V. Podolskii. Computing majority by constant depth majority circuits with low fan-in gates. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 49:1–49:14, 2017.

8 Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

9 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

10 Ryan O'Donnell and Karl Wimmer. Approximation by DNF: examples and counterexamples. In *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, pages 195–206, 2007.

11 Gleb Posobin. Computing majority with low-fan-in majority queries. *CoRR*, abs/1711.10176, 2017. `arXiv:1711.10176`.

12 Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012.

13 Leslie G. Valiant. Short monotone formulae for the majority function. *J. Algorithms*, 5(3):363–366, 1984.

Optimization over the Boolean Hypercube via Sums of Nonnegative Circuit Polynomials

Mareike Dressler
Goethe-Universität, FB 12 – Institut für Mathematik,
Robert-Mayer-Str. 6-10, 60054 Frankfurt am Main, Germany
dressler@math.uni-frankfurt.de

Adam Kurpisz[1]
ETH Zürich, Department of Mathematics,
Rämistrasse 101, 8092 Zürich, Switzerland
adam.kurpisz@ifor.math.ethz.ch

Timo de Wolff[2]
Technische Universität Berlin, Institut für Mathematik,
Straße des 17. Juni 136, 10623 Berlin, Germany
dewolff@math.tu-berlin.de

Abstract

Various key problems from theoretical computer science can be expressed as polynomial optimization problems over the boolean hypercube. One particularly successful way to prove complexity bounds for these types of problems are based on sums of squares (SOS) as nonnegativity certificates. In this article, we initiate optimization over the boolean hypercube via a recent, alternative certificate called sums of nonnegative circuit polynomials (SONC). We show that key results for SOS based certificates remain valid: First, there exists a SONC certificate of degree at most $n + d$ for polynomials which are nonnegative over the n-variate boolean hypercube with constraints of degree d. Second, if there exists a degree d SONC certificate for nonnegativity of a polynomial over the boolean hypercube, then there also exists a short degree d SONC certificate, that includes at most $n^{O(d)}$ nonnegative circuit polynomials. Finally, we show certain differences between SOS and SONC cones: we prove that, in contrast to SOS, the SONC cone is not closed under taking affine transformation of variables and that for SONC there does not exist an equivalent to Putinar's Positivestellensatz. We discuss these results both from algebraic and optimization perspective.

2012 ACM Subject Classification Mathematics of computing → Convex optimization

Keywords and phrases nonnegativity certificate, hypercube optimization, sums of nonnegative circuit polynomials, relative entropy programming, sums of squares

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.82

1 Introduction

An *optimization problem over a boolean hypercube* is an n-variate (constrained) polynomial optimization problem where the feasibility set is restricted to some vertices of an n-dimensional hypercube. This class of optimization problems belongs to the core of theoretical computer

[1] Swiss National Science Foundation project PZ00P2_174117 "Theory and Applications of Linear and Semidefinite Relaxations for Combinatorial Optimization Problems"
[2] DFG grant WO 2206/1-1

science. However, it is known that solving them is NP-hard in general, since one can easily cast, e.g., the Independent Set problem in this framework.

One of the most promising approaches in constructing theoretically efficient algorithms is the *sum of squares (SOS) hierarchy* [23, 45, 48, 55], also known as *Lasserre relaxation* [34]. The method is based on a Positivstellensatz result [50] saying that the polynomial f, which is nonnegative over the feasibility set, can be expressed as a sum of squares times the constraints defining the set. Bounding a maximum degree of a polynomial used in a representation of f provides a family of algorithms parametrized by an integer d. Finding a degree d SOS certificate for nonnegativity of f can be performed by solving a *semidefinite programming* (SDP) formulation of size $n^{O(d)}$. Finally, for every (feasible) n-variate hypercube optimization problem, with constraints of degree at most d, there exists a degree $2(n + \lceil d/2 \rceil)$ SOS certificate.

The SOS algorithm is a frontier method in algorithm design. It provides the best available algorithms used for a wide variety of optimization problems. The degree 2 SOS for the Independent Set problem implies the Lovász θ-function [40] and gives the Goemans-Williamson relaxation [20] for the Max Cut problem. The Goemans-Linial relaxation for the Sparsest Cut problem (analyzed in [2]) can be captured by the SOS of degree 6. Finally, the subexponential time algorithm for Unique Games [1] is implied by a SOS of sublinear degree [4, 24]. Moreover, it has been shown that SOS is equivalent in power to any SDP extended formulation of comparable size in approximating maximum constraint satisfaction problems (CSP) [39]. Recently SOS has been also applied to problems in dictionary learning [6, 54], tensor completion and decomposition [7, 26, 49], and robust estimation [28]. Other applications of the SOS method can be found in [4, 8, 12, 13, 16, 17, 24, 41, 42, 51], see also the surveys [14, 35, 37].

On the other hand it is known that the SOS algorithm admits certain weaknesses. For example, Grigoriev shows in [21] that a $\Omega(n)$ degree SOS certificate is needed to detect a simple integrality argument for the Knapsack problem, see also [22, 31, 36]. Other SOS degree lower bounds for Knapsack problems appeared in [11, 32]. Some lower bounds on the effectiveness of SOS has been shown for CSP problems [29, 56] and for planted clique problem [3, 43]. Finally degree $\Omega(\sqrt{n})$ SOS was proved to have problems scheduling unit size jobs on a single machine to minimize the number of late jobs [33]. The problem is solvable in polynomial time using the Moore-Hodgson algorithm. Finally, SOS has hard time proving global nonnegativity, as first proved by Hilbert [25]. Later an explicit example was given by Motzkin [44]. Moreover, as shown by Blekherman [9], there are significantly more nonnegative polynomials than SOS polynomials. The above arguments motivate the search of new nonnegativity certificates for solving optimization problems efficiently.

In this article, we initiate an analysis of hypercube optimization problems via *sums of nonnegative circuit polynomials (SONC)*. SONCs are a nonnegativity certificate introduced in [27], which are independent of sums of squares; see Definition 1 and Theorem 5 for further details. This means particularly that certain polynomials like the Motzkin polynomial, which have no SOS certificate for global nonnegativity, can be certified as nonnegative via SONCs. Moreover, SONCs generalize polynomials which are certified to be nonnegative via the arithmetic-geometric mean inequality [52]. Similarly as Lasserre relaxation for SOS, a Schmüdgen-like Positivstellensatz yields a converging hierarchy of lower bounds for polynomial optimization problems with compact constraint set; see [19, Theorem 4.8] and Theorem 6. These bounds can be computed via a convex optimization program called *relative entropy programming* [19, Theorem 5.3]. Our main question in this article is:

Can SONC certificates be an alternative for SOS methods for optimization problems over the hypercube?

We answer this question affirmatively in the sense that we prove SONC complexity bounds for boolean hypercube optimization analogous to the SOS bounds mentioned above. More specifically, we show:

1. For every polynomial which is nonnegative over an n-variate hypercube with constraints of degree at most d there exists a SONC certificate of nonnegativity of degree at most $n + d$; see Theorem 16 and Corollary 17.
2. If a polynomial f admits a degree d SONC certificate of nonnegativity over an n-variate hypercube, then the polynomial f admits also a *short* degree d SONC certificate that includes at most $n^{O(d)}$ nonnegative circuit polynomials; see Theorem 18.

Furthermore, we show some structural properties of SONCs:

1. We give a simple, constructive example showing that the SONC cone is not closed under multiplication. Subsequently we use this construction to show that the SONC cone is neither closed under taking affine transformations of variables, see Lemma 8 and Corollary 9 and the discussion afterwards.
2. We address an open problem raised in [19] asking whether the Schmüdgen-like Positivstellensatz for SONCs (Theorem 6) can be improved to an equivalent of Putinar's Positivstellensatz [50]. We answer this question negatively by showing an explicit hypercube optimization example, which provably does not admit a Putinar representation for SONCs; see Theorem 19 and the discussion afterwards.

Our article is organized as follows: In Section 2 we introduce the necessary background from theoretical computer sciences and about SONCs. In Section 3 we show that the SONC cone is closed neither under multiplication nor under affine transformations. In Section 4 we provide our two main results regarding the degree bounds for SONC certificates over the hypercube. In Section 5 we prove the non-existence of an equivalent of Putinar's Positivstellensatz for SONCs and discuss this result.

2 Preliminaries

In this section we collect basic notions and statements on sums of nonnegative circuit polynomials (SONC).

Throughout the paper, we use bold letters for vectors, e.g., $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{R}^n$. Let $\mathbb{N}^* = \mathbb{N} \setminus \{\mathbf{0}\}$ and $\mathbb{R}_{\geq 0}$ ($\mathbb{R}_{>0}$) be the set of nonnegative (positive) real numbers. Furthermore let $\mathbb{R}[\mathbf{x}] = \mathbb{R}[x_1, \dots, x_n]$ be the ring of real n-variate polynomials and the set of all n-variate polynomials of degree less than or equal to $2d$ is denoted by $\mathbb{R}[\mathbf{x}]_{n,2d}$. We denote by $[n]$ the set $\{1, \dots, n\}$ and the sum of binomial coefficients $\sum_{k=0}^{d} \binom{n}{k}$ is abbreviated by $\binom{n}{\leq d}$. Let $\mathbf{e_1}, \dots, \mathbf{e_n}$ denote the canonical basis vectors in $\mathbb{R}^n$.

2.1 Sums of Nonnegative Circuit Polynomials

Let $A \subset \mathbb{N}^n$ be a finite set. In what follows, we consider polynomials $f \in \mathbb{R}[\mathbf{x}]$ supported on A. Thus, f is of the form $f(\mathbf{x}) = \sum_{\boldsymbol{\alpha} \in A} f_{\boldsymbol{\alpha}} \mathbf{x}^{\boldsymbol{\alpha}}$ with $f_{\boldsymbol{\alpha}} \in \mathbb{R}$ and $\mathbf{x}^{\boldsymbol{\alpha}} = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$. A lattice point is called *even* if it is in $(2\mathbb{N})^n$ and a term $f_{\boldsymbol{\alpha}} \mathbf{x}^{\boldsymbol{\alpha}}$ is called a *monomial square* if $f_{\boldsymbol{\alpha}} > 0$ and α even. We denote by $\mathrm{New}(f) = \mathrm{conv}\{\boldsymbol{\alpha} \in \mathbb{N}^n : f_{\boldsymbol{\alpha}} \neq 0\}$ the Newton polytope of f.

Initially, we introduce the foundation of SONC polynomials, namely *circuit polynomials*; see also [27]:

▶ **Definition 1.** A polynomial $f \in \mathbb{R}[\mathbf{x}]$ is called a *circuit polynomial* if it is of the form

$$f(\mathbf{x}) \;:=\; f_{\boldsymbol{\beta}}\mathbf{x}^{\boldsymbol{\beta}} + \sum_{j=0}^{r} f_{\boldsymbol{\alpha}(j)}\mathbf{x}^{\boldsymbol{\alpha}(j)}, \tag{2.1}$$

with $r \leq n$, exponents $\boldsymbol{\alpha}(j)$, $\boldsymbol{\beta} \in A$, and coefficients $f_{\boldsymbol{\alpha}(j)} \in \mathbb{R}_{>0}$, $f_{\boldsymbol{\beta}} \in \mathbb{R}$, such that the following conditions hold:

(C1) $\mathrm{New}(f)$ is a simplex with all even vertices $\boldsymbol{\alpha}(0), \boldsymbol{\alpha}(1), \ldots, \boldsymbol{\alpha}(r) \in \mathbb{Z}^n$.

(C2) The exponent $\boldsymbol{\beta}$ is in the strict interior of $\mathrm{New}(f)$. Hence, there exist unique *barycentric coordinates* λ_j relative to the vertices $\boldsymbol{\alpha}(j)$ with $j = 0, \ldots, r$ satisfying

$$\boldsymbol{\beta} \;=\; \sum_{j=0}^{r} \lambda_j \boldsymbol{\alpha}(j) \;\text{ with }\; \lambda_j \;>\; 0 \;\text{ and }\; \sum_{j=0}^{r} \lambda_j \;=\; 1.$$

We call the terms $f_{\boldsymbol{\alpha}(0)}\mathbf{x}^{\boldsymbol{\alpha}(0)}, \ldots, f_{\boldsymbol{\alpha}(r)}\mathbf{x}^{\boldsymbol{\alpha}(r)}$ the *outer terms* and $f_{\boldsymbol{\beta}}\mathbf{x}^{\boldsymbol{\beta}}$ the *inner term* of f.

For every circuit polynomial we define the corresponding *circuit number* as

$$\Theta_f \;:=\; \prod_{j=0}^{r} \left(\frac{f_{\boldsymbol{\alpha}(j)}}{\lambda_j}\right)^{\lambda_j}. \tag{2.2}$$

Note that the name of these polynomials is motivated by the fact that their support set forms a *circuit*, i.e. a minimal affine dependent set, see e.g. [47]. The first fundamental statement about circuit polynomials is that its nonnegativity is determined by its circuit number Θ_f and f_β entirely:

▶ **Theorem 2** ([27], Theorem 3.8)**.** *Let f be a circuit polynomial with inner term $f_{\boldsymbol{\beta}}\mathbf{x}^{\boldsymbol{\beta}}$ and let Θ_f be the corresponding circuit number, as defined in* (2.2)*. Then the following statements are equivalent:*

1. *f is nonnegative.*
2. *$|f_{\boldsymbol{\beta}}| \leq \Theta_f$ and $\boldsymbol{\beta} \notin (2\mathbb{N})^n$ or $f_{\boldsymbol{\beta}} \geq -\Theta_f$ and $\boldsymbol{\beta} \in (2\mathbb{N})^n$.*

We illustrate the previous definition and theorem by an example:

▶ **Example 3.** The *Motzkin polynomial* [44] is given by

$$M(x_1, x_2) \;:=\; x_1^4x_2^2 + x_1^2x_2^4 - 3x_1^2x_2^2 + 1.$$

It is a circuit polynomial since $\mathrm{New}(f) = \{(4,2), (2,4), (0,0)\}$, and $\boldsymbol{\beta} = (2,2)$ with $\lambda_0, \lambda_1, \lambda_2 = 1/3$. We have $|f_{\boldsymbol{\beta}}| = 3$ and compute $\Theta_f \;=\; \sqrt[3]{\left(\frac{1}{1/3}\right)^3} \;=\; 3$. Hence, we can conclude that $M(x_1, x_2)$ is nonnegative by Theorem 2.

▶ **Definition 4.** We define for every $n, d \in \mathbb{N}^*$ the set of *sums of nonnegative circuit polynomials* (SONC) in n variables of degree $2d$ as

$$C_{n,2d} \;:=\; \left\{ f \in \mathbb{R}[\mathbf{x}]_{n,2d} \;:\; f = \sum_{i=1}^{k} p_i,\; p_i \text{ is a nonnegative circuit polynomial},\; k \in \mathbb{N}^* \right\}$$

Note that the degree is attained at the outer terms and hence it is even.

We denote by SONC both the set of SONC polynomials and the property of a polynomial to be a sum of nonnegative circuit polynomials.

In what follows let $P_{n,2d}$ be the cone of nonnegative n-variate polynomials of degree at most $2d$ and $\Sigma_{n,2d}$ be the corresponding cone of sums of squares respectively. An important observation is, that SONC polynomials form a convex cone independent of the SOS cone:

▶ **Theorem 5** ([27], Proposition 7.2). *$C_{n,2d}$ is a convex cone satisfying:*

1. *$C_{n,2d} \subseteq P_{n,2d}$ for all $n, d \in \mathbb{N}^*$,*
2. *$C_{n,2d} \subseteq \Sigma_{n,2d}$ if and only if $(n, 2d) \in \{(1, 2d), (n, 2), (2, 4)\}$,*
3. *$\Sigma_{n,2d} \not\subseteq C_{n,2d}$ for all $(n, 2d)$ with $2d \geq 6$.*

For further details about the SONC cone see [18, 19, 27].

2.2 SONC certificates over a Constrained Set

In [19, Theorem 4.8], Iliman, the first, and the third author showed that for an arbitrary real polynomial which is strictly positive on a compact, basic closed semialgebraic set K there exists a SONC certificate of nonnegativity. Hereinafter we recall this result.

We assume that K is given by polynomial inequalities $g_i(\mathbf{x}) \geq 0$ for $i = 1, \ldots, s$ and is compact. For technical reason we add $2n$ redundant box constraints $l_j(\mathbf{x}) := N \pm x_j \geq 0$ for some sufficiently large $N \in \mathbb{N}$, which always exists due to our assumption of compactness of K; see [19] for further details. Hence, we have

$$K \;:=\; \{\mathbf{x} \in \mathbb{R}^n : g_i(\mathbf{x}) \geq 0 \text{ for } i \in [s] \text{ and } l_j(\mathbf{x}) \geq 0 \text{ for } j \in [2n]\}. \tag{2.3}$$

In what follows we consider polynomials $H^{(q)}(\mathbf{x})$ defined as products of at most $q \in \mathbb{N}^*$ of the polynomials g_i, l_j and 1, i.e.,

$$H^{(q)}(\mathbf{x}) \;:=\; \prod_{k=1}^{q} h_k(\mathbf{x}), \tag{2.4}$$

where $h_k \in \{1, g_1, \ldots, g_s, l_1, \ldots, l_{2n}\}$. Now we can state:

▶ **Theorem 6** ([19], Theorem 4.8). *Let $f, g_1, \ldots, g_s \in \mathbb{R}[\mathbf{x}]$ be real polynomials and K be a compact, basic closed semialgebraic set as in (2.3). If $f > 0$ on K then there exist $d, q \in \mathbb{N}^*$ such that we have an explicit representation of f of the form:*

$$f(\mathbf{x}) \;=\; \sum_{\text{finite}} s(\mathbf{x}) H^{(q)}(\mathbf{x}),$$

where the $s(\mathbf{x})$ are contained in $C_{n,2d}$ and every $H^{(q)}(\mathbf{x})$ is a product as in (2.4).

The central object of interest is the smallest value of d and q that allows f a decomposition as in Theorem 6. This motivates the following definition of a *degree d SONC certificate*.

▶ **Definition 7.** Let $f \in \mathbb{R}[\mathbf{x}]$ such that f is positive on the set K given in (2.3). Then f has a *degree d SONC certificate* if it admits for some $q \in \mathbb{N}^*$ the following decomposition:

$$f(\mathbf{x}) \;=\; \sum_{\text{finite}} s(\mathbf{x}) H^{(q)}(\mathbf{x}),$$

for $s(\mathbf{x})$ SONCs, the $H^{(q)}(\mathbf{x})$ products as in (2.4), and $\deg\left(\sum s(\mathbf{x}) H^{(q)}(\mathbf{x})\right) \;\leq\; d$.

For a given set $A \subseteq \mathbb{N}^n$, searching through the space of degree d certificates can be computed via a *relative entropy program (REP)* [19] of size $n^{O(d)}$. REPs are convex optimization programs which are slightly more general than geometric programs but still efficiently solvable with interior point methods; see e.g. [10, 46] for more details.

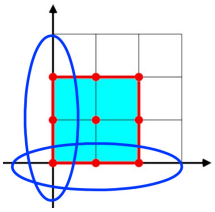


Figure 1 The Newton polytope and the support set of $r(x_1, x_k)$ with the supports of p_1 and p_2 in blue ovals.

3 Properties of the SONC cone

In this section we show that the SONC cone is neither closed under multiplication nor under affine transformations. First, we give a constructive proof for the fact that the SONC cone is not closed under multiplication, which is simpler than the initial proof of this fact in [19, Lemma 4.1]. Second, we use our construction to show that the SONC cone is not closed under affine transformation of variables.

▶ **Lemma 8.** *For every $d \geq 2$, $n \in \mathbb{N}^*$ the SONC cone $C_{n,d}$ is not closed under multiplication in the following sense: if $p_1, p_2 \in C_{n,d}$, then $p_1 \cdot p_2 \notin C_{n,2d}$ in general.*

Proof. For every $d = 2n$, $n \in \mathbb{N}^*$ we construct two SONC polynomials p_1, $p_2 \in C_{n,d}$ such that the product $p_1 p_2$ is an n variate, degree $2d$ polynomial that is not inside $C_{n,2d}$.

Let $n = 2$. We construct the following two polynomials $p_1,\ p_2 \in \mathbb{R}[x_1, x_2]$:

$$p_1(x_1, x_2) \ := \ (1 - x_1)^2, \qquad p_2(x_1, x_2) \ := \ (1 - x_2)^2.$$

First, observe that p_1, p_2 are nonnegative circuit polynomials, since, in both cases, $\lambda_1 = \lambda_2 = 1/2$, $f_{\boldsymbol{\alpha}(1)} = f_{\boldsymbol{\alpha}(2)} = 1$, and $f_{\boldsymbol{\beta}} = -2$, thus $2 = \Theta_f \geq |f_{\boldsymbol{\beta}}|$.

Now consider the polynomial $r(x_1, x_2) = ((1 - x_1)(1 - x_2))^2$. We show that this polynomial, even though it is nonnegative, is not a SONC polynomial. Note that $r(x_1, x_2) = 1 - 2x_1 - 2x_2 + 4x_1x_2 + x_1^2 + x_2^2 - 2x_1^2x_2 - 2x_1x_2^2 + x_1^2x_2^2$; the support of r is shown in Figure 1. Assume that $r \in C_{2,4}$, i.e., r has a SONC decomposition. This implies that the term $-2x_1$ has to be an inner term of some nonnegative circuit polynomial r_1 in this representation. Such a circuit polynomial necessarily has the terms 1 and x_1^2 as outer terms, that is, $r_1(x_1) = p_1(x_1, x_2) = 1 + x_1^2 - 2x_1$. Since $\Theta_{r_1} = 2$ the polynomial r_1 is indeed nonnegative and, in addition, we cannot choose a smaller constant term and preserve nonnegativity without simultaneously increasing the coefficient x_1^2. Next, also the term $-2x_2$ has to be an inner term of SONC r_2. Since this term again is on the boundary of New(r) the only option for such an r_2 is: $r_2(x_2) = p_2(x_1, x_2) = 1 + x_2^2 - 2x_2$. However, the term 1 has been already used in r_1, which leads to a contradiction, i.e., $r \notin C_{2,4}$. Since $C_{n,2d} \subseteq C_{n+1,2d}$, the general statement follows. ◀

Hereinafter we show another operation, which behaves differently for SONC than it does for SOS: Similarly as for multiplications, affine transformations also do not preserve the SONC structure. This observation is important for possible degree bounds on SONC certificates, when considering optimization problems over distinct descriptions of the hypercube.

▶ **Corollary 9.** *For every $d \geq 4$, $n \in \mathbb{N}^*$ the SONC cone $C_{n,d}$ is not closed under affine transformation of variables.*

Proof. Consider the polynomial $f(x_1, x_2) = x_1^2 x_2^2$. Clearly, the polynomial f is a nonnegative circuit polynomial since it is a monomial square, hence $u \in C_{n,d}$. Now consider the following affine transformation of the variables x_1 and x_2: $x_1 \to 1 - x_1$, $x_2 \to 1 - x_2$. After applying the transformation the polynomial f equals the polynomial $p_1 p_2$ from the proof of Lemma 8 and thus is not inside $C_{n,d}$. ◀

Corollary 9, from optimization perspective, implies that problem formulations obtained by applying affine transformations of variables can lead to problems of different tractability when using the SONC method. This means, on the one hand, that a choice of representation has to be done carefully, which makes the process of algorithm design more demanding. On the other hand, even a small change of representation might allow to find a SONC certificate or simplify an existing one. Note that whatever affine transformation of variables is applied to the Motzkin polynomial it *never* has a SOS certificate over reals, as the SOS cone is closed under affine transformations. The affine closure of the SONC cone, however, strictly contains the SONC cone and still yields a certificate of nonnegativity. In this sense, Corollary 9 motivates the following future research question:

Find an efficient algorithm to determine whether an affine transformation of a given polynomial f admits a SONC representation.

4 An Upper Bound on the Degree of SONC Certificates over the Hypercube

In this section we prove that every n-variate polynomial which is nonnegative over the boolean hypercube has a degree n SONC certificate. Moreover, if the hypercube is additionally constrained with some polynomials of degree at most d, then the nonnegative polynomial over such a set has degree $n + d$ SONC certificate. Motivated by the Corollary 9 and the discussion afterwards, we show this fact for *all* affine transformations of the 0/1 hypercube, that is for hypercubes $\{a_i, b_i\}^n$.

Formally, we consider the following setting: We investigate real multivariate polynomials in $\mathbb{R}[\mathbf{x}]$. For $j \in [n]$, and $a_j, b_j \in \mathbb{R}$, such that $a_j < b_j$ let

$$g_j(\mathbf{x}) \;:=\; (x_j - a_j)(x_j - b_j)$$

be a *quadratic polynomial with two distinct real roots*. Let $\mathcal{H} \subset \mathbb{R}^n$ denote the *n-dimensional hypercube* given by $\prod_{j=1}^n \{a_j, b_j\}$. Moreover, let

$$\mathcal{P} \;:=\; \{p_1, \ldots, p_m :\; p_i \in \mathbb{R}[\mathbf{x}],\; i \in [m]\}$$

be a set of polynomials, which we consider as constraints $p_i(\mathbf{x}) \geq 0$ with $\deg(p_i(\mathbf{x})) \leq d$ for all $i \in [n]$ as follows. We define

$$\mathcal{H}_{\mathcal{P}} \;:=\; \{\mathbf{x} \in \mathbb{R}^n :\; g_j(\mathbf{x}) = 0,\; j \in [n],\; p(\mathbf{x}) \geq 0,\; p \in \mathcal{P}\}$$

as the *n-dimensional hypercube $\mathcal{H}$ constrained by polynomial inequalities given by $\mathcal{P}$*. Throughout the paper we assume that $|\mathcal{P}| = \text{poly}(n)$, i.e. the size of the constraint set $\mathcal{P}$ is polynomial in n. This is usually the case, since otherwise the problem gets less tractable from the optimization point of view.

As a first step, we introduce a *Kronecker delta* function:

▶ **Definition 10.** For every $\mathbf{v} \in \mathcal{H}$ the function

$$\delta_{\mathbf{v}}(\mathbf{x}) \;:=\; \prod_{j \in [n]:\; v_j = a_j} \left(\frac{-x_j + b_j}{b_j - a_j}\right) \cdot \prod_{j \in [n]:\; v_j = b_j} \left(\frac{x_j - a_j}{b_j - a_j}\right) \tag{4.1}$$

is called the *Kronecker delta (function)* of the vector $\mathbf{v}$.

Next we justify the term "Kronecker delta", we show that for every $\mathbf{v} \in \mathcal{H}$ the function $\delta_{\mathbf{v}}(\mathbf{x})$ takes the value zero for all $\mathbf{x} \in \mathcal{H}$ except for $\mathbf{x} = \mathbf{v}$ where it takes the value one.

▶ **Lemma 11.** *For every $\mathbf{v} \in \mathcal{H}$ it holds that:*

$$\delta_{\mathbf{v}}(\mathbf{x}) \;=\; \begin{cases} 0, & \text{for every } \mathbf{x} \in \mathcal{H} \setminus \{\mathbf{v}\}, \\ 1, & \text{for } \mathbf{x} = \mathbf{v}. \end{cases}$$

Proof. On the one hand, if $\mathbf{x} \in \mathcal{H} \setminus \{\mathbf{v}\}$, then there exists an index k such that $\mathbf{x}_k \neq \mathbf{v}_k$. This implies that there exists at least one multiplicative factor in $\delta_{\mathbf{v}}$ which attains the value zero due to (4.1). On the other hand if $\mathbf{x} = \mathbf{v}$ then we have

$$\delta_{\mathbf{v}}(\mathbf{x}) \;=\; \prod_{j\in[n]:\ v_j=a_j} \left(\frac{-a_j+b_j}{b_j-a_j}\right) \prod_{j\in[n]:\ v_j=b_j} \left(\frac{b_j-a_j}{b_j-a_j}\right) \;=\; 1. \qquad \blacktriangleleft$$

The main result of this section is the following theorem.

▶ **Theorem 12.** *Let $f(\mathbf{x}) \in \mathbb{R}[\mathbf{x}]_{n,n}$. Then $f(\mathbf{x}) \geq 0$ for every $\mathbf{x} \in \mathcal{H}_{\mathcal{P}}$ if and only if f has the following representation:*

$$f(\mathbf{x}) \;=\; \sum_{\mathbf{v}\in\mathcal{H}_{\mathcal{P}}} c_{\mathbf{v}}\delta_{\mathbf{v}}(\mathbf{x}) + \sum_{\mathbf{v}\in\mathcal{H}\setminus\mathcal{H}_{\mathcal{P}}} c_{\mathbf{v}}\delta_{\mathbf{v}}(\mathbf{x})p_{\mathbf{v}}(\mathbf{x}) + \sum_{j=1}^{n} s_j(\mathbf{x})g_j(\mathbf{x}) + \sum_{j=1}^{n} s_{n+j}(\mathbf{x})(-g_j(\mathbf{x})), \quad (4.2)$$

where $s_1, \ldots, s_{2n} \in C_{n,n-2}$, $c_{\mathbf{v}} \in \mathbb{R}_{\geq 0}$ and $p_{\mathbf{v}} \in \mathcal{P}$.

Since we are interested in optimization on the boolean hypercube $\mathcal{H}$, we assume without loss of generality that the polynomial f considered in Theorem 12 has degree at most n. Otherwise, one can efficiently reduce the degree of f by applying iteratively the polynomial division with respect to polynomials g_j with $j \in [n]$. The remainder of the division process is a polynomial with degree at most n that agrees with f on all the vertices of $\mathcal{H}$.

We begin with proving the easy direction of the equivalence stated in Theorem 12.

▶ **Lemma 13.** *If f admits a decomposition* (4.2)*, then $f(\mathbf{x})$ is nonnegative for all $\mathbf{x} \in \mathcal{H}_{\mathcal{P}}$.*

Proof. The coefficients $c_{\mathbf{v}}$ are nonnegative, all $s_j(\mathbf{x})$ are SONC and hence nonnegative on $\mathbb{R}^n$. We have $\pm g_j(\mathbf{x}) \geq 0$ for all $\mathbf{x} \in \mathcal{H}$, and for all choices of $\mathbf{v} \in \mathcal{H}$ we have $p_{\mathbf{v}}(\mathbf{x}) \geq 0$ for all $\mathbf{x} \in \mathcal{H}_{\mathcal{P}}$, and $\delta_{\mathbf{v}}(\mathbf{x}) \in \{0, 1\}$ for all $\mathbf{x} \in \mathcal{H}$. Thus, the right hand side of (4.2) is a sum of positive terms for all $\mathbf{x} \in \mathcal{H}_{\mathcal{P}}$. ◀

We postpone the rest of the proof of Theorem 12 to the end of the section. Now, we state a result about the presentation of the Kronecker delta function $\delta_{\mathbf{v}}$. In what follows let K be the basic closed semialgebraic set defined by $g_1, \ldots, g_n$ and $l_1, \ldots, l_{2n}$ as in (2.3).

▶ **Lemma 14.** *For every $\mathbf{v} \in \mathcal{H}$ the Kronecker delta function can be written as*

$$\delta_{\mathbf{v}} \;=\; \sum_{j=1}^{2^n} s_j H_j^{(n)},$$

for $s_1, \ldots, s_{2^n} \in \mathbb{R}_{\geq 0}$ and every $H_j^{(n)}$ given as in (2.4) *with $q = n$.*

Proof. First note that the function $\delta_{\mathbf{v}}$ can be rewritten as

$$\delta_{\mathbf{v}}(\mathbf{x}) \;=\; \prod_{j=1}^{2n} \frac{1}{b_j-a_j} \prod_{j\in[n]:\ v_j=a_j} (-x_j+b_j) \prod_{j\in[n]:\ v_j=b_j} (x_j-a_j)\,,$$

where $\prod_{j=1}^{n} \frac{1}{b_j-a_j} \in \mathbb{R}_{\geq 0}$. Now, the proof follows just by noting that for every $j \in [n]$ both inequalities $-x_j + b_j \geq 0$ and $x_j - a_j \geq 0$ are in K. ◀

The following statement is well-known in similar variations; see e.g. [5, Lemma 2.2 and its proof]. For clarity, we provide an own proof in the appendix.

▶ **Proposition 15.** *Let $f \in \mathbb{R}[\mathbf{x}]_{n,2d}$ be a polynomial vanishing on $\mathcal{H}$. Then $f = \sum_{j=1}^{n} p_j g_j$ for some polynomials $p_j \in \mathbb{R}[\mathbf{x}]_{n,2d-2}$.*

Proof. Let $\mathcal{J} := \langle g_1, \dots, g_n \rangle$ be the ideal generated by the g_j's. Let $\mathcal{V}(\mathcal{J})$ denote the affine variety corresponding to $\mathcal{J}$, $\mathcal{I}(\mathcal{V}(\mathcal{J}))$ denote its radical ideal, and let $\mathcal{I}(\mathcal{H})$ denote the ideal of $\mathcal{H}$. It follows from $\prod_{j=1}^{n} g_j \in \mathcal{J}$ that $\mathcal{V}(\mathcal{J}) \subseteq \mathcal{H}$ and hence $\mathcal{I}(\mathcal{H}) \subseteq \mathcal{I}(\mathcal{V}(\mathcal{J})) = \mathcal{J}$. The last equality holds since $\mathcal{J}$ itself is a radical ideal. This results from Seidenberg's Lemma; see [30, Proposition 3.7.15] by means of the following observations. The affine variety $\mathcal{V}(\mathcal{J})$ consists exactly of the points defining $\mathcal{H}$, therefore we know that $\mathcal{J}$ is a zero-dimensional ideal. Furthermore, for every $j \in [n]$ the polynomials g_j satisfy $g_j \in \mathcal{J} \cap K[x_j]$ and $\gcd(g_j, g_j') = 1$. Thus, every $f \in \mathcal{I}(\mathcal{H})$ is of the form $f = \sum_{j=1}^{n} p_j g_j$.

Moreover $G := \{g_1, \dots, g_n\}$ is a Gröbner basis for $\mathcal{J}$ with respect to the graded lexicographic order $\prec_{\text{glex}}$. This follows from Buchberger's Criterion, which says that G is a Gröbner basis for $\mathcal{J}$ if and only if for all pairs $i \neq j$ the remainder on the division of the S-polynomials $S(g_i, g_j)$ by G with respect to $\prec_{\text{glex}}$ is zero. Consider an arbitrary pair g_i, g_j with $i > j$. Then the corresponding S-polynomial is given by

$$S(g_i, g_j) \;=\; (a_j + b_j)x_i^2 x_j - (a_i + b_i)x_i x_j^2 - a_j b_j x_i^2 + a_i b_i x_j^2 \,.$$

Applying polynomial division with respect to $\prec_{\text{glex}}$ yields the remainder 0 and hence G is a Gröbner basis for $\mathcal{J}$ with respect to $\prec_{\text{glex}}$. Therefore, we conclude that if $f \in \mathbb{R}[\mathbf{x}]_{n,2d}$, then $\deg(p_j) \leq 2d - 2$. ◀

For an introduction to Gröbner bases see for example [15].

▶ **Theorem 16.** *Let $d \in \mathbb{N}$ and $f \in \mathbb{R}[\mathbf{x}]_{n,2d+2}$ such that f vanishes on $\mathcal{H}$. Then there exist $s_1, \dots, s_{2n} \in C_{n,2d}$ such that $f = \sum_{j=1}^{n} s_j g_j + \sum_{j=1}^{n} s_{n+j}(-g_j)$.*

Proof. By Proposition 15 we know that $f = \sum_{j=1}^{n} p_j g_j$ for some polynomials p_j of degree $\leq 2d$. Hence, it is sufficient to show that every single term $p_j g_j$ is of the form $\sum_{j=1}^{n} s_j g_j - \sum_{j=1}^{n} s_{n+j} g_j$ for some $s_1, \dots, s_{2n} \in C_{n,2d}$. Let $p_j = \sum_{i=1}^{\ell} a_{ji} m_{ji}$ where every $a_{ji} \in \mathbb{R}$ and every m_{ji} is a single monomial. We show that $p_j g_j$ has the desired form by investigating an arbitrary individual term $a_{ji} m_{ji} g_j$.

Case 1: Assume the exponent of m_{ji} is contained in $(2\mathbb{N})^n$. If $a_{ji} m_{ji}$ is a monomial square, then $a_{ji} m_{ji}$ is a circuit polynomial. If $a_{ji} < 0$, then $-a_{ji} m_{ji}$ is a monomial square. In both cases we obtain a representation $s_{ji}(\pm g_{ji})$, where $s_{ji} \in C_{n,2d}$.

Case 2: Assume the the exponent $\boldsymbol{\beta}$ of m_{ji} contains odd numbers. Without loss of generality, assume that $\boldsymbol{\beta} = (\beta_1, \dots, \beta_k, \beta_{k+1}, \dots, \beta_n)$ such that the first k entries are odd and the remaining $n - k$ entries are even. We construct a SONC polynomial $s_{ji} = a_{\boldsymbol{\alpha}(1)} \mathbf{x}^{\boldsymbol{\alpha}(1)} + a_{\boldsymbol{\alpha}(2)} \mathbf{x}^{\boldsymbol{\alpha}(2)} + a_{ji} \mathbf{x}^{\boldsymbol{\beta}}$ such that

$$\boldsymbol{\alpha}(1) \;=\; \boldsymbol{\beta} + \sum_{j=1}^{\lceil k/2 \rceil} \mathbf{e}_j - \sum_{j=\lceil k/2 \rceil + 1}^{k} \mathbf{e}_j, \qquad \boldsymbol{\alpha}(2) \;=\; \boldsymbol{\beta} - \sum_{j=1}^{\lceil k/2 \rceil} \mathbf{e}_j + \sum_{j=\lceil k/2 \rceil + 1}^{k} \mathbf{e}_j, \tag{4.3}$$

$$|a_{ji}| \;\leq\; \sqrt{2 a_{\boldsymbol{\alpha}(1)} a_{\boldsymbol{\alpha}(2)}}. \tag{4.4}$$

By the construction (4.3) $\boldsymbol{\alpha}(1), \boldsymbol{\alpha}(2) \in (2\mathbb{N})^n$ and $\boldsymbol{\beta} = 1/2(\boldsymbol{\alpha}(1) + \boldsymbol{\alpha}(2))$. Thus, s_{ji} is a circuit polynomial and by (4.4) the coefficients $a_{\boldsymbol{\alpha}(1)}, a_{\boldsymbol{\alpha}(2)}$ are chosen large enough such that $|a_{ji}|$ is bound by the circuit number $\sqrt{2 a_{\boldsymbol{\alpha}(1)} a_{\boldsymbol{\alpha}(2)}}$ corresponding to s_{ji}. Thus, s_{ji} is nonnegative by [27, Theorem 1.1]. Thus, we obtain

$$a_{ji} m_{ji} g_j \;=\; s_{ji} g_j + (a_{\boldsymbol{\alpha}(1)} \mathbf{x}^{\boldsymbol{\alpha}(1)} + a_{\boldsymbol{\alpha}(2)} \mathbf{x}^{\boldsymbol{\alpha}(2)})(-g_j),$$

where s_{ji}, $a_{\boldsymbol{\alpha}(1)} \mathbf{x}^{\boldsymbol{\alpha}(1)}$, and $a_{\boldsymbol{\alpha}(2)} \mathbf{x}^{\boldsymbol{\alpha}(2)}$ are nonnegative circuit polynomials.

Degree: All involved nonnegative circuit polynomials are of degree at most $2d$. In Case 1 this follows by construction. In Case 2 we have for the circuit polynomial s_{ji} that $\deg(\boldsymbol{\alpha}(1)), \deg(\boldsymbol{\alpha}(2)) = \deg(\boldsymbol{\beta})$ if k is even, and $\deg(\boldsymbol{\alpha}(1)) = \deg(\boldsymbol{\beta}) + 1$, $\deg(\boldsymbol{\alpha}(2)) = \deg(\boldsymbol{\beta})$ if k is odd. Since $\boldsymbol{\beta}$ is an exponent of the polynomial f, we know that $\deg(\boldsymbol{\beta}) \leq 2d$. If k is odd, however, then

$$\deg(\boldsymbol{\beta}) \;=\; \sum_{j=1}^{k} \underbrace{\beta_j}_{\text{odd number}} + \sum_{j=k+1}^{n} \underbrace{\beta_j}_{\text{even number}} \;,$$

i.e., $\deg(\boldsymbol{\beta})$ is a sum of k many odd numbers, with k being odd, plus a sum of even numbers. Thus, $\deg(\boldsymbol{\beta})$ has to be an odd number and hence $\deg(\boldsymbol{\beta}) < 2d$. Therefore, all degrees of terms in s_{ji} are bounded by $2d$ and thus $s_{ji} \in C_{n,2d}$.

Conclusion: We have that

$$f \;=\; \sum_{j=1}^{n} p_j g_j \;=\; \sum_{j=1}^{n} \sum_{i=1}^{\ell_j} a_{ji} m_{ji} g_j \;=\; \sum_{j=1}^{n} \sum_{i=1}^{\ell_j} s_{ji} g_j.$$

By Cases 1 and 2 and the degree argument, we have $s_{ji} \in C_{n,2d}$ for every i, j and by defining $s_j = \sum_{i=1}^{\ell_j} s_{ji} \in C_{n,2d}$ we obtain the desired representation of f. ◀

4.1 Proof of Theorem 12

In this section we combine the results of this section and finish the proof of Theorem 12.

Due to Lemma 13, it remains to show that $f(\mathbf{x})$ admits a decomposition of the form (4.2) with $\mathcal{H}_\mathcal{P} = \mathcal{H}$ if $f(\mathbf{x}) \geq 0$ for every $\mathbf{x} \in \mathcal{H}$.

Hence, when restricted to the hypercube $\mathcal{H}$, the polynomial f can be represented as:

$$\begin{aligned} f(\mathbf{x}) &= f(\mathbf{x}) \sum_{\mathbf{v} \in \mathcal{H}_\mathcal{P}} \delta_\mathbf{v}(\mathbf{x}) + f(\mathbf{x}) \sum_{\mathbf{v} \in \mathcal{H} \setminus \mathcal{H}_\mathcal{P}} \delta_\mathbf{v}(\mathbf{x}) \qquad \text{for all } \mathbf{x} \in \mathcal{H} \\ &= \sum_{\mathbf{v} \in \mathcal{H}_\mathcal{P}} \delta_\mathbf{v}(\mathbf{x}) f(\mathbf{v}) + \sum_{\mathbf{v} \in \mathcal{H} \setminus \mathcal{H}_\mathcal{P}} \delta_\mathbf{v}(\mathbf{x}) f(\mathbf{v}) \qquad \text{for all } \mathbf{x} \in \mathcal{H}, \end{aligned}$$

where the last equality follows by Lemma 11 .

Note that there might exist a vector $\mathbf{v} \in \mathcal{H} \setminus \mathcal{H}_\mathcal{P}$ such that f attains a negative value at $\mathbf{v}$. If $f(\mathbf{v}) < 0$, then let $p_\mathbf{v} \in \mathcal{P}$ be one of the polynomials among the constraints satisfying $p_\mathbf{v}(\mathbf{v}) < 0$. Otherwise, let $p_\mathbf{v} = 1$. Since by Lemma 11 we have $\delta_\mathbf{v}(\mathbf{x}) p_\mathbf{v}(\mathbf{x}) = \delta_\mathbf{v}(\mathbf{x}) p_\mathbf{v}(\mathbf{v})$ for every $\mathbf{v}, \mathbf{x} \in \mathcal{H}$, we can now write:

$$f(\mathbf{x}) \;=\; \sum_{\mathbf{v} \in \mathcal{H}_\mathcal{P}} \delta_\mathbf{v}(\mathbf{x}) f(\mathbf{v}) + \sum_{\mathbf{v} \in \mathcal{H} \setminus \mathcal{H}_\mathcal{P}} \delta_\mathbf{v}(\mathbf{x}) p_\mathbf{v}(\mathbf{x}) \frac{f(\mathbf{v})}{p_\mathbf{v}(\mathbf{v})} \qquad \text{for all } \mathbf{x} \in \mathcal{H}.$$

Thus, the polynomial $f(\mathbf{x}) - \sum_{\mathbf{v} \in \mathcal{H}_\mathcal{P}} \delta_\mathbf{v}(\mathbf{x}) f(\mathbf{v}) - \sum_{\mathbf{v} \in \mathcal{H} \setminus \mathcal{H}_\mathcal{P}} \delta_\mathbf{v}(\mathbf{x}) p_\mathbf{v}(\mathbf{x}) \frac{f(\mathbf{v})}{p_\mathbf{v}(\mathbf{v})}$ has degree at most $n + d$ and vanishes on $\mathcal{H}$. By Theorem 16 we finally get

$$f(\mathbf{x}) \;=\; \sum_{j=1}^{n} s_j(\mathbf{x}) g_j(\mathbf{x}) + \sum_{j=1}^{n} s_{n+j}(\mathbf{x})(-g_j(\mathbf{x})) + \sum_{\mathbf{v} \in \mathcal{H}_\mathcal{P}} \delta_\mathbf{v}(\mathbf{x}) f(\mathbf{v}) + \sum_{\mathbf{v} \in \mathcal{H} \setminus \mathcal{H}_\mathcal{P}} \delta_\mathbf{v}(\mathbf{x}) p_\mathbf{v}(\mathbf{x}) \frac{f(\mathbf{v})}{p_\mathbf{v}(\mathbf{v})} \;,$$

for some $s_1, \ldots, s_{2n} \in C_{n,n-2}$ and $p_\mathbf{v} \in \mathcal{P}$. This finishes proof together with Lemma 14. ◀

▶ **Corollary 17.** *For every polynomial f, nonnegative over the boolean hypercube, constrained with polynomial inequalities of degree at most d, there exists a degree $n+d$ SONC certificate.*

Proof. The argument follows directly from Theorem 12 by noting that the right hand side of (4.2) is a SONC certificate of degree $n+d$ (see the Definition 7). ◀

4.2 Degree d SONC Certificates

In this section we show that if a polynomial f admits a degree d SONC certificate, then f also admits a short degree d certificate that involves at most $n^{O(d)}$ terms.

▶ **Theorem 18.** *Let f be an n-variate polynomial, nonnegative on the constrained hypercube $\mathcal{H}_\mathcal{P}$ with $|\mathcal{P}| = \text{poly}(n)$. Assume that there exists a degree d SONC certificate for f, then there exists a degree d SONC certificate for f involving at most $n^{O(d)}$ many nonnegative circuit polynomials.*

Proof. Since there exists a degree d SONC proof of the nonnegativity of f on $\mathcal{H}_\mathcal{P}$ we know that $f(\mathbf{x}) = \sum_j s_j H_j^{(q)}$, where the summation is finite, s_j's are SONCs, and $H_j^{(q)}$'s are product as defined in (2.4).

Step 1: We analyze the terms s_j. Since every s_j is a SONC, there exists a representation

$$s_j = \kappa_j \cdot \sum_{i=1}^{k_j} \mu_{ij} \cdot q_{ij}$$

such that $\kappa_j, \mu_{1j}, \ldots, \mu_{k_j j} \in \mathbb{R}_{>0}$, $\sum_{i=1}^{k_j} \mu_{ij} = 1$, and the q_{ij} are nonnegative circuit polynomials. Since s_j is of degree at most d, we know that $Q_j := \{q_{1j}, \ldots, q_{k_j j}\}$ is contained in $\mathbb{R}_{n,d}[\mathbf{x}]$, which is a real vector space of dimension $\binom{n+d}{d}$. Since s_j/κ_j is a convex combination of the q_{ij}, i.e. in the convex hull of Q_j, and $\dim(Q_j) \leq \binom{n+d}{d}$, applying Carathéodory's Theorem, see e.g. [57], yields that s_j/κ_j can be written as a convex combination of at most $\binom{n+d}{d}+1$ many of the q_{ij}.

Step 2: We analyze the terms $H_j^{(q)}$. By definition of the $\mathcal{H}_\mathcal{P}$ and the terms $H_j^{(q)}$ we have $H_j^{(q)} = g_{j_1} \cdots g_{j_s} \cdot l_{r_1} \cdots l_{r_t} \cdot p_{\ell_1} \cdot p_{\ell_v}$ with $j_1, \ldots, j_s \in [n]$, $r_1, \ldots, r_t \in [2n]$, and $\ell_1, \ldots, \ell_v \in [m]$. Since the maximal degree of $H_j^{(q)}$ is d, the number of different $H_j^{(q)}$'s is bounded from above by $\binom{n+2n+m}{d}$.

Conclusion: In summary, we obtain a representation:

$$f(\mathbf{x}) = \sum_{i=1}^{\binom{n+2n+m}{d}} H_j^{(q)} s_j = \sum_{i=1}^{\binom{n+2n+m}{d}} H_j^{(q)} \kappa_j \sum_{j=1}^{\binom{n+d}{d}+1} \mu_{ij} c_{ij}$$

Since we assume that m can bounded by $\text{poly}(n)$ the total number of summands is $\text{poly}(n)^{O(d)} = n^{O(d)}$, and we found a desired representation with at most $n^{O(d)}$ nonnegative circuit polynomials of degree at most d. ◀

The Theorem 18 states that when searching for a degree d SONC certificate it is enough to restrict to certificates containing at most $n^{O(d)}$ nonnegative circuit polynomials. Moreover, as proved in [19, Theorem 3.2] for a given set $A \subseteq \mathbb{N}^n$, searching through the space of degree d SONC certificates supported on a set A can be computed via a relative entropy program (REP) of size $n^{O(d)}$, see e.g. [19] for more information about REP. However, the above arguments do *not* necessarily imply that the search through the space of degree d SONC certificates can be performed in time $n^{O(d)}$. The difficulty is that one needs to restrict the

configuration space of n-variate degree d SONCs to a subset of order $n^{O(d)}$ to be able to *formulate* the corresponding REP in time $n^{O(d)}$. Since the current proof of Theorem 18 just guarantees the *existence* of a short SONC certificate, it is currently not clear, how to search for a short certificate efficiently. We leave this as an open problem.

5 There Exists No Equivalent to Putinar's Positivstellensatz for SONCs

In this section we address the open problem raised in [19] asking whether the Theorem 6 can be strengthened by requiring $q = 1$. Such a strengthening, for a positive polynomial over some basic closed semialgebraic set, would provide a SONC decomposition equivalent to Putinar's Positivstellensatz for SOS. The advantage of Putinar's Positivstellensatz over Schmüdgen's Positivstellensatz is that for every fixed degree d the cardinality of possible degree d certificates is smaller; for background see e.g., [38, 50] however, asymptotically still in both cases it is $n^{O(d)}$.

We answer this question in a negative way. More precisely, we provide a polynomial f which is strictly positive over the hypercube $\{\pm 1\}^n$ such that there does not exist a SONC decomposition of f for $q = 1$. Moreover, we prove it not only for the most natural choice of the box constraints that is $l_i = 1 \pm x_i$, but for a generic type of box constraints of the form $\ell_i = 1 + c_i \pm x_i$, for $c_i \in \mathbb{R}_{\geq 0}$. We close the section with a short discussion.

Let $\mathcal{H} = \{\pm 1\}^n$ and consider the following family of polynomials parametrized by a natural number a:

$$f_a(\mathbf{x}) \;:=\; (a-1)\prod_{i=1}^{n}\left(\frac{x_i+1}{2}\right)+1.$$

These functions take the value a for a vector $\mathbf{e} = \sum_{i=1}^{n} \mathbf{e}_i$ and the value 1 for every other $\mathbf{x} \in \mathcal{H} \setminus \{\mathbf{e}\}$. We define for every $d \in \mathbb{N}$

$$S_d \;:=\; \left\{ \sum_{\text{finite}} s \cdot h \;:\; s \in C_{n,2d},\; h \in \left\{1, \pm(x_i^2-1), 1+c_i \pm x_i \;:\; i \in [n], c_i \in \mathbb{R}_{\geq 0}\right\} \right\}$$

be the set of polynomials admitting a SONC decomposition over $\mathcal{H}$ given by Theorem 6 for $q = 1$. The main result of this section is the following theorem.

▶ **Theorem 19.** *For every $a > \frac{2^n-1}{2^{n-2}-1}$ we have $f_a \notin S_d$ for all $d \in \mathbb{N}$.*

Before we prove this theorem, we show the following structural results. Note that similar observations were already made for AGIforms by Reznick in [52] using a different notation.

▶ **Lemma 20.** *Every $s(\mathbf{x}) \in C_{n,2d}$ attains at most two different values on $\mathcal{H} = \{\pm 1\}^n$. Moreover, if $s(\mathbf{x})$ attains two different values, then each value is attained for exactly the half of the hypercube vertices.*

Proof. By Definition 1 every nonnegative circuit polynomial is of the form:

$$s(\mathbf{x}) \;=\; \sum_{j=0}^{r} f_{\boldsymbol{\alpha}(j)}\mathbf{x}^{\boldsymbol{\alpha}(j)} + f_{\boldsymbol{\beta}}\mathbf{x}^{\boldsymbol{\beta}}.$$

Note that for $j = 0, \ldots, r$, we have $\boldsymbol{\alpha}(j) \in (2\mathbb{N})^n$. Hence when evaluated over the hypercube $\mathbf{x} \in \mathcal{H} = \{\pm 1\}^n$, $s(\mathbf{x})$ can take only one of at most two different values $\sum_{j=0}^{r} f_{\boldsymbol{\alpha}(j)} \pm f_{\boldsymbol{\beta}}$.

If $s(\mathbf{x})$ attains two different values over $\mathcal{H}$, then there has to exist a non empty subset of variables that have an odd entry in $\boldsymbol{\beta}$. Let $I \subseteq [n]$ be this subset. Then $s(\mathbf{x}) = \sum_{j=0}^{r} f_{\boldsymbol{\alpha}(j)}(\mathbf{x}) - f_{\boldsymbol{\beta}}(\mathbf{x})$, for $\mathbf{x} \in \mathcal{H}$ if and only if $\mathbf{x}$ has an odd number of -1 entries in the set I. The number of such vectors is equal to

$$2^{n-|I|} \sum_{\substack{i=0,\\ i \text{ odd}}}^{|I|} 2^i \;=\; 2^{n-|I|}2^{|I|-1} \;=\; 2^{n-1}. \qquad \blacktriangleleft$$

▶ **Lemma 21.** *Every polynomial $s(\mathbf{x})\ell_i(\mathbf{x})$, with $s \in C_{n,2d}$ and $\ell_i = 1 + c_i \pm x_i$ being a box constraint, attains at most four different values on $\mathcal{H} = \{\pm 1\}^n$. Moreover, each value is attained for at least one forth of the hypercube vertices.*

Proof. By Lemma 20, $s(\mathbf{x})$ attains at most the two values $\left(\sum_{j=0}^{r} f_{\boldsymbol{\alpha}(j)} \pm f_{\boldsymbol{\beta}}\right)$ on $\mathcal{H}$. Similarly, $\ell_i(\mathbf{x})$ attains at most the two values $1 + c_i \pm x_i$ over $\mathcal{H}$. Thus, a polynomial $s(\mathbf{x})\ell_i(\mathbf{x})$ attains at most the four different values $\left(\sum_{j=0}^{r} f_{\boldsymbol{\alpha}(j)} \pm f_{\boldsymbol{\beta}}\right)(1 + c_i \pm x_i)$ on $\mathcal{H}$.

Let I be as in the proof of Lemma 20, i.e., the subset of variables that have an odd entry in $\boldsymbol{\beta}$. If $I = \emptyset$, then the first term $\sum_{j=0}^{r} f_{\boldsymbol{\alpha}(j)} + f_{\boldsymbol{\beta}}$ is constant over the hypercube $\mathcal{H}$, thus $s(\mathbf{x})\ell_i(\mathbf{x})$ takes two different values depending on the i-th entry of the vector. Each value is attained for exactly half of the vectors.

If $I \neq \emptyset$ and $i \notin I$ the claim holds since the value of the first term depends only on the entries in I and the value of the second term depends on the i-th entry. Hence, the polynomial $s(\mathbf{x})\ell_i(\mathbf{x})$ attains four values each on exactly one fourth of $\mathcal{H}$ vectors.

Finally, let $I \neq \emptyset$ and $i \in I$. Partition the hypercube vertices into two sets depending on the i-th entry. Each set has cardinality 2^{n-1}. Consider the set with $x_i = 1$. For the vectors in this set the second term takes a constant value $2 + c$. Over this set the polynomial s takes one of the values $\sum_{j=0}^{r} f_{\boldsymbol{\alpha}(j)}(\mathbf{x}) \pm f_{\boldsymbol{\beta}}(\mathbf{x})$, depending on whether $\mathbf{x}$ has an odd or even number of -1 entries in the set $I \setminus \{-1\}$. In both cases the number of such vectors is equal to

$$2^{n-|I|} \sum_{\substack{i=0,\\ i \text{ odd}}}^{|I|-1} 2^i \;=\; 2^{n-|I|}2^{|I|-2} = 2^{n-2}.$$

The analysis for the case $x_i = -1$ is analogous. $\blacktriangleleft$

Now we can provide the proof of Theorem 19.

Proof of Theorem 19. Assume $f_a \in S_d$ for some $a \in \mathbb{N}$ and $d \in \mathbb{N}$. We prove that a has to be smaller or equal than $\frac{2^n-1}{2^{n-2}-1}$. Since $f_a \in S_d$ we know that

$$f_a(\mathbf{x}) \;=\; s_0(\mathbf{x}) + \sum_{i=1}^{n} s_i(\mathbf{x})\ell_i(\mathbf{x}) + \sum_{j=1}^{n} \tilde{s}_j(\mathbf{x})(x_j^2 - 1) + \tilde{s}_{j+n}(\mathbf{x})(1 - x_j^2)$$

with $s_0, \ldots, s_n, \tilde{s}_1, \ldots, \tilde{s}_{2n} \in C_{n,2d}$. Since $\pm(x_j^2 - 1)$ for $j \in [n]$ vanishes over the hypercube $\mathcal{H}$, for some $s_0,\ s_i \in C_{n,2d}$ we can conclude

$$f_a(\mathbf{x}) \;=\; s_0(\mathbf{x}) + \sum_{i=1}^{n} s_i(\mathbf{x})\ell_i(\mathbf{x}) \qquad \text{for all } \mathbf{x} \in \mathcal{H} \tag{5.1}$$

Let $s_{0,k}$, and $s_{i,j}$ be some nonnegative circuit polynomials such that $s_0 = \sum_k s_{0,k}$, and $s_i = \sum_j s_{i,j}$. Thus, we get

$$\begin{aligned}\sum_{\mathbf{x}\in\mathcal{H}} \left(s_0(\mathbf{x}) + \sum_i s_i(\mathbf{x})\ell_i(\mathbf{x}) \right) &= \sum_k \sum_{\mathbf{x}\in\mathcal{H}} s_{0,k}(\mathbf{x}) + \sum_i \sum_j \sum_{\mathbf{x}\in\mathcal{H}} s_{i,j}(\mathbf{x})\ell_{i,j}(\mathbf{x}) \\ &\geq \sum_k 2^{n-1} s_{0,k}(\mathbf{e}) + \sum_i \sum_j 2^{n-2} s_{i,j}(\mathbf{e})\ell_{i,j}(\mathbf{e}) \\ &\geq 2^{n-2} \left(s_0(\mathbf{e}) + \sum_i s_i(\mathbf{e})\ell_i(\mathbf{e}) \right) = 2^{n-2} a,\end{aligned}$$

where the first inequality comes from Lemma 20 and 21 and the last equality from the fact that $f_a(\mathbf{e}) = a$. On the other hand, by the properties of f_a and the equality (5.1), we know that

$$\sum_{\mathbf{x}\in\mathcal{H}} \left(s_0(\mathbf{x}) + \sum_i s_i(\mathbf{x})\ell_i(\mathbf{x}) \right) = 2^n - 1 + a,$$

which makes the subsequent inequality a necessary requirement for $f_a \in S_d$:

$$a \leq \frac{2^n - 1}{2^{n-2} - 1}. \qquad \blacktriangleleft$$

Note that an easier example of polynomial nonnegative over the set $\mathcal{H}$ exists, that does not attain a SONC decomposition for $q = 1$. Consider a polynomial $\delta_{\mathbf{v}}(\mathbf{x})$ defined in Definition 10 for $x \in \mathbb{R}^n$, for $n \geq 3$. The analysis for this example is easier since the polynomial is zero on all vertices of $\mathcal{H}$ but one, thus by Lemma 21 it is impossible to fit a SONC certificate that matches those values. However, an important fact is that, by Theorem 6 a polynomial to admit a SONC certificate has to necessarily be strictly positive over the given set, which is not the case for $\delta_{\mathbf{v}}(\mathbf{x})$ and the set $\mathcal{H}$.

Speaking from a broader perspective, we interpret Theorem 19 as an indication that the real algebraic structures, which we use to handle sums of squares, do not apply in the same generality to SONCs. We find this not at all surprising from the point of view that in the 19th century Hilbert initially used SOS as a certificate for nonnegativity and many of the algebraic structures in question where developed afterwards with Hilbert's results in mind; see [53] for a historic overview. Our previous work shows that SONCs, in contrast, can, e.g., very well be analyzed with combinatorial methods. We thus see Theorem 19 as further evidence about the very different behavior of SONCs and SOS and as an encouragement to take methods beside the traditional real algebraic ones into account for the successful application of SONCs in the future.

References

1 S. Arora, B. Barak, and D. Steurer. Subexponential algorithms for unique games and related problems. In *FOCS*, pages 563–572, 2010.

2 S. Arora, S. Rao, and U. V. Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2):5:1–5:37, 2009. doi:10.1145/1502793.1502794.

3 B. Barak, S. B. Hopkins, J. A. Kelner, P. Kothari, A. Moitra, and A. Potechin. A nearly tight sum-of-squares lower bound for the planted clique problem. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 428–437, 2016.

4 B. Barak, P. Raghavendra, and D. Steurer. Rounding semidefinite programming hierarchies via global correlation. In *FOCS*, pages 472–481, 2011.

5 B. Barak and D. Steurer. Sum-of-squares proofs and the quest toward optimal algorithms. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:59, 2014.

6 Boaz Barak, Jonathan A. Kelner, and David Steurer. Dictionary learning and tensor decomposition via the sum-of-squares method. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 143–151, 2015.

7 Boaz Barak and Ankur Moitra. Noisy tensor completion via the sum-of-squares hierarchy. In *Proceedings of the 29th Conference on Learning Theory, COLT 2016, New York, USA, June 23-26, 2016*, pages 417–445, 2016.

8 M. H. Bateni, M. Charikar, and V. Guruswami. Maxmin allocation via degree lower-bounded arborescences. In *STOC*, pages 543–552, 2009. doi:10.1145/1536414.1536488.

9 Grigoriy Blekherman. There are significantly more nonegative polynomials than sums of squares. *Israel Journal of Mathematics*, 153(1):355–380, Dec 2006.

10 V. Chandrasekaran and P. Shah. Relative entropy optimization and its applications. *Math. Program.*, 161(1-2):1–32, 2017.

11 K. K. H. Cheung. Computation of the Lasserre ranks of some polytopes. *Math. Oper. Res.*, 32(1):88–94, 2007.

12 E. Chlamtac. Approximation algorithms using hierarchies of semidefinite programming relaxations. In *FOCS*, pages 691–701, 2007.

13 E. Chlamtac and G. Singh. Improved approximation guarantees through higher levels of SDP hierarchies. In *APPROX-RANDOM*, pages 49–62, 2008.

14 E. Chlamtac and M. Tulsiani. Convex relaxations and integrality gaps. In *to appear in Handbook on semidefinite, conic and polynomial optimization*. Springer, 2012.

15 D.A. Cox and J. Little D. O'Shea. *Ideals, varieties, and algorithms*. Undergraduate Texts in Mathematics. Springer, Cham, fourth edition, 2015. An introduction to computational algebraic geometry and commutative algebra.

16 M. Cygan, F. Grandoni, and M. Mastrolilli. How to sell hyperedges: The hypermatching assignment problem. In *SODA*, pages 342–351, 2013.

17 W. F. de la Vega and C. Kenyon-Mathieu. Linear programming relaxations of maxcut. In *SODA*, pages 53–61, 2007.

18 T. de Wolff. Amoebas, nonnegative polynomials and sums of squares supported on circuits. *Oberwolfach Rep.*, 23:53–56, 2015.

19 M. Dressler, S. Iliman, and T. de Wolff. A Positivstellensatz for Sums of Nonnegative Circuit Polynomials. *SIAM J. Appl. Algebra Geom.*, 1(1):536–555, 2017.

20 M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.*, 42(6):1115–1145, 1995.

21 D. Grigoriev. Complexity of positivstellensatz proofs for the knapsack. *Comput. Complexity*, 10(2):139–154, 2001.

22 D. Grigoriev, E. A. Hirsch, and D. V. Pasechnik. Complexity of semi-algebraic proofs. In *STACS*, pages 419–430, 2002.

23 D. Grigoriev and N. Vorobjov. Complexity of null-and positivstellensatz proofs. *Ann. Pure App. Logic*, 113(1-3):153–160, 2001.

24 V. Guruswami and A. K. Sinop. Lasserre hierarchy, higher eigenvalues, and approximation schemes for graph partitioning and quadratic integer programming with psd objectives. In *FOCS*, pages 482–491, 2011.

25 D. Hilbert. Uber die darstellung definiter formen als summe von formen-quadraten. *Annals of Mathematics*, 32:342–350, 1888.

26 Samuel B. Hopkins, Tselil Schramm, Jonathan Shi, and David Steurer. Fast spectral algorithms from sum-of-squares proofs: tensor decomposition and planted sparse vectors. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 178–191, 2016.

27 S. Iliman and T. de Wolff. Amoebas, nonnegative polynomials and sums of squares supported on circuits. *Res. Math. Sci.*, 3:3:9, 2016.

28 Pravesh Kothari, Jacob Steinhardt, and David Steurer. Robust moment estimation and improved clustering via sum of squares. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, 2018.

29 Pravesh K. Kothari, Ryuhei Mori, Ryan O'Donnell, and David Witmer. Sum of squares lower bounds for refuting any CSP. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 132–145, 2017.

30 M. Kreuzer and L. Robbiano. *Computational commutative algebra. 1.* Springer-Verlag, Berlin, 2000.

31 A. Kurpisz, S. Leppänen, and M. Mastrolilli. Sum-of-squares hierarchy lower bounds for symmetric formulations. In *Integer Programming and Combinatorial Optimization - 18th International Conference, IPCO 2016, Liège, Belgium, June 1-3, 2016, Proceedings*, pages 362–374, 2016.

32 A. Kurpisz, S. Leppänen, and M. Mastrolilli. On the hardest problem formulations for the 0/1 lasserre hierarchy. *Math. Oper. Res.*, 42(1):135–143, 2017.

33 A. Kurpisz, S. Leppänen, and M. Mastrolilli. An unbounded sum-of-squares hierarchy integrality gap for a polynomially solvable problem. *Math. Program.*, 166(1-2):1–17, 2017.

34 J.B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM J. Optim.*, 11(3):796–817, 2000/01.

35 M. Laurent. A comparison of the Sherali-Adams, Lovász-Schrijver, and Lasserre relaxations for 0-1 programming. *Math. Oper. Res.*, 28(3):470–496, 2003.

36 M. Laurent. Lower bound for the number of iterations in semidefinite hierarchies for the cut polytope. *Math. Oper. Res.*, 28(4):871–883, 2003.

37 M. Laurent. Sums of squares, moment matrices and optimization over polynomials. In *Emerging applications of algebraic geometry*, volume 149 of *IMA Vol. Math. Appl.*, pages 157–270. Springer, New York, 2009.

38 M. Laurent. Sums of squares, moment matrices and optimization over polynomials. In *Emerging applications of algebraic geometry*, volume 149 of *IMA Vol. Math. Appl.*, pages 157–270. Springer, New York, 2009.

39 J. R. Lee, P. Raghavendra, and D. Steurer. Lower bounds on the size of semidefinite programming relaxations. In *STOC*, pages 567–576, 2015.

40 L. Lovász. On the shannon capacity of a graph. *IEEE Trans. Inform. Theory*, 25:1–7, 1979.

41 A. Magen and M. Moharrami. Robust algorithms for on minor-free graphs based on the Sherali-Adams hierarchy. In *APPROX-RANDOM*, pages 258–271, 2009.

42 M. Mastrolilli. High degree sum of squares proofs, bienstock-zuckerberg hierarchy and CG cuts. In *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, pages 405–416, 2017.

43 R. Meka, A. Potechin, and A. Wigderson. Sum-of-squares lower bounds for planted clique. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 87–96, 2015.

44 T.S. Motzkin. The arithmetic-geometric inequality. *Symposium on Inequalities*, pages 205–224, 1967. cited By 1.

45 Y. Nesterov. *Global quadratic optimization via conic relaxation*, pages 363–384. Kluwer Academic Publishers, 2000.

46 Y. Nesterov and A. Nemirovskii. *Interior Point Polynomial Algorithms in Convex Programming*. Studies in Applied Mathematics. Society for Industrial and Applied Mathematics, 1994.

47 J. Oxley. *Matroid theory*, volume 2 of *Oxford Graduate Texts in Mathematics*. Oxford University Press, 2011.

48 P. Parrilo. *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. PhD thesis, California Institute of Technology, 2000.

49 Aaron Potechin and David Steurer. Exact tensor completion with sum-of-squares. In *Proceedings of the 30th Conference on Learning Theory, COLT 2017, Amsterdam, The Netherlands, 7-10 July 2017*, pages 1619–1673, 2017.

50 M. Putinar. Positive polynomials on compact semi-algebraic sets. *Indiana Univ. Math. J.*, 42(3):969–984, 1993.

51 P. Raghavendra and N. Tan. Approximating csps with global cardinality constraints using sdp hierarchies. In *SODA*, pages 373–387, 2012.

52 B. Reznick. Forms derived from the arithmetic-geometric inequality. *Math. Ann.*, 283(3):431–464, 1989.

53 B. Reznick. Some concrete aspects of Hilbert's 17th Problem. In *Real algebraic geometry and ordered structures (Baton Rouge, LA, 1996)*, volume 253 of *Contemp. Math.*, pages 251–272. Amer. Math. Soc., Providence, RI, 2000.

54 Tselil Schramm and David Steurer. Fast and robust tensor decomposition with applications to dictionary learning. In *Proceedings of the 30th Conference on Learning Theory, COLT 2017, Amsterdam, The Netherlands, 7-10 July 2017*, pages 1760–1793, 2017.

55 N. Shor. Class of global minimum bounds of polynomial functions. *Cybernetics*, 23(6):731–734, 1987.

56 Johan Thapper and Stanislav Zivny. The limits of SDP relaxations for general-valued csps. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017.

57 G.M. Ziegler. *Lectures on Polytopes*. Springer Verlag, 2007.

Rainbow Vertex Coloring Bipartite Graphs and Chordal Graphs

Pinar Heggernes
Department of Informatics, University of Bergen, Norway
pinar.heggernes@uib.no

Davis Issac
Max Planck Institute for Informatics, Saarland Informatics Campus, Germany
dissac@mpi-inf.mpg.de

Juho Lauri
Nokia Bell Labs, Dublin, Ireland
juho.lauri@nokia-bell-labs.com

Paloma T. Lima
Department of Informatics, University of Bergen, Norway
paloma.lima@uib.no

Erik Jan van Leeuwen
Department of Information and Computing Sciences, Utrecht University, The Netherlands
e.j.vanleeuwen@uu.nl

Abstract

Given a graph with colors on its vertices, a path is called a rainbow vertex path if all its internal vertices have distinct colors. We say that the graph is rainbow vertex-connected if there is a rainbow vertex path between every pair of its vertices. We study the problem of deciding whether the vertices of a given graph can be colored with at most k colors so that the graph becomes rainbow vertex-connected. Although edge-colorings have been studied extensively under similar constraints, there are significantly fewer results on the vertex variant that we consider. In particular, its complexity on structured graph classes was explicitly posed as an open question.

We show that the problem remains NP-complete even on bipartite apex graphs and on split graphs. The former can be seen as a first step in the direction of studying the complexity of rainbow coloring on sparse graphs, an open problem which has attracted attention but limited progress. We also give hardness of approximation results for both bipartite and split graphs. To complement the negative results, we show that bipartite permutation graphs, interval graphs, and block graphs can be rainbow vertex-connected optimally in polynomial time.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Rainbow coloring, graph classes, polynomial-time algorithms, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.83

1 Introduction

Graph coloring and graph connectivity are two of the most famous topics in graph algorithms. Many different types of colorings and connectivity measures have been considered throughout time. The concept of rainbow coloring brings these two extensively studied topics together, and it was first defined a decade ago by Chartrand et al. [8] using edge-colorings. Let G be a connected, edge-colored graph. A *rainbow path* in G is a path all of whose edges are colored

with distinct colors, and G is *rainbow-connected* if there is a rainbow path between every pair of its vertices. The resulting computational problem Rainbow Coloring (RC) takes as input a connected (uncolored) graph G and an integer k, and the task is to decide whether the edges of G can be colored with at most k colors so that G is rainbow-connected. This problem has various applications in telecommunications, data transfer, and encryption [25, 4, 11] and has been studied rather thoroughly from both graph-theoretic and complexity-theoretic viewpoints (see related work below and the surveys [19, 25]).

The intense interest in Rainbow Coloring led Krivelevich and Yuster [18] to define a natural variant on *vertex-colored* graphs. Here, a path in a vertex-colored graph H is a *rainbow vertex path* if all its *internal* vertices have distinct colors. We say that H is *rainbow vertex-connected* if there is a rainbow vertex path between every pair of its vertices. Similarly to the edge variant, Rainbow Vertex Coloring (RVC) is the decision problem in which we are given a connected (uncolored) graph H and an integer k, and the task is to decide whether the vertices of H can be colored with at most k colors such that H is rainbow vertex-connected. The *rainbow vertex connection number* of G, denoted by $\mathbf{rvc}(G)$, is the minimum k such that G has a rainbow vertex coloring with k colors. RVC is NP-complete for every $k \geq 2$ [10, 9], and remains NP-complete for $k = 3$ for bipartite graphs [23]. In addition, it is NP-hard to approximate $\mathbf{rvc}(G)$ within a factor of $2 - \varepsilon$ unless $\mathsf{P} \neq \mathsf{NP}$, for any $\varepsilon > 0$ [13]. It is also known that RVC is linear-time solvable on planar graphs for every fixed k [19]. Finally, assuming the Exponential Time Hypothesis, there is no algorithm for solving RVC in time $2^{o(n^{3/2})}$ for any $k \geq 2$ [19].

A stronger variant of rainbow vertex-colorings was introduced by Li et al. [24]. A vertex-colored graph H is *strongly rainbow vertex-connected* if between every pair of vertices of H, there is a *shortest* path that is also a rainbow vertex path. The Strong Rainbow Vertex Coloring (SRVC) problem takes as input a connected (uncolored) graph H and an integer k, and the task is to decide whether the vertices of H can be colored such that H is strongly rainbow vertex-connected. This definition is the vertex variant of the Strong Rainbow Coloring problem, which was also broadly studied (see related work below and the surveys [19, 25]). The *strong rainbow vertex connection number* of G, denoted by $\mathbf{srvc}(G)$, is the minimum k such that G has a strong rainbow vertex coloring with k colors. SRVC is NP-complete for every $k \geq 2$ [12] and linear-time solvable on planar graphs for every fixed k [19]. In addition, it is NP-hard to approximate $\mathbf{srvc}(G)$ within a factor of $n^{1/2-\varepsilon}$ unless $\mathsf{P} \neq \mathsf{NP}$, for any $\varepsilon > 0$ [13].

While RC has been widely studied in more than 300 published papers, we are unaware of any further complexity results on RVC and SRVC than those mentioned previously. In particular, the complexity of RVC and SRVC on structured graph classes is mostly open. This led Lauri [19, Open problem 6.6] to explicitly ask the following:

For what restricted graph classes do RVC *and* SRVC *remain NP-complete?*

Our Results. In this paper, we make significant progress towards addressing this open problem. In particular, we study bipartite graphs and chordal graphs, and some of their subclasses, and give hardness results and polynomial-time algorithms for RVC and SRVC. Our main result is a hardness result for bipartite apex graphs:

▶ **Theorem 1.** *Let G be a bipartite apex graph of diameter 4. It is* NP*-complete to decide both whether $\mathbf{rvc}(G) \leq 4$ and whether $\mathbf{srvc}(G) \leq 4$. Moreover, it is* NP*-hard to approximate $\mathbf{rvc}(G)$ and $\mathbf{srvc}(G)$ within a factor of $5/4 - \varepsilon$, for every $\varepsilon > 0$.*

This result is particularly interesting since no hardness result was known on a sparse graph class (like apex graphs) for any of the variants of rainbow coloring. Moreover, this result can be considered tight in conjunction with the known result that RVC and SRVC are linear-time solvable on planar graphs for every fixed number of colors k [19]. Finally, we observe (like Li et al. [23]) that $\mathbf{rvc}(G)$ and $\mathbf{srvc}(G)$ can be computed in linear time if G is a bipartite graph of diameter 3, providing further evidence that this result is tight.

For general bipartite graphs and for split graphs (a well-known subclass of chordal graphs), we exhibit stronger hardness results:

▶ **Theorem 2.** *Let G be a bipartite graph of diameter 4. It is* NP*-complete to decide both whether $\mathbf{rvc}(G) \leq k$ and whether $\mathbf{srvc}(G) \leq k$, for every $k \geq 3$. Moreover, it is* NP*-hard to approximate both $\mathbf{rvc}(G)$ and $\mathbf{srvc}(G)$ within a factor of $n^{1/3-\varepsilon}$, for every $\varepsilon > 0$.*

We remark that, previously, it was only known that deciding whether $\mathbf{rvc}(G) \leq 3$ for bipartite graphs G is NP-complete by the result of [23]. Our construction, however, is conceptually simpler, gives hardness for every $k \geq 3$, and is easily extended to the strong variant. Moreover, for RVC on general graphs, this result implies a considerable improvement over the previous result of Eiben et al. [13] which only excluded a polynomial-time approximation with a factor of less than 2 assuming $\mathsf{P} \neq \mathsf{NP}$.

▶ **Theorem 3.** *Let G be a split graph of diameter 3. It is* NP*-complete to decide both whether $\mathbf{rvc}(G) \leq k$ and whether $\mathbf{srvc}(G) \leq k$, for every $k \geq 2$. Moreover, it is* NP*-hard to approximate both $\mathbf{rvc}(G)$ and $\mathbf{srvc}(G)$ within a factor of $n^{1/3-\varepsilon}$, for every $\varepsilon > 0$.*

To the best of our knowledge, our results for split graphs give the first non-trivial graph class besides diameter-two graphs for which the complexity of the edge and the vertex variant differ (see e.g, [19, Table 4.2] but note that it contains a typo erroneously claiming that RVC can be solved in polynomial-time for split graphs). In particular, RC can be solved in polynomial time on split graphs when $k \geq 4$ [5, 7]. Moreover, we observe that $\mathbf{rvc}(G)$ and $\mathbf{srvc}(G)$ can be computed in linear time if G is a graph of diameter 2, providing evidence that this result is tight.

To contrast our hardness results, we show that both problems can be solved in polynomial time on several other subclasses of bipartite graphs and chordal graphs.

▶ **Theorem 4.** *If G is a bipartite permutation graph, a block graph, or a unit interval graph, then $\mathbf{rvc}(G)$ and $\mathbf{srvc}(G)$ can be computed in linear time. If G is an interval graph, then $\mathbf{rvc}(G)$ can be computed in linear time.*

Combined, these results paint a much clearer picture of the complexity landscape of RVC and SRVC than was possible previously.

Related Work. We briefly survey the known work for the edge variants of rainbow coloring; we refer to [19, 25] for more detailed surveys. RC is NP-complete for every $k \geq 2$ [4, 2, 22], even on chordal graphs [5]. On split graphs, RC is NP-complete when $k \in \{2, 3\}$, but solvable in polynomial time otherwise [5, 7]. It is also solvable in polynomial time on threshold graphs [5]. On bridgeless chordal graphs, there is a linear-time (3/2)-approximation algorithm for RC, however the problem cannot be approximated with a factor less than 5/4 on this graph class, unless $\mathsf{P} = \mathsf{NP}$ [6]. Some lower bounds on algorithms for solving RC are given by Kowalik et al. [17] and Agrawal [1] under the Exponential Time Hypothesis.

For the strong edge variant, an edge-colored graph is said to be *strongly rainbow-connected* if there is a rainbow shortest path between every pair of its vertices. The problem of deciding whether the edges of a given graph G can be colored in k colors to make G strongly rainbow-connected is referred to as SRC. For $k = 2$, it is not difficult to verify that RC is equivalent

to SRC. Not surprisingly, SRC is also NP-complete for $k \geq 2$ [2]. In contrast to RC, SRC remains hard on split graphs for every $k \geq 2$ [19, Theorem 4.1]. Moreover, on n-vertex split graphs, it is NP-hard to approximate SRC within a factor of $n^{1/2-\varepsilon}$ for any $\varepsilon > 0$, while RC admits an additive-1 approximation [5]. The former statement also holds for n-vertex bipartite graphs instead of split graphs [2]. For block graphs, computing SRC can be done in linear time [16], while RC on block graphs is conjectured to be hard (see [19, Conjecture 6.3] or [16]). In general, it appears that despite the interest, there are fewer complexity-theoretic results on SRC. In fact, the same is true when considering combinatorial results (see [25] for a broader discussion).

2 Preliminaries

In this paper, we work on undirected simple graphs. Such a graph is denoted by $G = (V, E)$, where V is the vertex set of G, and E is the edge set. We let n denote the number of vertices of G. For a vertex $x \in V$, $N(x)$ is the set of its *neighbors*, and $\deg(x) = |N(x)|$ is its *degree*. For a $S \subseteq V$, the subgraph of G induced by S is denoted by $G[S]$. A *cut vertex* of G is a vertex whose removal increases the number of connected components of G.

Given a path $P = x_1, x_2, \ldots, x_{p-1}, x_p$ in G, the vertices from x_2 to x_{p-1} are called the *internal vertices* of P. The *distance* between two vertices u and v in G, denoted by $\text{dist}(u, v)$, is the length of a shortest path between u and v. The *diameter* of G, denoted by $\text{diam}(G)$, is the maximum distance between any pair of vertices of G.

A *k-coloring of G* is a function $c : V \to \{1, 2, \ldots, k\}$. (From now on, we will denote a set of consecutive integers from 1 to k as $[k]$.) A *coloring* is simply a k-coloring for some $k \leq n$. A coloring c is *proper* if $c(u) \neq c(v)$ for every edge $uv \in E$. The *chromatic number* of G, denoted by $\chi(G)$, is the smallest k such that G has a proper k-coloring. A *d-distance coloring* of G is a coloring c of G such that $c(u) \neq c(v)$ whenever $\text{dist}(u, v) \leq d$. The minimum number of colors needed for a d-distance coloring of G is known as the *d-distance chromatic number* of G, and it is denoted by $\chi_d(G)$. Note that $\chi_d(G)$ is equivalent to $\chi(G^d)$, i.e., the chromatic number of the d^{th} power of G.

Since, in this paper, we will only be working on the vertex variant of the rainbow coloring and rainbow connectivity, we might sometimes omit the word "vertex" when there is no confusion. The parameter $\mathbf{srvc}(G)$ was defined by Li et al. [24], and they also verified that $\text{diam}(G) - 1 \leq \mathbf{rvc}(G) \leq \mathbf{srvc}(G) \leq n - 2$. The following upper bound was mentioned in [19] (see the same reference for further discussion and examples).

▶ **Proposition 5** ([19]). *Let G be a connected graph with $\text{diam}(G) = d \geq 3$. Then*

$$d - 1 \leq \mathbf{rvc}(G) \leq \mathbf{srvc}(G) \leq \chi_{d-2}(G).$$

Proof. There are at least two vertices in G connected by a shortest path of length d. Clearly, every coloring must use at least $d - 1$ colors to rainbow-connect this pair. On the other hand, between every pair of vertices u and v, there is a path of length at most d, meaning that it contains at most $d - 1$ internal vertices. As every $(d - 2)$-distance coloring colors these internal vertices distinctly, the statement follows. ◀

A *dominating set* of G is a set $D \subseteq V$ such that every vertex in $V \setminus D$ is adjacent to at least one vertex in D. If $G[D]$ is connected, then D is a *connected dominating set*. The minimum size of a connected dominating set in G, denoted by $\gamma_c(G)$, is known as the *connected domination number* of G. This parameter provides an upper bound on the rainbow

vertex connection number of a connected graph, since G becomes rainbow vertex-connected by simply coloring all vertices of the connected dominating set distinctly, and the remaining vertices with any of the already used colors. This observation can be derived from [18].

▶ **Proposition 6** ([18]). *If G is a connected graph, then* $\mathbf{rvc}(G) \leq \gamma_c(G)$.

2.1 Graph classes

As we will be studying the mentioned problems on some graph classes, let us give a brief definition of these classes here. More definitions and properties will be added as needed when we handle these graphs. A detailed background on these graph classes can be found, for example, in the book by Brandstädt, Le, and Spinrad [3].

A graph is an *apex graph* if it contains a vertex (called an *apex*) whose removal results in a planar graph. A graph is *chordal* if all of its induced simple cycles are of length 3. Some well-known subclasses of chordal graphs are interval graphs, split graphs, and block graphs. A graph is an *interval graph* if it is chordal and it contains no triple of non-adjacent vertices, such that there is a path between every two of them that does not contain a neighbor of the third. A graph is a *split graph* if its vertex set can be partitioned into an independent set and a clique. A graph is a *block graph* if every biconnected component (block) of G is a complete graph.

Let σ be a permutation of the integers between 1 and n. We can make a graph G_σ on vertex set $[n]$ in the following way. Vertices i and j are adjacent in G_σ if and only if they appear in σ in the opposite order of their natural order. A graph on n vertices is a *permutation graph* if it is isomorphic to G_σ for some permutation σ of the integers between 1 and n. A graph is a *bipartite permutation* graph if it is both a bipartite graph and a permutation graph.

2.2 Hypergraph coloring

For our hardness reductions we will use a well-known NP-complete problem called Hypergraph Coloring. A *hypergraph* $H = (N, \mathcal{E})$ with vertex set N and hyperedge set $\mathcal{E}$ is a generalization of a graph, in which edges can contain more than two vertices. Thus $\mathcal{E}$ consists of subsets of N of arbitrary size. The definition of a (vertex) coloring of a hypergraph is exactly that same as that of a graph. In a colored hypergraph, an edge is called *monochromatic* if all of its vertices received the same color. A *proper coloring* of a hypergraph generalizes a proper coloring of a graph in a natural way: we require that no hyperedge is monochromatic. To avoid trivial cases, we can assume from now on that every hyperedge contains at least two vertices. Thus a proper coloring must always use at least two colors.

The Hypergraph Coloring problem takes as input a hypergraph H and an integer k and asks whether there is a proper coloring of H with at most k colors. The problem is well-known to be NP-complete for every $k \geq 2$ [26]. The Graph Coloring problem takes as input an undirected graph G and asks to determine the smallest k such that G has a proper k-coloring. This problem is NP-hard to approximate within a factor of $n^{1-\varepsilon}$ for any $\varepsilon > 0$, where n is the number of vertices [30]. Finally, the Planar 3-Coloring problem takes as input a planar graph G and asks whether G has a proper 3-coloring. This problem is NP-complete [14].

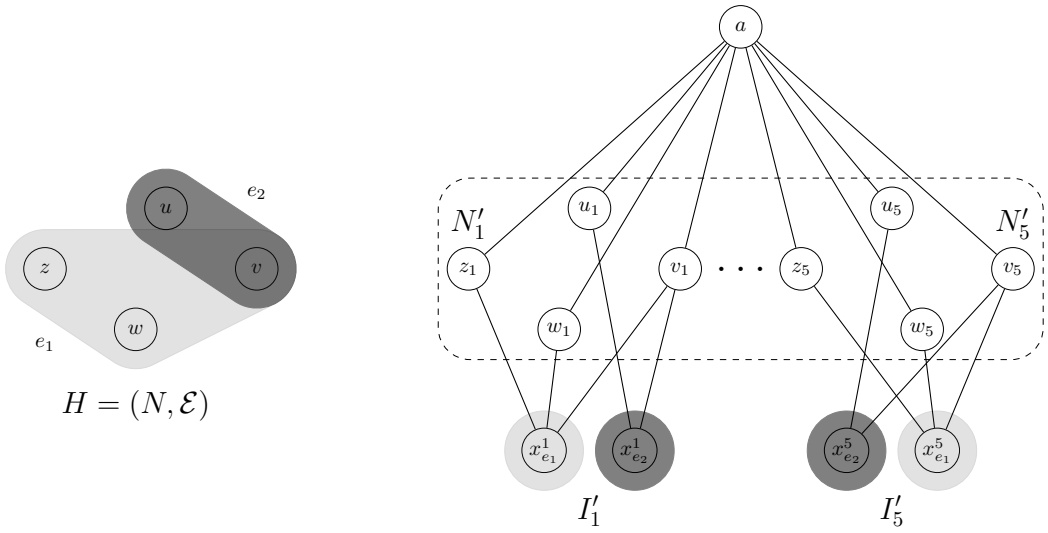


Figure 1 A hypergraph $H = (N, \mathcal{E})$ (left) transformed into a bipartite graph G (right) as described in the proof of Lemma 8. The dashed rectangle with rounded corners contains the sets in N'.

3 Bipartite graphs and their subclasses

In this section, we show that RVC and SRVC are hard on bipartite graphs for $k \geq 3$. We complement these results by showing that both problems can be solved in linear time on bipartite permutation graphs. We first observe that computing $\mathbf{rvc}(G)$ or $\mathbf{srvc}(G)$ is easy on bipartite graphs of diameter 3. The same observation was made by Li et al. [23].

▶ **Proposition 7** ([23]). *If G is a bipartite graph with $\mathrm{diam}(G) = 3$, then $\mathbf{rvc}(G) = \mathbf{srvc}(G) = 2$. Moreover, such a coloring can be found in linear time.*

Proof. The statement follows from Proposition 5 and the fact that every bipartite graph has a proper 2-coloring that can be found in linear time. ◀

It turns out that if $\mathrm{diam}(G) \geq 4$, then $\mathbf{rvc}(G)$ and $\mathbf{srvc}(G)$ of a bipartite graph G become much harder to compute, as claimed in Theorem 2. We prove the following general construction.

▶ **Lemma 8.** *Let H be a hypergraph on n vertices. Then in polynomial time we can construct a bipartite graph G of diameter 4 and with $O(n^3)$ vertices such that for any $k \in [n]$, H has a proper k-coloring if and only if G has a $(k+1)$-coloring under which G is (strongly) rainbow vertex-connected. Moreover, if H is a planar graph, then G is an apex graph.*

Proof. Let $H = (N, \mathcal{E})$ be an arbitrary hypergraph and let $n = |N|$. We construct a bipartite graph $G = (\{a\} \cup N' \cup I',\ E)$ where $N' = N'_1 \cup \cdots \cup N'_{n+1}$, $I' = I'_1 \cup \cdots \cup I'_{n+1}$, $N'_i := \{v_i \mid v \in N\}$, $I'_i := \{x^i_e \mid e \in \mathcal{E}\}$ and $E := \{av_i \mid v \in N, i \in [n+1]\} \cup \{v_i x^i_e \mid v \in N, e \in \mathcal{E}, i \in [n+1], v \in e\}$. Let $V = \{a\} \cup N' \cup I'$. A bipartition of G is given by $(\{a\} \cup I',\ N')$. Observe that $\mathrm{diam}(G) = 4$ and that G has $O(n^3)$ vertices. Moreover, if H is a planar graph, then G consists of vertex a plus $n+1$ copies of the graph obtained from H by subdividing each edge of H, and thus G is an apex graph. For an illustration of the construction, see Figure 1.

Consider any proper k-coloring $h : N \to [k]$ of H, i.e., no hyperedge of H is monochromatic under h. We construct a coloring $c : V \to [k+1]$ in the following way. First, for every $v \in N$, we give the vertices $v_1, v_2, \ldots, v_n$ of G the same color as v, i.e., $c(v_i) = h(v)$ for all $v \in N$ and $i \in [n+1]$. We give vertex a the color $k+1$, i.e., $c(a) = k+1$. The vertices in I all receive the same color, which is any arbitrary color in $[k+1]$. Now we prove that G is strongly rainbow vertex-connected under c by showing that there is a rainbow vertex shortest path between every pair of vertices. The only non-trivial case is when both vertices of the pair are in I. Consider two distinct vertices $x_e^i, x_f^j \in I$ (it is possible that $e = f$ or $i = j$ but not both). Since e and f are not monochromatic under h, we can pick two distinct vertices $u \in e$ and $v \in f$ such that $h(u) \neq h(v)$. It is clear that the path $x_e^i u a v x_f^j$ is a shortest path between x_e^i and x_f^j and that it is a rainbow vertex path. Hence, G is strongly rainbow vertex-connected under c.

Conversely, let c be a $(k+1)$-coloring of G under which G is (strongly) rainbow vertex-connected. For each $i \in [n+1]$, define h_i to be the vertex coloring of H such that $h_i(v) = c(v_i)$ for all $v \in N$. Let M_i be the set of vertices $v \in N$ such that $h_i(v) \neq c(a)$. Let $h_i'(v) = h_i(v)$ if $v \in M_i$ and $h_i'(v) = 1$ otherwise. We claim that there exists an $i \in [n+1]$ such that h_i' is a proper k-coloring of H. For the sake of contradiction, suppose that h_i' is not a proper k-coloring of H for every $i \in [n+1]$. For each $i \in [n+1]$, let $e_i \in \mathcal{E}$ be a monochromatic edge under h_i'. Suppose that, for some $i \in [n+1]$, all vertices in e_i are colored $c(a)$ under c. Then any path from $x_{e_i}^i$ to $x_{e_i}^j$ for some $j \neq i$ uses two vertices having color $c(a)$ under c. Hence, c would not be a rainbow vertex coloring, a contradiction. Therefore, for each $i \in [n+1]$, there is a vertex $v_i \in e_i$ for which $c(v_i) \neq c(a)$. Suppose now that for every $i \in [n+1]$, all vertices in e_i are colored either $c(v_i)$ or $c(a)$ under c. If $c(v_i) = c(v_j)$ for $i \neq j$, then any path from $x_{e_i}^i$ to $x_{e_j}^j$ uses either two vertices having color $c(a)$ or two vertices having color $c(v_i) = c(v_j)$ under c. This would contradict the assumption that G is rainbow vertex-connected under c. Hence, $c(v_i) \neq c(v_j)$ for all distinct $i, j \in [n+1]$. This implies that c uses at least $n+2$ colors, a contradiction to the assumptions that c is a $(k+1)$-coloring of G and that $k \in [n]$. Therefore, for some $i \in [n+1]$, there is a vertex $v_i' \in e_i$ for which $c(v_i') \neq c(a)$ and $c(v_i') \neq c(v_i)$. The latter implies that e_i is not monochromatic under h_i', a contradiction. The claim follows, and thus H has a proper k-coloring. ◀

Proof of Theorem 2. For membership in NP, a certificate that $\mathbf{rvc}(G) \leq k$ ($\mathbf{srvc}(G) \leq k$) consists of a k-coloring and a list of (shortest) paths, one for every pair of non-adjacent vertices, that are rainbow vertex connected. For NP-hardness, we observe that the transformation of Lemma 8 implies a straightforward reduction from Hypergraph Coloring. Since Hypergraph Coloring is NP-complete for each $k \geq 2$, this proves the first part of the theorem.

For the second part of the theorem, we consider an instance of Graph Coloring that consists of a graph on ℓ vertices and apply Lemma 8. Note that the total number of vertices in G is $n = O(\ell^3)$. From the hardness of approximation of Graph Coloring, we know that for all $\varepsilon > 0$, it is NP-hard to distinguish between the case when H is properly colorable with ℓ^ε colors and the case when H is not properly colorable with fewer than $\ell^{1-\varepsilon}$ colors [30]. By Lemma 8, this implies that it is NP-hard to distinguish between the case when G is (strong) rainbow vertex colorable with $\ell^\varepsilon + 1 \leq n^\varepsilon + 1$ colors and the case when G is not (strong) rainbow vertex colorable with fewer than $\ell^{1-\varepsilon} + 1 = \Omega(n^{1/3-\varepsilon})$ colors. The second statement of the theorem follows. ◀

We then proceed to give a proof of Theorem 1. This result can be considered as a first step to understand rainbow coloring on sparse graphs classes.

Proof of Theorem 1. The proof follows along the same lines as the proof of the first part of Theorem 2. Instead of Hypergraph Coloring, however, we reduce from Planar 3-Coloring, the problem of deciding whether a planar graph has a proper 3-coloring. This problem is NP-complete. The statement follows from Lemma 8, because the graph resulting from the construction is a bipartite apex graph of diameter 4.

For the hardness of approximation, we recall that any planar graph has a proper 4-coloring, and thus the graph G constructed in Lemma 8 has a 5-coloring under which G is rainbow vertex-connected. Hence, Lemma 8 combined with the NP-hardness of Planar 3-Coloring makes it NP-hard to decide whether G has a 5-coloring or a 4-coloring under which G is rainbow vertex-connected. ◀

We now complement the above hardness results with a positive result in the case when a bipartite graph is also a permutation graph, as claimed in Theorem 4. Bipartite permutation graphs have a desirable property, related to breadth-first search (BFS), that we will use heavily in our next result. Let us first define a chain graph. A bipartite graph is a *chain graph* if the vertices of the two independent sets A and B can be ordered as $\{a_1, a_2, \ldots, a_k\}$ and $\{b_1, b_2, \ldots, b_\ell\}$, such that $N(a_1) \subseteq N(a_2) \subseteq \cdots \subseteq N(A_k)$, equivalently, $N(b_\ell) \subseteq N(b_{\ell-1}) \subseteq \cdots \subseteq N(b_1)$.

In every bipartite permutation graph G it is possible to find a vertex v such that the levels $L_0, L_1, L_2, \ldots$ of the tree resulting from a BFS starting from v have the following properties. For all i, $L_0 = \{v\}$, L_i is an independent set and $G[L_i \cup L_{i+1}]$ is a chain graph. Moreover, for each level i, there exists a special vertex $a_i \in L_i$ such that $L_{i+1} \subset N(a_i)$. The vertex v can be picked as the first vertex of a *strong ordering*. It has been shown by Spinrad et al. [27] that a bipartite graph is a permutation graph if and only if it has a strong ordering, and such an ordering can be computed in linear time. The properties of the BFS tree above are well-known and easy to deduce from a strong ordering [29].

▶ **Theorem 9.** *If G is a bipartite permutation graph, then* $\mathbf{rvc}(G) = \mathbf{srvc}(G) = \mathrm{diam}(G) - 1$, *and the corresponding (strong) rainbow vertex coloring can be found in time that is linear in the size of G.*

Proof. Let $G = (V, E)$ be a bipartite permutation graph. Let v be a first vertex in a strong ordering for G. We start by doing a BFS on G with v as the root. Let k be the number of levels in the BFS tree in addition to level 0. Hence, L_i is the set of vertices in level i of the BFS tree, $0 \leq i \leq k$, with $L_0 = \{v\}$. Since $\mathrm{dist}(v, y) = k$ for every $y \in L_k$, we conclude that $\mathrm{diam}(G) \geq k$. Furthermore, if $\mathrm{dist}(x, y) > k - 1$ for some $x \in L_1$ and some $y \in L_k$, then we can conclude that $\mathrm{dist}(x, y) = k + 1$, where $x, v, a_1, a_2, \ldots, a_{k-1}, y$ is a shortest path between x and y. In this case, $\mathrm{diam}(G) = k + 1$. We distinguish between these two cases:

Case 1. $\mathrm{diam}(G) = k$.

We construct a strong rainbow vertex coloring $c : V \to [k - 1]$ for G in the following way. If $x \in L_i$, we define $c(x) = i$, for $1 \leq i \leq k - 1$. We define $c(v) = k - 1$, and we give arbitrary colors between 1 and $k - 1$ to the vertices of L_k. To see that G is indeed rainbow-connected under c, consider any pair $x, y \in V$. If $xy \in E$ or if they are in the same level of the BFS tree, there is nothing to prove, since $\mathrm{dist}(x, y) \leq 2$. Otherwise, we have exactly the following cases:

1. $x = v$ and $y \in L_j$: Then the path $v, a_1, \ldots, a_{j-1}, y$ is shortest and it is rainbow.
2. $x \in L_1$ and $y \in L_k$: In this case, $\mathrm{dist}(x, y) = k - 1$. Otherwise, since each L_i is an independent set, we would have $\mathrm{dist}(x, y) \geq k + 1$, which contradicts our assumption that

$\text{diam}(G) = k$. Since $\text{dist}(x, y) = k - 1$, every shortest path between x and y is rainbow, as every vertex of such a shortest path has to be in a distinct level of the BFS tree.

3. $x \in L_1$ and $y \in L_j$ with $2 \leq j \leq k - 1$: If $\text{dist}(x, y) = j - 1$, then again by the same argument used above, every shortest path between x and y is rainbow. If $\text{dist}(x, y) > j-1$, then $\text{dist}(x, y) = j + 1$, and the shortest path $x, v, a_1, \ldots, a_{j-1}, y$ has distinct colors on all its internal vertices. (Note that y might have the same color as v if $j = k - 1$, but this is fine since y is the end of the path.)
4. $x \in L_i$ and $y \in L_j$ with $2 \leq i < j \leq k$: If $\text{dist}(x, y) = j - i$, then every shortest path is rainbow. If $\text{dist}(x, y) > j - i$, then the path $x, a_{i-1}, a_i, \ldots, a_{j-1}, y$ is rainbow and has length $j - i + 1$, and it is therefore shortest.

Case 2. $\text{diam}(G) = k + 1$.

We construct a strong rainbow vertex coloring $c : V \to [k]$ for G in the following way. If $x \in L_i$, we define $c(x) = i$, for $1 \leq i \leq k - 1$. We define $c(v) = k$, and we give arbitrary colors between 1 and k to the vertices of L_k. To see that G is indeed rainbow-connected under c, consider any pair $x, y \in V$. Again, if $xy \in E$ or if they are in the same level of the BFS, there is nothing to prove, since $\text{dist}(x, y) \leq 2$. Otherwise, there is only one remaining case:

- $x \in L_i$ and $y \in L_j$, with $0 \leq i < j \leq k$: If $\text{dist}(x, y) = j - i$ then every shortest path between x and y is rainbow. Otherwise, the path $x, a_{i-1}, a_i, \ldots, a_{j-1}, y$ is rainbow and has length $j - i + 2$, therefore being shortest.

In both cases, c is a strong rainbow vertex coloring for G with $\text{diam}(G) - 1$ colors. By Proposition 5 we can conclude that $\mathbf{rvc}(G) = \mathbf{srvc}(G) = \text{diam}(G) - 1$. ◀

4 Chordal graphs and their subclasses

In this section, we investigate the complexity of RVC and SRVC on chordal graphs and some subclasses of chordal graphs. We start by proving that both problems are NP-complete when the input graph is a split graph, implying that they are also NP-complete on chordal graphs. On the positive side, we show that RVC is polynomial-time solvable on interval graphs, and both RVC and SRVC are polynomial-time solvable on block graphs and on unit interval graphs.

We start by observing that computing $\mathbf{rvc}(G)$ or $\mathbf{srvc}(G)$ is easy on graphs of diameter 2.

▶ **Proposition 10** ([18]). *If G is a graph with $\text{diam}(G) = 2$, then $\mathbf{rvc}(G) = \mathbf{srvc}(G) = 1$. Moreover, such a coloring can be found in linear time.*

Proof. Color each vertex of G with the same color. Since each shortest path between two vertices contains at most one internal vertex, G is strongly rainbow vertex-connected under this coloring. ◀

If G is a split graph of $\text{diam}(G) = 3$ (note that split graphs have diameter at most 3), then $\mathbf{rvc}(G)$ and $\mathbf{srvc}(G)$ become much harder to compute, as claimed in Theorem 3. We prove the following general construction, which closely mimics the construction of Lemma 8.

▶ **Lemma 11.** *Let H be a hypergraph on n vertices. Then in polynomial time we can construct a split graph G of diameter 3 and with $O(n^3)$ vertices such that for any $k \in [n]$, H has a proper k-coloring if and only if G has a k-coloring under which G is (strongly) rainbow vertex-connected.*

Proof. Let $H = (N, \mathcal{E})$ be an arbitrary hypergraph and let $n = |N|$. We construct a split graph $G = (N' \cup I',\ E)$ where $N' = N'_1 \cup \cdots \cup N'_{n+1}$, $I' = I'_1 \cup \cdots \cup I'_{n+1}$, $N'_i := \{v_i \mid v \in N\}$, $I'_i := \{x^i_e \mid e \in \mathcal{E}\}$ and $E := \{u_i v_j \mid u, v \in N,\ i, j \in [n+1]\} \cup \{v_i x^i_e \mid v \in N, e \in \mathcal{E}, i \in [n+1], v \in e\}$. Let $V = N' \cup I'$. The constructed graph G is a split graph since $G[I']$ is an independent set and $G[N']$ is a clique. Observe that $\mathrm{diam}(G) = 3$ and that G has $O(n^3)$ vertices. The construction is illustrated in Figure 1: note that since $G[N']$ is a clique, all possible edges now appear between the vertices inside the rectangle with rounded corners.

Consider any proper k-coloring $h : N \to [k]$ of H, i.e., no hyperedge of H is monochromatic under h. We construct a coloring $c : V \to [k]$ in the following way. First, for every $v \in N$, we give the vertices $v_1, v_2, \ldots, v_n$ of G the same color as v, i.e., $c(v_i) = h(v)$ for all $v \in N$ and $i \in [n+1]$. The vertices in I all receive the same color, which is any arbitrary color in $[k]$. Now, we prove that G is strongly rainbow vertex-connected under c by showing that there is a rainbow vertex shortest path between every pair of vertices. The only non-trivial case is when both vertices of the pair are in I. Consider two distinct vertices $x^i_e, x^j_f \in I$ (it is possible that $e = f$ or $i = j$ but not both). Since e and f are not monochromatic under h, we can pick two distinct vertices $u \in e$ and $v \in f$ such that $h(u) \neq h(v)$. It is clear that the path $x^i_e u v x^j_f$ is a shortest path between x^i_e and x^j_f and that it is rainbow vertex path.

Conversely, let c be a k-coloring of G under which G is (strongly) rainbow vertex-connected. For each $i \in [n+1]$, define h_i to be the vertex coloring of H such that $h_i(v) = c(v_i)$ for all $v \in N$. We claim that there exists an $i \in [n+1]$ such that h'_i is a proper k-coloring of H. For the sake of contradiction, suppose that h'_i is not a proper k-coloring of H for every $i \in [n+1]$. For each $i \in [n+1]$, let $e_i \in \mathcal{E}$ be a monochromatic edge under h'_i. Let v_i be an arbitrary vertex in e_i. Suppose now that for every $i \in [n+1]$, all vertices in e_i are colored $c(v_i)$ under c. If $c(v_i) = c(v_j)$ for $i \neq j$, then any path from $x^i_{e_i}$ to $x^j_{e_j}$ uses two vertices having color $c(v_i) = c(v_j)$ under c. This would contradict the assumption that G is rainbow vertex-connected under c. Hence, $c(v_i) \neq c(v_j)$ for all distinct $i, j \in [n+1]$. This implies that c uses at least $n+1$ colors, a contradiction to the assumptions that c is a k-coloring of G and $k \in [n]$. Therefore, for some $i \in [n+1]$, there is a vertex $v'_i \in e_i$ for which $c(v'_i) \neq c(a)$ and $c(v'_i) \neq c(v_i)$. The latter implies that e_i is not monochromatic under h'_i, a contradiction. The claim follows, and thus H has a proper H-coloring. ◀

Proof of Theorem 3. The proof follows in exactly the same way as Theorem 2, except that we apply Lemma 11 instead of Lemma 8. ◀

We now move on to the positive results. As a consequence of the following theorems, we complete the proof of Theorem 4.

▶ **Theorem 12.** *Let G be a block graph, and let ℓ be the number of cut vertices in G. Then $\mathbf{rvc}(G) = \mathbf{srvc}(G) = \ell$. The corresponding (strong) rainbow vertex coloring can be found in time that is linear in the size of G.*

Proof. Let $G = (V, E)$ be a block graph and $\{a_1, a_2, \ldots, a_\ell\}$ be the set of cut vertices of G. We construct a strong rainbow vertex coloring $c : V \to [\ell]$ for G by defining $c(a_i) = i$ for $i \in [\ell]$ and giving the other vertices arbitrary colors between 1 and ℓ. An important property of block graphs is that there is a unique shortest path between every pair of vertices. Moreover, each internal vertex of such a path is a cut vertex. Since all the cut vertices received distinct colors, these shortest paths are all rainbow. The proof follows by observing that $\mathbf{rvc}(G) \geq \mathbf{srvc}(G) \geq \ell$ as well. ◀

For our next result, we need to mention that every interval graph has a representation called an *interval model*. Let $\mathcal{I}$ be a set of n intervals of the real line. Then we can define a graph $G_{\mathcal{I}}$ with a vertex for each interval, such that two vertices are adjacent if and only if their corresponding intervals overlap. A graph G is an interval graph if and only if G is isomorphic to $G_{\mathcal{I}}$ for some set $\mathcal{I}$ of intervals. In this case $\mathcal{I}$ is called an interval model of G.

▶ **Theorem 13.** *If G is an interval graph, then* $\mathbf{rvc}(G) = \mathrm{diam}(G) - 1$*, and the corresponding rainbow vertex coloring can be found in time that is linear in the size of G.*

Proof. Let $G = (V, E)$ be an interval graph and $\mathcal{I}$ be an interval model for G. The interval corresponding to vertex v is denoted by I_v. For each interval $I \in \mathcal{I}$, we let $r(I)$ be its right endpoint and $\ell(I)$ its right endpoint. Let $I_u \in \mathcal{I}$ be such that $r(I_u) \leq r(I)$ for all $I \in \mathcal{I}$. Let $I_v \in \mathcal{I}$ be such that $\ell(I_v) \geq \ell(I)$ for all $I \in \mathcal{I}$. Let $P = u, x_1, x_2, \ldots, x_k, v$ be a shortest path between u and v in G. Observe that P is a connected dominating set. Furthermore, since P is a shortest path, $k \leq \mathrm{diam}(G) - 1$. By the way we defined u and v, we have that $N(u) \subseteq N(x_1)$ and $N(v) \subseteq N(x_k)$. This implies that the set $\{x_1, x_2, \ldots, x_k\}$ is also a connected dominating set. By Proposition 6, G has a rainbow vertex coloring $c : V \to [k]$ with $c(x_i) = i$, and we can give all the other vertices arbitrary colors. ◀

An interval graph is a *unit interval graph* if it has an interval model in which every interval has the same length (or no interval properly contains another interval). Unit interval graphs have the same BFS tree structure as that of bipartite permutation graphs, with the single difference that every level of the BFS tree is a clique instead of an independent set [15].

▶ **Theorem 14.** *If G is a unit interval graph, then* $\mathbf{rvc}(G) = \mathbf{srvc}(G) = \mathrm{diam}(G) - 1$*, and the corresponding (strong) rainbow vertex coloring can be found in time that is linear in the size of G.*

Proof. Let $G = (V, E)$ be a unit interval graph. Let v be the vertex corresponding to a first interval in an ordering of the intervals in the unit interval model of G by their right endpoints. Do a BFS on G with v as the root. Let L_i be the set of vertices in level i of the BFS tree, $0 \leq i \leq k$, with $L_0 = \{v\}$. Recall that, for $0 \leq i \leq k - 1$, there exists a special vertex $a_i \in L_i$ such that $L_{i+1} \subset N(a_i)$.

Consider a vertex $u \in L_k$. A shortest path between v and u has $k - 1$ internal vertices, which implies that $\mathrm{diam}(G) \geq k$. To construct a strong rainbow coloring $c : V \to [k - 1]$, we assign, for $1 \leq i \leq k - 1$, $c(x) = i$ if $x \in L_i$ and we give arbitrary colors to the vertices of L_k.

To see that G is strongly rainbow vertex-connected under c, consider $x, y \in V$. If both x and y are in the same level of the BFS tree, then they are adjacent. So let us consider the case when $x \in L_i$ and $y \in L_j$, with $1 \leq i < j \leq k$. If there is a shortest path between x and y each of whose vertices is in a distinct level of the BFS tree, then this path is rainbow. If this is not the case, we consider the path $x, a_i, a_{i+1}, \ldots, a_{j-1}, y$. In this case, this path is a shortest path between x and y, and its internal vertices have distinct colors, since only x and a_i belong to the same level of the BFS. This proves that c is indeed a strong rainbow coloring for G with $\mathrm{diam}(G) - 1$ colors. ◀

5 Concluding remarks and related problems

It should be mentioned that other variants of rainbow problems have been studied as well. When a coloring of the edges or the vertices of a graph is already given as input, we can ask whether the graph is rainbow-connected or rainbow vertex-connected. Both of these problems are known to be NP-complete even on highly restricted graphs, like interval graphs,

series-parallel graphs, and k-regular graphs for every $k \geq 3$ [21, 20, 28]. However, we stress that these problems are strictly different from RC and RVC. That is, complexity results on one problem are not transferable to the other.

Finally, we end our paper with the following open question.[1] A *diametral path* of a graph G is a shortest path whose length is equal to $\mathrm{diam}(G)$. A graph is a *diametral path* if every connected induced subgraph has a dominating diametral path.

▶ **Conjecture 15.** *Let G be a diametral path graph. Then* $\mathbf{rvc}(G) = \mathrm{diam}(G) - 1$.

References

1 Akanksha Agrawal. Fine-grained complexity of rainbow coloring and its variants. In *Proceedings of the 42nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 60:1–60:14, 2017.

2 Prabhanjan Ananth, Meghana Nasre, and Kanthi K. Sarpatwar. Rainbow connectivity: Hardness and tractability. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 241–251, 2011.

3 Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*. SIAM Monographs in Discrete Mathematics and Applications, 1999.

4 Sourav Chakraborty, Eldar Fischer, Arie Matsliah, and Raphael Yuster. Hardness and algorithms for rainbow connection. *Journal of Combinatorial Optimization*, 21(3):330–347, 2011.

5 L. Sunil Chandran and Deepak Rajendraprasad. Rainbow Colouring of Split and Threshold Graphs. In *Proceedings of the 18th Annual International Computing and Combinatorics Conference (COCOON)*, volume 7434 of *Lecture Notes in Computer Science*, pages 181–192. Springer, 2012.

6 L. Sunil Chandran and Deepak Rajendraprasad. Inapproximability of rainbow colouring. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 153–162, 2013.

7 L. Sunil Chandran, Deepak Rajendraprasad, and Marek Tesař. Rainbow colouring of split graphs. *Discrete Applied Mathematics*, 216:98–113, 2017.

8 Gary Chartrand, Garry L Johns, Kathleen A McKeon, and Ping Zhang. Rainbow connection in graphs. *Mathematica Bohemica*, 133(1):85–98, 2008.

9 Lily Chen, Xueliang Li, and Huishu Lian. Further hardness results on the rainbow vertex-connection number of graphs. *Theoretical Computer Science*, 481:18–23, 2013.

10 Lily Chen, Xueliang Li, and Yongtang Shi. The complexity of determining the rainbow vertex-connection of a graph. *Theoretical Computer Science*, 412(35):4531–4535, 2011.

11 Paul Dorbec, Ingo Schiermeyer, Elżbieta Sidorowicz, and Éric Sopena. Rainbow connection in oriented graphs. *Discrete Applied Mathematics*, 179:69–78, 2014.

12 Eduard Eiben, Robert Ganian, and Juho Lauri. On the complexity of rainbow coloring problems. In *Proceedings of the 26th International Workshop on Combinatorial Algorithms (IWOCA)*, volume 9538 of *Lecture Notes in Computer Science*, pages 209–220. Springer, 2015.

13 Eduard Eiben, Robert Ganian, and Juho Lauri. On the complexity of rainbow coloring problems. *Discrete Applied Mathematics*, 246:38–48, 2018.

14 M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some Simplified NP-Complete Graph Problems. *Theoretical Compututer Science*, 1(3):237–267, 1976.

[1] The statement is claimed in [19, Proposition 5.2] but its proof contains an error. Essentially, only an upper bound of $\mathrm{diam}(G) + 1$ is known by Proposition 6.

15 Pavol Hell and Jing Huang. Certifying LexBFS Recognition Algorithms for Proper Interval Graphs and Proper Interval Bigraphs. *SIAM Journal on Discrete Mathematics*, 18(3):554–570, 2004.

16 Melissa Keranen and Juho Lauri. Computing minimum rainbow and strong rainbow colorings of block graphs. *arXiv preprint arXiv:1405.6893*, 2014.

17 Łukasz Kowalik, Juho Lauri, and Arkadiusz Socała. On the fine-grained complexity of rainbow coloring. In *Proceedings of the 24th Annual European Symposium on Algorithms (ESA)*, pages 58:1–58:16, 2016.

18 Michael Krivelevich and Raphael Yuster. The rainbow connection of a graph is (at most) reciprocal to its minimum degree. *Journal of Graph Theory*, 63(3):185–191, 2010.

19 Juho Lauri. *Chasing the Rainbow Connection: Hardness, Algorithms, and Bounds*. PhD thesis, Tampere University of Technology, 2016.

20 Juho Lauri. Further hardness results on rainbow and strong rainbow connectivity. *Discrete Applied Mathematics*, 201:191–200, 2016.

21 Juho Lauri. Complexity of rainbow vertex connectivity problems for restricted graph classes. *Discrete Applied Mathematics*, 219:132–146, 2017.

22 Van Bang Le and Zsolt Tuza. Finding optimal rainbow connection is hard. Technical Report CS-03-09, Universität Rostock, 2009.

23 Shasha Li, Xueliang Li, and Yongtang Shi. Note on the complexity of deciding the rainbow (vertex-)connectedness for bipartite graphs. *Applied Mathematics and Computation*, 258:155–161, 2015.

24 Xueliang Li, Yaping Mao, and Yongtang Shi. The strong rainbow vertex-connection of graphs. *Utilitas Mathematica*, 93:213–223, 2014.

25 Xueliang Li, Yongtang Shi, and Yuefang Sun. Rainbow connections of graphs: A survey. *Graphs and Combinatorics*, 29(1):1–38, 2013.

26 László Lovász. Coverings and colorings of hypergraphs. In *Proceedings of the 4th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, pages 3–12, 1973.

27 Jeremy Spinrad, Andreas Brandstädt, and Lorna Stewart. Bipartite permutation graphs. *Discrete Applied Mathematics*, 18(3):279–292, 1987.

28 Kei Uchizawa, Takanori Aoki, Takehiro Ito, Akira Suzuki, and Xiao Zhou. On the Rainbow Connectivity of Graphs: Complexity and FPT Algorithms. *Algorithmica*, 67(2):161–179, 2013.

29 Ryuhei Uehara and Gabriel Valiente. Linear structure of bipartite permutation graphs and the longest path problem. *Information Processing Letters*, 103(2):71–77, 2007.

30 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 681–690. ACM, 2006.

Listing Subgraphs by Cartesian Decomposition

Alessio Conte
National Institute of Informatics, Tokyo, Japan
conte@nii.ac.jp

Roberto Grossi
Dipartimento di Informatica, Università di Pisa, Pisa, Italy
grossi@di.unipi.it

Andrea Marino
Dipartimento di Informatica, Università di Pisa, Pisa, Italy
marino@di.unipi.it

Romeo Rizzi
Dipartimento di Informatica, Università di Verona, Verona, Italy
romeo.rizzi@univr.it

Luca Versari
Dipartimento di Informatica, Università di Pisa, Pisa, Italy
luca.versari@di.unipi.it

Abstract

We investigate a decomposition technique for listing problems in graphs and set systems. It is based on the Cartesian product of some iterators, which list the solutions of simpler problems. Our ideas applies to several problems, and we illustrate one of them in depth, namely, listing all minimum spanning trees of a weighted graph G. Here iterators over the spanning trees for unweighted graphs can be obtained by a suitable modification of the listing algorithm by [Shioura et al., SICOMP 1997], and the decomposition of G is obtained by suitably partitioning its edges according to their weights. By combining these iterators in a Cartesian product scheme that employs Gray coding, we give the first algorithm which lists all minimum spanning trees of G in constant delay, where the delay is the time elapsed between any two consecutive outputs. Our solution requires polynomial preprocessing time and uses polynomial space.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Graph algorithms, listing, minimum spanning trees, constant delay

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.84

Funding This work has been partially supported by MIUR (Italian Ministry of University and Research) and JST CREST, Grant Number JPMJCR1401, Japan

1 Introduction

Listing problems in set systems have solutions corresponding to sets of elements from a ground set $\mathcal{U}$. Set systems formalize, among others, most subgraph listing problems as subgraphs are usually modeled as sets of vertices or edges (e.g. independent sets where $\mathcal{U}$ is the vertex set, or matchings where $\mathcal{U}$ is the edge set). A listing problem can be thus identified with the set $\mathcal{P} \subseteq 2^{\mathcal{U}}$ of its solutions to be listed, where typically each solution satisfies certain properties (e.g. maximality under inclusion).

In this paper we consider *decomposable* listing problems, where a problem $\mathcal{P}$ is decomposable if it can be modeled as the Cartesian product $\mathcal{P} = S_1 \times \cdots \times S_h$, with $S_1, \ldots, S_h \subseteq 2^{\mathcal{U}}$. We call this a *Cartesian decomposition*.

Many graph listing problems offer natural Cartesian decompositions. A trivial example is given by the (maximal or not) independent sets of a graph G, which can be decomposed using the connected components of G: indeed, any independent set of G corresponds to choosing an independent set from each of its connected components; hence letting S_i be all the independent sets in the i-th connected component in G, all those in G are obtained by combining them as $\mathcal{P} = S_1 \times \cdots \times S_h$, where $X \times Y$ returns all the unions of their vertex sets, $\{x \cup y : x \in X, y \in Y\}$. This same decomposition applies also to other types of subgraphs such as matchings and the edge sets.

This already can give improvements in the general case: even a simple reduction, such as reducing Steiner trees in a graph to combinations of Steiner trees in its biconnected components (see Section 4), already makes listing Steiner trees easier on graphs with small biconnected components. On the other hand, we will see in this paper that there are less trivial Cartesian decompositions, which allow us to reduce some problems to simpler ones, and design more efficient listing algorithms.

For a Cartesian decomposition $\mathcal{P} = S_1 \times \cdots \times S_h$, we observe that each set S_i is the output of some listing algorithm A_i, for $1 \leq i \leq h$. Thus S_i is not explicitly given, and may even have exponential size. As using polynomial space is one of the goals of this paper, among others, we consider A_i as a procedure that outputs all solutions of S_i in some order in polynomial space[1]. There are several definitions of efficiency for listing algorithms [15]: we consider the *delay*, that is, a worst-case measure representing the maximum time elapsed between any two consecutive outputs of A_i. If A_i has bounded delay (e.g. polynomial in the size of G for listing subgraphs of G), it is also called output-sensitive, as the total running time is proportional to the size $|S_i|$.

Given h listing algorithms $A_1, \ldots, A_h$, we model them as iterators over the sets $S_1, \ldots, S_h$, thus avoiding to store explicitly the latter ones. We thus consider the problem of listing implicitly the solutions of the Cartesian product $S_1 \times \cdots \times S_h$, which amounts to listing all the solutions of $\mathcal{P}$. This allow us to design the resulting listing algorithm that handle a complex counter with h digits with constant delay,[2] where the i-th digit goes through $|S_i|$ values and each configuration is a solution of $\mathcal{P}$. While intuitively a good representation, this is not the whole picture of the algorithmic challenge. For example, (re)setting digit i to "0" may be expensive, as we should pay for this solution the preprocessing cost of A_i; even if the latter is just $O(1)$ time, it does not guarantee a good delay as we may need to reset many counters at once, and this gives us a delay of $\Omega(h)$, and possibly worse; if opting for Gray codes, as we do, we should have some control on the order in which solutions of S_i are listed, which requires to modify A_i.

In this paper we propose a technique for iterating over the Cartesian product which allows us, under suitable conditions, to list the solutions of $\mathcal{P}$, with polynomial preprocessing cost and space usage, and with delay bounded by the worst case delay on any A_i. This exploits a form of Gray coding and common properties of state-of-the-art listing algorithms.

After describing how to list solutions using Cartesian decomposition in Section 2, we give some concrete examples of the wide class of problems which can be decomposed in Section 1.1 (or more in detail in Section 4), and study the case of minimum spanning trees

[1] Otherwise, if A_i takes exponential space, we can just use S_i directly, and our listing problem becomes of little interest.

[2] Constant amortized cost is easier to obtain, as shown in [10, Chapter 17].

(MSTs hereafter) in Section 3 to show how to iron all the details in a complete example. We will propose a decomposition which reduces the MSTs of a graph G to the Cartesian product among the spanning trees of a set of unweighted graphs $G_1, \ldots, G_h$. Furthermore, we will show that, by combining this decomposition with a modified version of the state of the art algorithm for listing spanning trees [29], we can obtain the first listing algorithm for MSTs with constant delay, using $O(mn)$ space.

1.1 Related Work

Solving complex problems by decomposing them into simpler problems is the main principle in algorithm design, and many techniques follow that principle: divide-and-conquer, dynamic programming, and so on. For enumeration and listing, some papers addressed this issue: backtracking algorithms [5, 27], and binary partition algorithms [4] in particular, can be seen as an implicit form of decomposition. Other forms of decompositions for listing include [24], which lists all arborescences in a digraph by modelling the solution space as a polynomial, and decomposing this into prime factors, [13] which deletes vertices and edges to generate formulas for counting subgraphs such as spanning trees and acyclic subgraphs, [25] which uses the treewidth decomposition for enumerating purposes, and [31] which shows how to decompose some problems into quasi-independent subproblems using clique separators.

In the case of fixed parameter algorithms, a form of decomposition has been explicited in the so-called *kernelization* which is a technique which preprocesses the input to get a smaller input, called *kernel*. The result of solving the problem on the kernel should either be the same as on the original input, or it should be easy to transform the output on the kernel to the desired output for the original problem [11]. As for listing algorithms for graphs, in many cases the instance of the problem can be partitioned in *several kernels*, so that the listing algorithm can be solved in each of them, and the result of the original problem can be obtained by the result of a Cartesian product among these set of solutions. Some examples are given next, and a more detailed discussion can be found in Section 4.

Application examples. Decomposition, for instance considering connected or biconnected components separately, applies to several listing problems including st-paths [4], unweighted spanning trees, Steiner trees, maximal independent sets [7], maximal induced matching [2], maximal k-degenerate subgraphs [8], minimal feedback vertex/arc sets [28], bipartite subgraphs [33], bounded girth subgraphs [9], dominating sets [19], and acyclic orientations [6] (see Section 4 for more details).[3] One of the goals of this paper is explicitly defining this Cartesian decomposition in a general way, so as to be used as a black box in future works.

MSTs, which we consider as case study for this technique, have a less trivial but powerful decomposition. Spanning trees (in short STs) and MSTs have a rich and long history which has been pointed out in several surveys [3, 26, 20, 14]. Due to this interest, several listing algorithms for STs and MSTs have been proposed over the years. One of the most popular listing algorithms for undirected unweighted graphs is the one by [16] which lists all the STs in optimal time, i.e. $O(\alpha+n+m)$ and space $O(nm)$, where α is the number of solutions. On the other hand, using reverse search, the algorithm in [22] has linear space with $O(m+n+\alpha n)$ time. The one in [29] is optimal time and has also linear memory. The algorithm in [30] lists all the STs in increasing order of weights with cost $O(\alpha m \log m + n^2)$ total time and $O(\alpha m)$ space.

[3] Moreover, it has been implicitly adopted in other graph enumeration papers, like [4].

Concerning more specifically weighted graphs, the algorithm in [23] works also for MSTs, as also one of the algorithms in [16] allows to list MSTs in a similar time bound, i.e. $O(\alpha n)$. Other algorithms have been proposed: the algorithm in [35] has cost per solution equal to $O(m \log n)$ and $O(m)$ space. A generalization of the Kruskal Algorithm for listing purposes has been done in [34]. The reduction in [12] allows us to reduce the enumeration of MSTs to STs in a different graph through edge sliding operations, which, using [16] as a subroutine, lists all MSTs in constant amortized time, but not constant delay. This reduction may be combined with the techniques proposed in this paper to obtain an alternative, equivalent, algorithm for listing MSTs with the same delay. To this end, our approach based on Cartesian decomposition is general and can be applied to a vast range of other graphs problems, as long as their set of solutions can be decomposed in some way.

1.2 Preliminaries

We consider an undirected, edge weighted, graph $G = (V(G), E(G))$. $N_G(v)$ represents the neighborhood of v in G. For simplicity, in the following we call $|V(G)| = n$ and $|E(G)| = m$. Whenever G is clear from the context, we may drop subscripts and use simplified notation like V, E, $N(v)$ instead of $V(G)$,$E(G)$, $N_G(v)$.

Let $W = \{w_1, \ldots, w_k\}$, with $w_1 < \cdots < w_k$, be the distinct edge weights of G, where $1 \leq k \leq m$. For a weight w_i, let $E_{<w_i}(G)$, $E_{w_i}(G)$, and $E_{>w_i}(G)$ be the sets of all edges in $E(G)$ which have weight respectively *smaller than* w_i, *equal to* w_i, and *larger than* w_i.

For a given edge $e = \{x, y\}$, we call *contracting* e in G the operation of deleting x and y from G, as well as all edges incident to them, and replacing them with a new node z such that $N(z) = N(x) \dot\cup N(y)$, observing that $N(z)$ can be a multiset (and thus we obtain a multigraph). We refer to the (multi)graph obtained by contracting e in G as G/e. We also implicitly maintain a correspondence between the new edges in G/e and those in G to make the operation reversible. Note that this is not a one-to-one correspondence as two edges $\{v, x\}$ and $\{v, y\}$ will correspond to the same edge $\{v, z\}$ in G/e. Instead, $G \setminus e$ represents the graph obtained by simply deleting e from G (but not its extremes). The operations G/X and $G \setminus X$ are similarly defined for a set of edges $X \subseteq E(G)$, where edges (or their corresponding ones) are respectively contracted or deleted one by one in any order.

Given a set of edges $E' \subseteq E(G)$, $V[E']$ is the set of nodes incident to at least one edge in E'. The subgraph of G *induced* by E' is the graph $G[E'] = (V[E'], E')$. A spanning tree of a connected graph G is a subgraph $T = G[E_T]$, for some E_T such that T is connected, acyclic, and contains all nodes of G, i.e., $V[E_T] = V(G)$. One can also define a spanning tree as a *maximal* set of edges E_T such that the corresponding subgraph $G[E_T]$ is acyclic. Yet another equivalent definition of spanning tree is a *minimal* set of of edges E_T which guarantees connectivity between all pairs of nodes, since any edge that partakes in a cycle may be deleted without affecting the connectivity of the graph [10]. We denote by $\mathcal{T}(G)$ the set of all spanning trees t_i of G.

When G is *not connected*, let $\mathcal{C}(G) = \{C_1, \ldots, C_j\}$ be the set of connected components of G; for simplicity, we define a spanning tree of the non connected graph G as the union of a spanning tree of each connected component. In this case, we can easily see how $\mathcal{T}(G)$ corresponds to the *Cartesian product* among the sets of trees of its connected components, i.e., $\mathcal{T}(C_1) \times \ldots \times \mathcal{T}(C_j)$.

2 Finding Solutions of the Cartesian Product via Gray Coding

In the previous section we have seen that the problem of listing patterns in several cases reduces to combining solutions from other listing subproblems in all the possible ways. This corresponds to our notion of decomposable problem and our goal here is to achieve efficient enumeration of the original problem knowing how to efficiently solve each of the subproblems, as summarized next.

Main Problem. Given h subproblems, whose solutions are sets $S_1, \ldots, S_h$, the solutions of our problem can be seen conceptually as tuples $\langle s_1, \ldots, s_h \rangle \in \mathcal{P} = S_1, \times \ldots \times S_h$. Now suppose that we have a listing algorithm which is able to iterate over the solutions of S_i for each i with polynomial setup time $O(P)$, delay time $O(D)$ and polynomial space, modeled as an iterator A_i which can yield the solutions of S_i one by one after an initial setup.[4] We say that the iterator A_i is *reversible* if there exists an iterator A_i^{-1} which scans the solutions of A_i in the opposite order with the same time and space costs. We will show that, having reversible iterators A_i, it is possible to build an iterator for $\mathcal{P}$, whose delay is the same of A_i.

▶ **Theorem 1.** *Given a reversible iterator A_i for each set of solutions S_i, $1 \leq i \leq h$, running with delay $O(D)$, setup time $O(P)$, and polynomial space, it is possible to list all the solutions in $\mathcal{P} = S_1, \times \cdots \times S_h$ with delay $O(D)$, setup $O(h \cdot P)$ and polynomial space.*

While the polynomial space of an iterator can increase by a factor h in the statement of Theorem 1, we observe that the delay does not depend on h nor includes the setup times. This is non-trivial as we could trivially start running the algorithm A_1 to get S_1, and each time a solution $s_j \in S_1$ is found, recursively start the iterator A_2 to scan all the solutions in S_2. This is not satisfactory as it requires, in the worst case, a delay which can be $O(h \cdot (P + D))$, so that for each solution $s_j \in S_i$ $(1 \leq i < h)$ we have to setup the iterator S_{i+1}. To meet the result in Theorem 1, we need to erase the setup cost from the delay, meaning that for each $s_j \in S_i$, the iterator for S_{i+1} is ready to use without calling a setup function, e.g. `init()` for A_{i+1}. On the other hand, we need to erase also the dependency from h. To overcome this we start from the well known *Gray coding* technique, in a generalized form given by Knuth, in "The Art of Computer Programming" Volume 4 Fascicle 2A [17]. This requires to design forward and backward iterators, i.e. meaning that for each iterator A_i that scans all the solutions of S_i (with $1 \leq i \leq h$) in a certain order we need to design a corresponding iterator, which we call A_i^{-1}, that scans the same solutions in the reversed order.

2.1 Setup costs

Given an iterator A_i, with $O(D)$ delay and a given setup cost of $O(P)$ time (corresponding to the cost of `init()`), the main objective of this section is showing that it is possible to scan the set of solutions S_i an arbitrary number of times, paying the setup cost $O(P)$ only once (the first time A_i is started), and without affecting the delay $O(D)$.

To this aim, first consider the trivial case in which the total execution time of A_i, after the setup is done, is equal or less than the setup cost $O(P)$. This implies that the output is bounded in size by $O(P)$. In this case, we can simply run A_i, and store the complete whole output: this takes $O(P)$ time and space, and clearly allows us to iterate on the solutions of A_i (i.e., the output) any number of times with an equal (or better) delay.[5]

[4] This can be realized for instance providing the well-known `init()` and `getNext()` methods of Java iterators, to respectively initialize and give the next solution for a given iterator.

[5] To recognize this case, we simply run A_i for $O(P)$ steps after the setup, without affecting its cost.

Otherwise, let M be the total amount of data generated by the setup of A_i. This can be anything, such as initialized data structures, pre-processed information on the input, buffered solutions, or other data. Once M is given, then A_i can start without further ado, so our goal is achieved by guaranteeing that an unaltered copy of M is ready at any time to start again.

To obtain this, we keep two copies of M, namely M_a and M_b. Initially these need $O(P)$ time to be computed, and clearly we can restore any of them at any time in $O(P)$ time by executing the setup again. Our strategy works as follows. The first time that A_i is executed it will use the data in M_a. After its execution, M_a may have been altered and may not be usable, but M_b is intact. For the second execution, we will run A_i using M_b, and restore M_a *while* running the algorithm: we perform alternatively one step of A_i and one step of restoring M_a. The next time A_i is run, it can use the data in M_a, and restore M_b while running in the same way. Hence, even executions of the iterator A_i use M_a and restore M_b, while odd ones use M_b while restoring M_a. Note that, since the execution of A_i takes $\Omega(P)$ time (we covered the other case above), M_a (or M_b) will have been fully restored before A_i terminates. As a result, we can start the iterator A_i any number of times, but the setup cost is only paid once in the beginning. Furthermore, this slows down A_i by just a factor of two, so the asymptotic complexity is unchanged.

2.2 Gray Coding via Forward and Backward Iterators

The idea of Gray coding is producing tuples one after the other such that any two consecutive tuples differ by just one entry, i.e. after the output of $\langle s_1, \ldots, s_{j-1}, s_j, s_{j+1} \ldots, s_h \rangle$ we output $\langle s_1, \ldots, s_{j-1}, s'_j, s_{j+1}, \ldots, s_h \rangle$ for some j. A result in [17] was given for explicit sets, but in the following we show how to generalize it for implicit sets, i.e. sets which are results of an iterator. Clearly, to do so, we take into account the delay between the output of s_j and s'_j in A_j, namely $O(D)$ in Theorem 1. The adaptation is not trivial as [17] assumes a certain ability of moving inside the objects in S_j, while a listing algorithm produces solutions in one fixed order.

Among the several variations of Gray coding, we consider *loopless reflected mixed-radix Gray generation* (Algorithm H in [17]), which we refer to as LRMG. This algorithm visits all the tuples changing only one coordinate ± 1 at each step. It maintains an array of focus pointers which says which iterator we have to call at each step, and an array of directions, which for each j (with $1 \leq j \leq h$) says whether we are iterating on S_j from the first solution to the last one or vice versa. Without loss of generality, we can assume $|S_i| > 1$ for each i, as if $|S_i| = 1$ then the only element in S_i is present in all solutions and needs to be output just once at the beginning.

The approach of LRMG can be generalized to deal with the tuples of $\mathcal{P} = S_1 \times \ldots \times S_h$, when the elements of each S_i are implicit, i.e. the result of an iterator. This generalization can be done if the following holds: for each i, on top of the iterator A_i that scans the solutions of S_i in a certain order π, with delay $O(D)$, we can obtain another iterator, called A_i^{-1}, that scans the same solutions in the *opposite* order of π, still with delay $O(D)$. The logic behind this is that the "direction" variable of each iterator is changed only at the end of each iteration, thus we only need to perform complete iteration on S_i in one order and its opposite.

An important remark is that whenever switching from using a certain A_i to the corresponding A_i^{-1}, we should not consider as output the first solution found by A_i^{-1}, since it is the same as the last one output by A_i. The same applies when switching from A_i^{-1} back to A_i. In both cases the delay and preprocessing cost remain the same.

If this is given, we can prove Theorem 1. Indeed, we can plug the suitable iterators A_i and A_i^{-1} into LRMG, and uses the setup cost factorization described in Section 2.1, obtaining an algorithm for iterating the solutions of $\mathcal{P}$ having as delay and setup time cost respectively the sum of the delay and setup time costs of A_i and A_i^{-1}, which asymptotically is the same.

Getting the backward iterator. We now show how to obtain A_i^{-1} given A_i, under suitable conditions which are met by most state-of-the-art listing algorithms. In particular, we assume A_i to be a recursive algorithm with a recursion tree in which each node outputs at most one solution, and generates children as nested recursive calls. Moreover, we assume to be able to generate children of a given recursive call in the opposite order as in A_i, which is generally true for binary partition [5, 4] or reverse search [1, 21] algorithms, where the i-th child call is not influenced by the computation of the subtrees of the previous $i-1$ children.

Let T be the tree induced by the recursive calls. Note that in reverse search algorithms each node in T outputs a solution, while in other backtracking algorithms output may be done in just some nodes, or some of the leaves of T. Consider the following two traversals of T, called FORWARD (resp. BACKWARD), starting from a node u, where d is the depth of the current recursion node in T.

1. If d is even (resp. odd) output u.
2. For each child i of u in increasing (resp. descending) order call FORWARD (u) (resp. BACKWARD (u)).
3. If d is odd (resp. even) output u.

The following observation clearly holds.

▶ **Observation 2.** *For any tree T, let π be the visiting order of the nodes of T obtained by* FORWARD. *Then* BACKWARD *scans the nodes of T in the opposite order with respect to π.*

The above traversal is similar to the so-called *alternative output* in [32], which is usually done to reduce the delay of listing algorithms, but we use this variation to get forward and backward iterators A_i and A_i^{-1} at the same time. In particular, we define A_i and A_i^{-1} as the iterators respectively induced by FORWARD and BACKWARD visit algorithms on T.

Applying Observation 2, we observe that in order to design A_i and A_i^{-1}, we only need a method to scan all the children in each internal node of T in one direction and in the opposite one with the same cost. As a result, both A_i have the same delay $O(D)$.

3 Case Study: Minimum Spanning Trees

In this section we describe how to use our technique to list all the MSTs of a graph, improving the state of the art to get constant delay for STs and applying the technique in Section 2 to get constant delay also for MSTs.

Overview. We will firstly reduce the problem of listing MSTs to the one of listing STs. To this aim, we order the weights of the graph as in the well-known Kruskal Algorithm, $w_1 < \cdots < w_k$, and we solve a STs listing problem for each different w_i, that is finding all the STs in a series of h graphs $C_1, \ldots, C_h$ (with $h \geq k$). For the latter task, on each C_i we run our improvement of the algorithm by [29], which achieves $O(1)$ delay to list STs and is presented in Section 3.2.1. We then need to compose the solutions, so that each combination of solutions for $C_1, \ldots, C_h$ corresponds to a different MSTs. As we have delay $O(1)$ on each C_i (with setup time $O(m^2)$ and space $O(mn)$), our main goal is to preserve these bounds when combining the solutions of $C_1, \ldots, C_h$. Indeed, the usual algorithm would setup an

iterator for C_1, and for each solution of C_1 would setup the iterator for the solutions of C_2 and so on. In this way, the delay can be up to $O(hm^2)$. Our goal is to reduce this delay to $O(1)$, which is the delay of a single iterator, maintaining polynomial space.

We hence apply the techniques in Section 2, adapting listing algorithms to use the Gray coding strategy, basically showing how to build backward iterators from forward iterators, how to factorize setup time costs and use and rebuild auxiliary data structures without afflicting the time complexity.

This problem is related to the more general problem of listing all the tuples of a Cartesian product which is usually solved using Gray coding. However, as in the scenario presented in Section 2, in our case the elements of the tuples are not explicitly given but they are generated through iterators which give solutions in a linear way (it is not possible to jump from one solution to an arbitrary one) and require setup time costs. In this section, we adapt our general solution to deal with the specific case of MSTs. In particular:

- we show how to reduce the problem of listing MSTs to the listing the solutions of a Cartesian product among h sets of solutions, each one corresponding to a set of STs in a graph.
- we show how to list STs with constant delay, as the state of the art algorithm [29] for STs runs in constant amortized time per solution but *linear* delay.
- we show how to use output queue technique to get rid of setup times and pay them just once at the beginning.
- we design *ad hoc* forward and backward iterators for STs, both with constant delay, which help us to successfully applying Gray coding to MSTs, so that the difference between two consecutive listed solutions is constant and can be computed in constant time.

3.1 From Minimum Spanning Trees to Spanning Trees

A well known way to compute a minimum spanning tree is using the greedy algorithm by Kruskal [18], which consists of the following steps:

1. Scan the edges of G according to an increasing weight order.
2. Add an edge to the solution T if it does not create a cycle in T, and discard it otherwise.
3. When all edges have been considered, T is a minimum spanning tree of G.

It is immediately evident how if all edges have different weights they will always be scanned in the same order, thus the algorithm will return the same result. However, if several edges have the same weight, the algorithm may scan them in different orders and produce different trees. Actually, it turns out that *any* minimum spanning tree of G may be found by Kruskal's algorithm, if that the adequate order of the edges is provided. More formally

▶ **Lemma 3.** *Let T be any* MST *of G, and $e_1, \ldots, e_m$ an ordering of $E(G)$ in increasing weight, such that any edge $e_i \in T$ appears in the order* before *any edge not in T which has the* same *weight as e_i. Running Kruskal's algorithm according to this order yields exactly T.*

By shifting our point of view, we can rephrase the algorithm in a more convenient way for our goal, considering the weights $w_1 < \cdots < w_k$ of G. The algorithm is essentially saying that we should greedily add as many edges of E_{w_1} as possible, without creating a cycle, before considering those of E_{w_2}. In other words, we are selecting a *maximal acyclic subgraph* of the graph $G[E_{w_1}]$, that is, a spanning tree T of $G[E_{w_1}]$. Once such a tree (or combination of trees, if $G[E_{w_1}]$ is not connected) has been found, all edges joining two vertices in the same connected component of T may be discarded, as they will create a cycle and not improve

the connectivity. Furthermore, adding to T any edge from a node x to any node in some connected component C of T is equivalent in terms of connectivity.

This means that we can *contract* each C into a single node to obtain a new graph G', and continue to execute Kruskal's algorithm, i.e., considering edges in E_{w_2}, without affecting the outcome. Greedily adding edges of E_{w_2} to T without creating cycles corresponds to selecting a maximal acyclic subgraph (i.e., spanning tree) of $G'[E_{w_2}]$. Again, once the edges of this spanning tree are selected and added to T, we can contract the connected components and connect as many nodes as possible using the edges of E_{w_3}.

A crucial point is that any spanning tree of $G[E_{w_1}]$ will contain exactly the same connected components as any other, corresponding to the connected components of $G[E_{w_1}]$. In other words, $G'[E_{w_2}]$ is always the same, regardless of which spanning tree of $G[E_{w_1}]$ was previously selected. This means that we can abstract the analysis of each weight w_i from all others: indeed, when considering w_i, we can consider as a single node all the nodes connected by edges of weights smaller than w_i, and we must create as many new connections as possible using edges of E_{w_i} without creating cycles. This corresponds to selecting a spanning tree in the graph obtained by contracting all edges of weight smaller than w_i, i.e., $E_{<w_i}$, and then deleting all those of weight greater than w_i; we call this graph G_i, whose formal definition is $G_i = (G/E_{<w_i}) \setminus E_{>w_i}$. We can thus give a recursive definition to the set of all minimum spanning trees of G, i.e. MSTs(G), as:

$$\text{MSTs}(G) = \prod_{w_i \in W} (\mathcal{T}(G_i)) \tag{1}$$

Where we recall that $\mathcal{T}(G_i)$ is the set of all spanning trees of G_i, $\prod$ corresponds to the Cartesian product between the given sets and, if G_i is not connected, we defined its spanning trees as combinations of a spanning tree of each connected components, as follows:

$$\mathcal{T}(G_i) = \prod_{C_j \in \mathcal{C}(G_i)} \mathcal{T}(C_j). \tag{2}$$

From what we said above, as all (and only) the minimum spanning trees of G are obtained by combining a minimum spanning tree of each G_i, for all $1 \leq i \leq k$, we get the following.

▶ **Lemma 4.** *Equation 1 gives all (and only) the minimum spanning trees of G.*

Furthermore, note that all edges in a single G_i have the same weight, so all spanning trees have the same weight as well, and listing minimum spanning trees correspond to just listing all spanning trees. As finding solutions for each G_i is relatively quick, modifying the constant amortized time provided by [29] in constant delay, combining these solutions maintaining the constant delay the same space bound is not trivial. In the following section, we will see that this corresponds to a more general problem, which is finding tuples of a Cartesian product among k sets, with time per solution independent from k and neglecting setup costs, where the elements of these sets are given implicitly, as result of an iterator.

3.2 MWST Gray Coding

In Section 3.1 we showed that we can solve the MST enumeration problem by listing (non weighted) STs in a series of multigraphs. In particular, we have seen that the problem of listing MSTs can be reduced to that of listing STs in several graphs and combining them in all the possible ways. Indeed, each MST in G corresponds to a tuple $\langle t_1, \ldots, t_k \rangle \in \mathcal{T}(G[w_1]) \times \ldots \times \mathcal{T}(G[w_k])$, where $k = |W|$ and $t_i \in \mathcal{T}(G_i)$ is one of the possible combination

of STs of the connected components of G_i according to Equation 2. Let $C_1, \ldots, C_h$ be a sequence containing the connected components of $G[w_1]$, followed by the ones of $G[w_2]$, and so on until those of $G[w_k]$. By combining Equations 1 and 2, we get that a MST is a tuple $\langle s_1, \ldots, s_h \rangle$ where s_i is any spanning tree of C_i, i.e. $s_i \in \mathcal{T}(C_i)$, for $1 \leq i \leq h$. Our goal is to list all the elements in $\mathcal{T}(C_1) \times \ldots \times \mathcal{T}(C_h)$ in an efficient way, given that we know how to list all the solutions in $\mathcal{T}(C_i)$ for each i. In this section, we show how to adapt the state of the art algorithm for listing STs to run in constant delay, and how to combine this resulting algorithm with the techniques in Section 2 to obtain a constant delay algorithm for listing MSTs.

3.2.1 Enumerating Spanning Trees in Constant Time Delay

In the following, we show how to improve the state of the art algorithm for listing STs to get constant delay per solution, proving the following result.

▶ **Lemma 5.** *There exists an algorithm* ST-CDEL *that lists spanning trees with constant delay, $O(m^2 n)$ preprocessing time, and $O(mn)$ space.*

The state of the art algorithm by Shoioura et al. [29], denoted as ST-CAT in the following, lists STs in constant amortized time per solution (CAT) in multigraphs,[6] but its delay is not constant. We will explain in the following how to modify ST-CAT properly to prove Lemma 5.

As each spanning tree has size $O(n)$, it is clearly impossible to output each solution entirely, so ST-CAT outputs just the first solution, and then the *differences* between one solution and the previous one. Combining all these differences, starting from the first solution output by the algorithm until the last differences output, will yield the "current solution" that the algorithm is considering.[7] In particular, using the notation in [29], this is done in the form of *output("$-e_k$, $+g$, tree,")*, meaning that the edge e_k must be deleted and edge g must be added with respect to the previously output solution, and *"tree"* means that a new solution has been reached, i.e., performing all the addition/deletion instructions output so far yields a new solution. However, ST-CAT has two features which cause its delay to be more than constant, namely *output size* and *update costs*. Both of these can be overcome by with proper use of deamortization techniques. We address them separately in the following.

Output size. Each time a recursive call is closed by ST-CAT, the changes done to the output are restored. Namely, each recursive call of the algorithm prints a first *output("$-e_k$, $+g$, tree,")*, then generates some children recursive calls, then prints a second *output("$-g$, $+e_k$,")* to undo these changes. When backtracking, several recursive calls may be terminated at once: in the worst case, up to the depth of the recursion tree, which is $O(n)$ for ST-CAT.

This causes the cumulative changes to be up to $\Omega(n)$, before the next output is performed by printing a "tree" token. Even if the computation time in reaching this output could be ignored, printing this number of changes clearly cannot be done in constant time.

An important property of this structure is that combining the output streams of all algorithms A_i will yield an output stream which iterates over the solutions of G: indeed, a solution of the main problem is a combination of solutions of the subproblems, and each output

[6] For reference, its pseudo code can be found in [29] at page 690.

[7] Reconstructing and outputting the whole solution from the stream would take more than constant time, but since all solutions have size $\Theta(n)$, a constant-delay algorithm for the problem would be impossible with this requirement.

of each A_i changes the solution of exactly one subproblem, we have that the combination of sub-solutions (i.e., the solution of G) will be a new one.

Furthermore, *every* recursive node X of A_i first alters the current solution with the *first* output, and then undoes these changes with the *second* output. Since the same is valid for children nodes of X, it follows that the "current solution" which can be read on the output stream *just before* the second output is equal to the one which can be read *just after* the first one.

In other words, we can chose to make the node X output a solution at the *end* of its execution (i.e., after its children nodes have terminated), rather than at the beginning, by simply printing the "tree" token at the beginning of the *second* output instruction instead of the end of the first. More formally, recursive calls which are tweaked to output the solution at the end will perform the first output call in the form *output("$-e_k$, $+g$,")*, and the second one in the form *output("tree, $-g$, $+e_k$,")*, instead of how described above.

With this property, we can apply the *alternative output* technique from [32]: for each recursive node X in the recursion tree T, we output its solution at the *beginning* of X if its depth is even, and at the *end* if it is odd.

This means that each output is performed after just a constant number of recursive calls has been either generated or closed by the algorithm. Since the changes performed by a single call are constant in size, it follows that the difference between one solution and the previously output one is always of constant size.

Finally, note that the differences between one solution and the next one are the same in the reversed order, thus this applies to the reversed iterator A^{-1} as well, which can be obtained as described in Section 2.2.

Update costs. Each recursive node of ST-CAT has to update some data structure. While the update cost is cleverly amortized to be just $O(1)$ per node, one update can be $O(m)$ in the worst case. We solve this issue using the technique from [32], called *output queue*. The general idea is to accumulate the output in memory, so that it is suitably delayed in a way that the time between the output of two "tree" tokens is constant.

Given the recursion tree T of ST-CAT, let T^* be an upper bound on the cumulative cost of nodes in a root-to-leaf path of T. As reported in [29], the depth of T is $O(n)$ and the maximum cost of a single recursive call is $O(m)$. Thus $T^* = O(mn)$. Furthermore, let $\bar{T}$ be the maximum, among all subtrees T' of T, of the total cost of T' divided by the number of nodes of T' corresponding to a solution. It can be seen that the amortization scheme in [29] that allows ST-CAT to have constant amortized time can be equivalently applied to any subtree, as it only considers the descendants of a recursive node. Thus we have that this ratio is constant for any subtree, and that $\bar{T} = O(1)$.

The output queue Q technique requires us to fill a queue containing $2 \cdot T^*/\bar{T} + 1 = O(mn)$ solutions, i.e. all the output required before printing $O(mn)$ "tree" tokens. By Theorem 2 in [32], the time required to initially fill the queue is bounded by $O(T^* + \bar{T}) = O(mn)$. As for the space, the first solution is output completely, but for all the others we just output the difference with respect to the previous solution, which as constant size, thus the size of the queue is always bounded by $O(mn)$.

As soon as the queue is full, dequeue the top solution (i.e. the set of changes before a "tree" token appear in the queue) and output it. Repeat this step every $O(\bar{T}) = O(1)$ time while the algorithm is executed. Finally, during the execution of the algorithm, each time an output is required, the request is enqueued in Q instead of being output. The output queue guarantees that Q is never empty until the end of the execution, and thus that the algorithm will have in this case $O(1)$ delay after the preprocessing time.

By combining what said so far in this section, we obtain an iterator A, which after a one time preprocessing cost of $O(mn)$, can iterate over all the spanning trees of a given graph with n vertices and m edges, not just once but any number of times without paying for the preprocessing cost again. Due to the memory requirements of the output queue, the space is $O(mn)$.

3.2.2 Forward and Backward Iterators for Spanning Trees

Given the iterator A, we need to show that is possible to obtain the backward iterator A^{-1}, which is the iterator processing the solutions in opposite order with respect to A, still ensuring constant delay. This follows by combining the techniques used above and in Section 2.2. Indeed, consider the design of A for spanning trees given above: if we do not consider the use of the *output queue* technique, this is a recursive algorithm which performs alternative-output and outputs at most one solution per recursion node. Indeed, we can invert the order in which the solutions are output precisely as described in Section 2.2, obtaining an A^{-1} algorithm. Furthermore, it's easy to see that the algorithm so obtained still maintains the same T^* and $\bar{T}$, as well as the same recursive structure, thus we can apply the output queue technique for A^{-1} as well, obtaining the following lemma.

▶ **Lemma 6.** *There exist forward and backward iterators for listing spanning trees with constant delay, $O(mn)$ preprocessing time, and $O(mn)$ space.*

3.2.3 Constant delay algorithm for MSTs

Finally, we present how to use the techniques presented in Sections 3.1, 2 and 3.2 to get a constant delay algorithm for the MSTs listing problem.

Preprocessing. Computing each G_i can clearly be done in $O(m)$ time by performing a BFS to identify the components connected by edges of $E_{<w_i}$, and then adding those of E_{w_i}. Note that, as every edge appears in at most one G_i, the total space for storing these graphs is $O(m)$ and the total computation time is $O(m^2)$.

Furthermore, as a spanning tree has $n-1$ edges, and all minimum spanning trees have the same number of edges of a given weight, there are at most $n-1$ distinct edge weights which can appear in any MST, thus at most n graphs G_i will be non-empty.

For each of these graphs, we must run the preprocessing phase of the ST-CDEL algorithm shown in Section 3.2.1, which takes $O(mn)$ time and uses $O(mn)$ space. Note that this is done twice for each G_i, as we need to start both the forward iterator A_i for the STs of G_i and the corresponding backward iterator A_i^{-1}. The preprocessing phase will thus take $O(m^2 + \sum_{i=1,\dots,h} |V(G_i)| \cdot |E(G_i)|)$ time. The space required to keep track of all the information is $O(\sum_{i=1,\dots,h} |V(G_i)| \cdot |E(G_i)|)$. While these are both trivially bounded by $O(mn^2)$, we can refine this bound: indeed, any edge of G appears in at most one G_i, we have $\sum_{i=1,\dots,h} |E(G_i)| = O(|E(G)|) = O(m)$, and while this is not true for the vertices, we have $|V(G_i)| \leq |V(G)| = n$, meaning that $\sum_{i=1,\dots,h} |V(G_i)| \cdot |E(G_i)| = n \cdot \sum_{i=1,\dots,h} |E(G_i)| = O(mn)$. We thus have that the preprocessing time is $O(m^2 + mn) = O(m^2)$ and the space usage is $O(mn)$.

Algorithm. Once all A_i and A_i^{-1} algorithms are ready, we can use them as iterators on their respective space of solutions, and run the Gray coding algorithm described in Section 2. The Gray coding algorithm alternatively uses A_i and A_i^{-1} depending on the direction set,

which for each i says whether we are iterating from the first solution to the last one or vice versa. The resulting algorithm has constant delay, as we have seen that both when we are using A_i or A_i^{-1} we get a new output with constant delay. Indeed, recall that both A_i and A_i^{-1} dequeue from their corresponding output queue until a "tree" token is found, and the number of dequeues is $O(1)$.

An important remark for A_i, the differences with respect to the previous ST output by A_i turns out to be differences with respect to the previous MSTs of G.

Indeed each output of A_i consists in a difference which turns one ST t'_i of G_i into the next one t''_i (which differ for a constant number of changes). The corresponding solutions for MSTs are tuples $\langle t_1, \ldots, t'_i, \ldots, t_h \rangle$ and $\langle t_1, \ldots, t''_i, \ldots t_h \rangle$ which still differ for a constant number of changes, as t_j corresponds to the same spanning tree for each G_j with $j \neq i$. Similarly, the same applies in the case of A_i^{-1}.

Finally, note that whenever switching from using A_i to A_i^{-1}, we must not output the first solution found by A_i^{-1}, as it corresponds to the last one output by A_i; the same applies when switching from A_i^{-1} to A_i. Clearly, this does not affect the complexity of the algorithm.

As a result, we have proved the following.

▶ **Theorem 7.** *There exists an algorithm which lists all the* MSTs *of a weighted graph with constant delay, $O(m^2)$ preprocessing time, and $O(mn)$ space.*

4 Applications

In this section, we consider several listing problems on graphs which turn out to be suitable for Cartesian decomposition. Some of them can be decomposed by looking at the biconnected components $B_1, \ldots B_h$ of the input graph. Applying Theorem 1, assuming to use forward and backward iterators for each B_i, we get that the delay of iterating over all the solutions of the graph is the maximum delay among the iterators for generating $S_1, \ldots, S_h$, where S_i is the set of solutions associated to B_i.

- ***st*-Paths.** Listing all the st-paths given two nodes s and t in a graph can be done looking at the bead string of biconnected components $B_1, \ldots, B_h$ for some h, with $s \in B_1$ and $t \in B_h$. In particular, let $b_1, \ldots, b_{h-1}$ be the corresponding sequence of articulation points, i.e. b_i is the articulation point between B_i and B_{i+1} $(1 \leq i < h)$. Then all the st-paths are all the tuples in $S_1 \times \ldots \times S_h$ where S_1 is the set of sb_1-paths, S_i is the set of $b_{i-1}b_i$-paths (with $2 \leq i \leq h-1$), S_h is the set of $b_{h-1}t$-paths.
- **Unweighted Spanning Trees.** Given a connected graph G and its biconnected components $B_1, \ldots, B_h$, it is easy to show that all the STs of G correspond to all the tuples in $S_1 \times \ldots \times S_h$, where S_i is the set of STs of B_i.
- **Steiner Trees.** Given a connected graph, all the Steiner trees with set of terminals W can be obtained looking at the biconnected components of G. For each B_i, S_i corresponds to the set of Steiner trees of the graph induced by B_i, fixing as set of terminals the nodes in $W \cap B_i$ and the articulation points x of G such that $x \in B_i$ and x can reach in G some node in W without passing through nodes in B_i.

Other problems on graphs turn out to be decomposable looking at the connected components. Given a graph G, let $Z_1, \ldots, Z_h$ be its connected components. Then the following patterns in G correspond to all the tuples in $S_1 \times \ldots \times S_h$, where S_i is the set of the same kind of patterns just for the graph Z_i. For instance, the following patterns, whose formal definition is given in the corresponding reference, belong to this category.

Maximal Independent Sets (see [7])
Maximal Induced Matching (see [2])
Maximal k-Degenerate Subgraph (see [8])
Minimal Feedback Vertex/Arc Sets (see [28])
Bipartite Subgraphs (see [33])
Bounded Girth Subgraphs (see [9])
Dominating Sets (see [19])
Acyclic Orientations (see [6])

Once again, designing forward and backward iterators as discussed in the previous section for the above problems, and running each of them for each Z_i, we get that the delay of the iterator over the solutions of G has delay equal to the maximum among the delays of the iterators over the solutions of $Z_1, \ldots, Z_h$.

Some other problems are decomposable in a more complex way which clearly is linked to their nature. This is the case of minimum spanning trees, which is object of the case study in Section 3.

5 Conclusions

In this paper we have studied a natural decomposition technique which consists in reducing an enumeration problem to the Cartesian product of the result of several easier sub-problems. We proposed an efficient way to implement this decomposition using a form of Gray coding and forward/backward iterators on the solutions of the sub-problems. In some cases, the sub-problems correspond to smaller instances of the input problem, while in others the problem itself may be a simpler one (e.g., for Minimum Weight Spanning Trees the sub-problem is listing spanning trees of an unweighted graph). We showed an in-depth analysis for the listing of Minimum Weight Spanning Trees, reducing with this technique the known bounds from Constant Amortized Time to Constant Delay. We also gave examples of several other problem which can benefit from this decomposition.

References

1 David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1-3):21–46, 1996.

2 Manu Basavaraju, Pinar Heggernes, Pim van 't Hof, Reza Saei, and Yngve Villanger. Maximal induced matchings in triangle-free graphs. *Journal of Graph Theory*, 83(3):231–250, 2016. doi:10.1002/jgt.21994.

3 Cüneyt F Bazlamaçcı and Khalil S Hindi. Minimum-weight spanning tree algorithms a survey and empirical study. *Computers & Operations Research*, 28(8):767–785, 2001.

4 Etienne Birmelé, Rui Ferreira, Roberto Grossi, Andrea Marino, Nadia Pisanti, Romeo Rizzi, and Gustavo Sacomoto. Optimal listing of cycles and st-paths in undirected graphs. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1884–1896. SIAM, 2013.

5 Coenraad Bron and Joep Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Communications of the ACM*, 16(9):575–576, 1973.

6 Alessio Conte, Roberto Grossi, Andrea Marino, and Romeo Rizzi. Listing acyclic orientations of graphs with single and multiple sources. In *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, pages 319–333, 2016. doi:10.1007/978-3-662-49529-2_24.

7 Alessio Conte, Roberto Grossi, Andrea Marino, Takeaki Uno, and Luca Versari. Listing maximal independent sets with minimal space and bounded delay. In *String Processing and Information Retrieval - 24th International Symposium, SPIRE 2017, Palermo, Italy, September 26-29, 2017, Proceedings*, pages 144–160, 2017. doi:10.1007/978-3-319-67428-5_13.

8 Alessio Conte, Mamadou Moustapha Kanté, Yota Otachi, Takeaki Uno, and Kunihiro Wasa. Efficient enumeration of maximal k-degenerate subgraphs in a chordal graph. In Yixin Cao and Jianer Chen, editors, *Computing and Combinatorics*, pages 150–161, Cham, 2017. Springer International Publishing.

9 Alessio Conte, Kazuhiro Kurita, Kunihiro Wasa, and Takeaki Uno. Listing acyclic subgraphs and subgraphs of bounded girth in directed graphs. In *Combinatorial Optimization and Applications - 11th International Conference, COCOA 2017, Shanghai, China, December 16-18, 2017, Proceedings, Part II*, pages 169–181, 2017. doi:10.1007/978-3-319-71147-8_12.

10 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to algorithms, third edition, 2009.

11 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

12 David Eppstein. *Representing all minimum spanning trees with applications to counting and generation*. Information and Computer Science, University of California, Irvine, 1995.

13 Ira M. Gessel. Enumerative applications of a decomposition for graphs and digraphs. *Discrete Mathematics*, 139(1):257–271, 1995. doi:10.1016/0012-365X(94)00135-6.

14 Ronald L Graham and Pavol Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985.

15 David S Johnson, Mihalis Yannakakis, and Christos H Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

16 Sanjiv Kapoor and Hariharan Ramesh. Algorithms for enumerating all spanning trees of undirected and weighted graphs. *SIAM Journal on Computing*, 24(2):247–265, 1995.

17 Donald E Knuth. The art of computer programming, volume 4, fascicle 2: Generating all tuples and permutations, 2005.

18 Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956. URL: http://www.jstor.org/stable/2033241.

19 Kazuhiro Kurita, Kunihiro Wasa, Hiroki Arimura, and Takeaki Uno. Efficient enumeration of dominating sets for sparse graphs. *CoRR*, abs/1802.07863, 2018. arXiv:1802.07863.

20 F. Maffioli. Complexity of optimum undirected tree problems: a survey of recent results. In *Analysis and design of algorithms in combinatorial optimization*, pages 107–128. Springer, 1981.

21 Kazuhisa Makino and Takeaki Uno. New algorithms for enumerating all maximal cliques. In *Algorithm Theory - SWAT 2004*, pages 260–272, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

22 Tomomi Matsui. An algorithm for finding all the spanning trees in undirected graphs. *METR93-08, Department of Mathematical Engineering and Information Physics, Faculty of Engineering, University of Tokyo*, 16:237–252, 1993.

23 Tomomi Matsui. A flexible algorithm for generating all the spanning trees in undirected graphs. *Algorithmica*, 18(4):530–543, 1997.

24 Matúš Mihalák, Przemysław Uznański, and Pencho Yordanov. Prime factorization of the kirchhoff polynomial: Compact enumeration of arborescences. In *2016 Proceedings of the Thirteenth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 93–105. SIAM, 2016.

25 Reinhard Pichler, Stefan Rümmele, and Stefan Woltran. Counting and enumeration problems with bounded treewidth. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 387–404. Springer, 2010.

26 A. R. Pierce. Bibliography on algorithms for shortest path, shortest spanning tree, and related circuit routing problems (1956–1974). *Networks*, 5(2):129–149, 1975.

27 Ronald C Read and Robert E Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.

28 Benno Schwikowski and Ewald Speckenmeyer. On enumerating all minimal solutions of feedback problems. *Discrete Applied Mathematics*, 117(1-3):253–265, 2002. doi:10.1016/S0166-218X(00)00339-5.

29 Akiyoshi Shioura, Akihisa Tamura, and Takeaki Uno. An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM Journal on Computing*, 26(3):678–692, 1997.

30 Kenneth Sörensen and Gerrit K Janssens. An algorithm to generate all spanning trees of a graph in order of increasing cost. *Pesquisa Operacional*, 25(2):219–229, 2005.

31 Robert E. Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55(2):221–232, 1985. doi:10.1016/0012-365X(85)90051-2.

32 Takeaki Uno. Two general methods to reduce delay and change of enumeration algorithms, 2003. NII Technical Report NII-2003-004E, Tokyo, Japan.

33 Kunihiro Wasa and Takeaki Uno. Efficient enumeration of bipartite subgraphs in graphs. *CoRR*, abs/1803.03839, 2018. arXiv:1803.03839.

34 Perrin Wright. Counting and constructing minimal spanning trees. *Bulletin of the Institute of Combinatorics and its Applications*, 21:65–76, 1997.

35 Takeo Yamada, Seiji Kataoka, and Kohtaro Watanabe. Listing all the minimum spanning trees in an undirected graph. *International Journal of Computer Mathematics*, 87(14):3175–3185, 2010.